ADSP-BF60x Blackfin® Processor Hardware Reference

Preliminary Revision 0.5, February 2013

Part Number
82-100113-01

Analog Devices, Inc.
One Technology Way
Norwood, Mass. 02062-9106

ANALOG DEVICES

Copyright Information

Printed in the USA.

Disclaimer

Trademark and Service Mark Notice

Contents

Contents

Preface

Introduction

System Crossbars (SCB)

Clock Generation Unit (CGU)

System Protection Unit (SPU)

Dynamic Power Management (DPM)

Trigger Routing Unit (TRU)

Static Memory Controller (SMC)

L2 Memory Controller (L2CTL)

Dynamic Memory Controller (DMC)

Cyclic Redundancy Check (CRC)

Direct Memory Access (DMA)

General-Purpose Ports (PORT)

General-Purpose Timer (TIMER)

Watchdog Timer (WDOG)

General-Purpose Counter (CNT)

Pulse-Width Modulator (PWM)

Universal Asynchronous Receiver/Transmitter (UART)

2-Wire Interface (TWI)

Controller Area Network (CAN)

Universal Serial Bus (USB)

Ethernet Media Access Controller (EMAC)

Removable Storage Interface (RSI)

Serial Peripheral Interface (SPI)

Serial Port (SPORT)

ADC Control Module (ACM)

Link Port (LP)

Video Subsystem (VID)

Pipelined Vision Processor (PVP)

Enhanced Parallel Peripheral Interface (EPPI)

Pixel Compositor (PIXC)

Reset Control Unit (RCU)

Boot ROM and Booting the Processor

System Debug Unit (SDU)

System Watchpoint Unit (SWU)

Preface

Thank you for purchasing and developing systems using an enhanced Blackfin processor from Analog Devices, Inc.

Purpose of This Manual

The *ADSP-BF60x Blackfin Processor Hardware Reference* provides architectural information about the ADSP-BF60x 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 programming information, see the *Blackfin Processor Programming Reference.* For timing, electrical, and package specifications, see the *ADSP-BF60x Blackfin Processor 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.

Manual Contents

This manual consists of the following chapters:

- Preface
- Introduction — Provides a high level overview of the processor, including peripherals, power management, and development tools.
- System Cross Bar (SCB) — Describes on-chip buses, including how data moves through the system.
- Clock Generation Unit (CGU) — Describes the phase locked loop (PLL), PLL control unit (PCU), and CGU, which generate on-chip clocks.
- System Protection Unit (SPU) — Describes the system protection and how the SPU protects system resources from errant writes.

- Dynamic Power Management (DPM) — Describes the clocking, including the PLL, and the dynamic power management controller.
- Core Timers (TMR) — Each processor core has its own dedicated timer. The core timer is clocked by the internal processor clock and is typically used as a system tick clock for generating periodic operating system interrupts.
- System Event Controller (SEC) — Describes the system peripheral interrupts, including setup and clearing of interrupt requests.
- Trigger Routing Unit (TRU) — Describes TRU operations providing system-level sequence control without core intervention.
- Static Memory Controller (SMC) — Describes the static (SRAM) memory controller of the processor and the asynchronous memory interface.
- L2 Memory Controller (L2CTL) — Describes the memory controller and its operation.
- Dynamic Memory Controller (DMC) — Describes the dynamic (DDR2) memory controller of the processor, related registers, configuration, and commands.
- Cyclic Redundancy Check (CRC) — Describes the CRC operations on blocks of data presented to the peripheral.
- DMA Channel (DMA) — Describes the peripheral DMA and Memory DMA controllers, including topics such as performance, software management of DMA, and DMA errors.
- General-Purpose Ports (PORTS) — Describes the general-purpose I/O ports, including the structure of each port, multiplexing, configuring the pins, and generating interrupts.
- Pin Interrupts (PINT) — Describes operation of port pins on the processor, which can request interrupts in either an edge-sensitive or a level-sensitive manner with programmable polarity.
- General-Purpose Timer (TIMER) — Describes the general-purpose timers that can be configured in any of three modes; the core timer that can generate periodic interrupts for a variety of timing functions; and the watchdog timer that can implement software watchdog functions, such as generating events to the Blackfin processor core.
- Watchdog Timer (WDOG) — Describes the watchdog timer.
- General-Purpose Counter (CNT) — Describes the Rotary (up/down) Counter. This counter provides support for manually controlled rotary controllers, such as the volume wheel on a radio device. This unit also supports industrial or motor-control type of wheels.
- Pulsewidth Modulator (PWM) — Describes the The PWM controller—a flexible, programmable, three-phase PWM waveform generator that can be programmed to generate the required switching patterns to drive a three-phase voltage source inverter for ac induction motor (ACIM) or permanent magnet synchronous motor (PMSM) control.

- Universal Async Rx/Tx (UART) — Describes the Universal Asynchronous Receiver/Transmitter port that converts data between serial and parallel formats. The UART supports the half-duplex IrDA SIR protocol as a mode-enabled feature.
- Two Wire Interface (TWI) — Describes the Two Wire Interface (TWI) controller, which allows a device to interface to an Inter IC bus as specified by the *Philips I^2 C Bus Specification version 2.1* dated January 2000.
- Controller Area Network (CAN) — Describes the CAN module, a low bit rate serial interface intended for use in applications where bit rates are typically up to 1Mbit/s.
- Universal Serial Bus (USB) — Describes the USB OTG interface of the processor. This interface provides a low-cost connectivity solution for consumer mobile devices such as cell phones, digital still cameras and MP3 players, allowing these devices to transfer data via a point-to-point USB connection without the need for a PC host.
- 10/100 Ethernet MAC (EMAC) — Describes the Ethernet Media Access Controller (MAC) peripheral that provides a 10/100M bit/s Ethernet interface, compliant to IEEE Std. 802.3-2002, between an MII (Media Independent Interface) and the Blackfin peripheral subsystem. Also, describes the IEEE 1588 engine module and the module's operation.
- Removable Storage Interface (RSI) — Describes the RSI interface for multimedia cards (MMC), secure digital memory cards (SD) and secure digital input/output cards (SDIO).
- Serial Peripheral Interface (SPI) — Describes the Serial Peripheral Interface (SPI) port that provides an I/O interface to a variety of SPI compatible peripheral devices.
- Serial Port (SPORT) — Describes the independent, synchronous Serial Port Controller which provides an I/O interface to a variety of serial peripheral devices.
- ADC Control Module (ACM) — Describes the ADC control module (ACM), which provides an interface to synchronize the controls between the processor and an analog-to-digital converter (ADC) module.
- Link Port (LP) — Describes the two bidirectional 8-bit wide link ports, which can connect to other processor or peripheral link ports.
- Video Subsystem (VID) — Describes the subsystem's connectivity matrix that interconnects the video related peripherals.
- Pipelined Vision Processor (PVP) — Describes the PVP's support for co-processing and pre-processing of monochrome video frames in ADAS applications, robotic systems, and other machine applications.
- Parallel Peripheral Interface (PPI) — Describes the Parallel Peripheral Interface (PPI) of the processor. The PPI is a half-duplex, bidirectional port accommodating up to 16 bits of data and is used for digital video and data converter applications.

- Pixel Compositor (PIXC) — Describes the PIXC, which provides data overlay, transparent color, and color space conversion support for active (TFT) flat-panel digital color/monochrome LCD displays or analog NTSC/PAL video output.
- Reset Control Unit (RCU)
- Booting — Describes the booting methods, booting process and specific boot modes for the processor.
- Test Features — Describes test features for the processor, discusses the JTAG standard, boundary-scan architecture, instruction and boundary registers, and public instructions.
  - System Debug Unit (SDU)
  - System Watchpoint Unit (SWU)

What's New in This Manual

This revision (0.5) is a preliminary revision of the *ADSP-BF60x Blackfin Processor Hardware Reference*. This revision revises register information for the ACM and RSI chapters. Register name, bit name, and pin name usage has been updated for consistent usage in all chapters. Also, this revision adds supplementary information and corrections throughout the text.

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)
- Phone questions to *1-800-ANALOGD* (USA only)
- 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.
  One Technology Way
  P.O. Box 9106
  Norwood, MA 02062-9106 USA

Supported Processors

The following is the list of Analog Devices, Inc. processors supported in CrossCore Embedded Studio® development tools.

Blackfin (ADSP-BFxxx) Processors

The name *Blackfin* refers to a family of 16-bit, embedded processors. CrossCore Embedded Studio currently supports the following Blackfin families ADSP-BF50x, ADSP-BF51x, ADSP-BF52x, ADSP-BF53x, ADSP-BF54x, ADSP-BF59x, ADSP-BF561, and ADSP-BF60x processors.

SHARC® (ADSP-21xxx) Processors

The name *SHARC* refers to a family of high-performance, 32-bit, floating-point processors that can be used in speech, sound, graphics, and imaging applications. CrossCore Embedded Studio currently supports the following SHARC families: ADSP-2106x, ADSP-2116x, ADSP-2126x, ADSP-2136x, and ADSP-214xx.

Product Information

Product information can be obtained from the Analog Devices Web site and CrossCore Embedded Studio 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.

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
Close command (**File** menu)	Titles in reference sections indicate the location of an item within the CrossCore Embedded Studio IDE's menu system (for example, the **Close** command appears on the **File** menu).
`{this | that}`	Alternative required items in syntax descriptions appear within curly brackets and separated by vertical bars; read the example as `this` or `that`. 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 `this` or `that`.
`[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 `this`.
`.SECTION`	Commands, directives, keywords, and feature names are in text with `letter gothic` font.
`filename`	Non-keyword placeholders appear in text with italic style format.

Example	Description
NOTE: This is an note paragraph.	**Note:** For correct operation, ... A **Note** provides supplementary information on a related topic. In the online version of this book, the word **Note** appears instead of this symbol.
CAUTION: This is a caution paragraph.	**Caution:** Incorrect device operation may result if ... **Caution:** 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 **Caution** appears instead of this symbol.
DANGER: This is a danger paragraph.	**Danger:** Injury to device users may result if ... A **Danger** identifies conditions or inappropriate usage of the product that could lead to conditions that are potentially hazardous for the devices users. In the online version of this book, the word **Danger** appears instead of this symbol.

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-BF60x processors are members of the Blackfin family of products, incorporating the Analog Devices/Intel Micro Signal Architecture (MSA). Blackfin processors combine a dual-MAC state-of-the-art signal processing engine, the advantages of a clean, orthogonal RISC-like microprocessor instruction set, and single-instruction, multiple-data (SIMD) multimedia capabilities into a single instruction-set architecture.

The processor offers high performance, as well as low static power consumption. Produced with a low-power and low-voltage design methodology, they provide world-class power management and performance.

As shown in the block diagram figure, by integrating a rich set of industry-leading system peripherals and memory, Blackfin processors are 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 systems to embedded industrial, instrumentation and power/motor control applications.

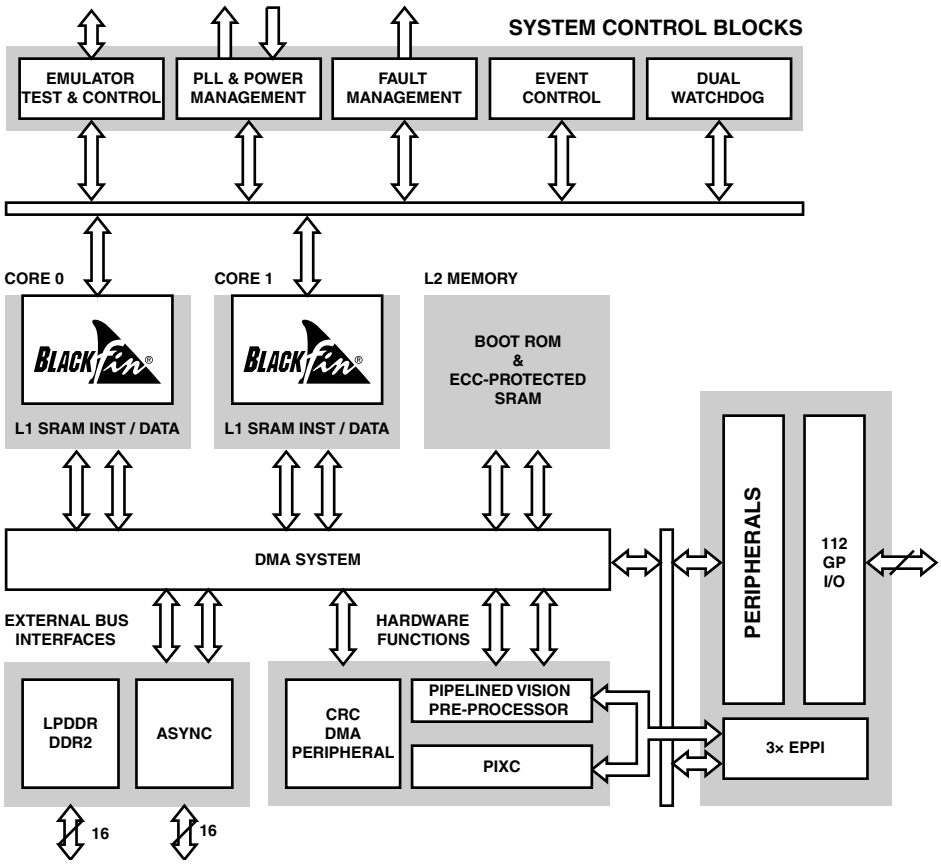


Figure 1-1: ADSP-BF60x Functional Block Diagram

Blackfin Processor Core

As shown in the *Introduction*, the processor integrates two Blackfin processor cores. Each core, shown in Blackfin Processor Core, contains two 16-bit multipliers, two 40-bit accumulators, two 40-bit ALUs, four video ALUs, and a 40-bit shifter. The computation units process 8-, 16-, or 32-bit data from the register file.

The compute register file contains eight 32-bit registers. When performing compute operations on 16-bit operand data, the register file operates as 16 independent 16-bit registers. All operands for compute operations come from the multiported register file and instruction constant fields.

Each MAC can perform a 16-bit by 16-bit multiply in each cycle, accumulating the results into the 40-bit accumulators. Signed and unsigned formats, rounding, and saturation are supported.

The ALUs perform a traditional set of arithmetic and logical operations on 16-bit or 32-bit data. In addition, many special instructions are included to accelerate various signal processing tasks. These include bit operations such as field extract and population count, modulo 2^{32} multiply, divide primitives, saturation and rounding, and sign/exponent detection. The set of video instructions include byte alignment and packing operations, 16-bit and 8-bit adds with clipping, 8-bit average operations, and 8-bit subtract/abso-

lute value/accumulate (SAA) operations. Also provided are the compare/select and vector search instructions.

For certain instructions, two 16-bit ALU operations can be performed simultaneously on register pairs (a 16-bit high half and 16-bit low half of a compute register). If the second ALU is used, quad 16-bit operations are possible.

The 40-bit shifter can perform shifts and rotates and is used to support normalization, field extract, and field deposit instructions.

The program sequencer controls the flow of instruction execution, including instruction alignment and decoding. For program flow control, the sequencer supports PC relative and indirect conditional jumps (with static branch prediction), and subroutine calls. Hardware supports zero-overhead looping. The architecture is fully interlocked, meaning that the programmer need not manage the pipeline when executing instructions with data dependencies.

The address arithmetic unit provides two addresses for simultaneous dual fetches from memory. It contains a multiported register file consisting of four sets of 32-bit index, modify, length, and base registers (for circular buffering), and eight additional 32-bit pointer registers (for C-style indexed stack manipulation).

Blackfin processors support a modified Harvard architecture in combination with a hierarchical memory structure. Level 1 (L1) memories are those that typically operate at the full processor speed with little or no latency. At the L1 level, the instruction memory holds instructions only. The data memory holds data, and a dedicated scratchpad data memory stores stack and local variable information.

In addition, multiple L1 memory blocks are provided, offering a configurable mix of SRAM and cache. The memory management unit (MMU) provides memory protection for individual tasks that may be operating on the core and can protect system registers from unintended access.

The architecture provides three modes of operation: user mode, supervisor mode, and emulation mode. User mode has restricted access to certain system resources, thus providing a protected software environment, while supervisor mode has unrestricted access to the system and core resources.

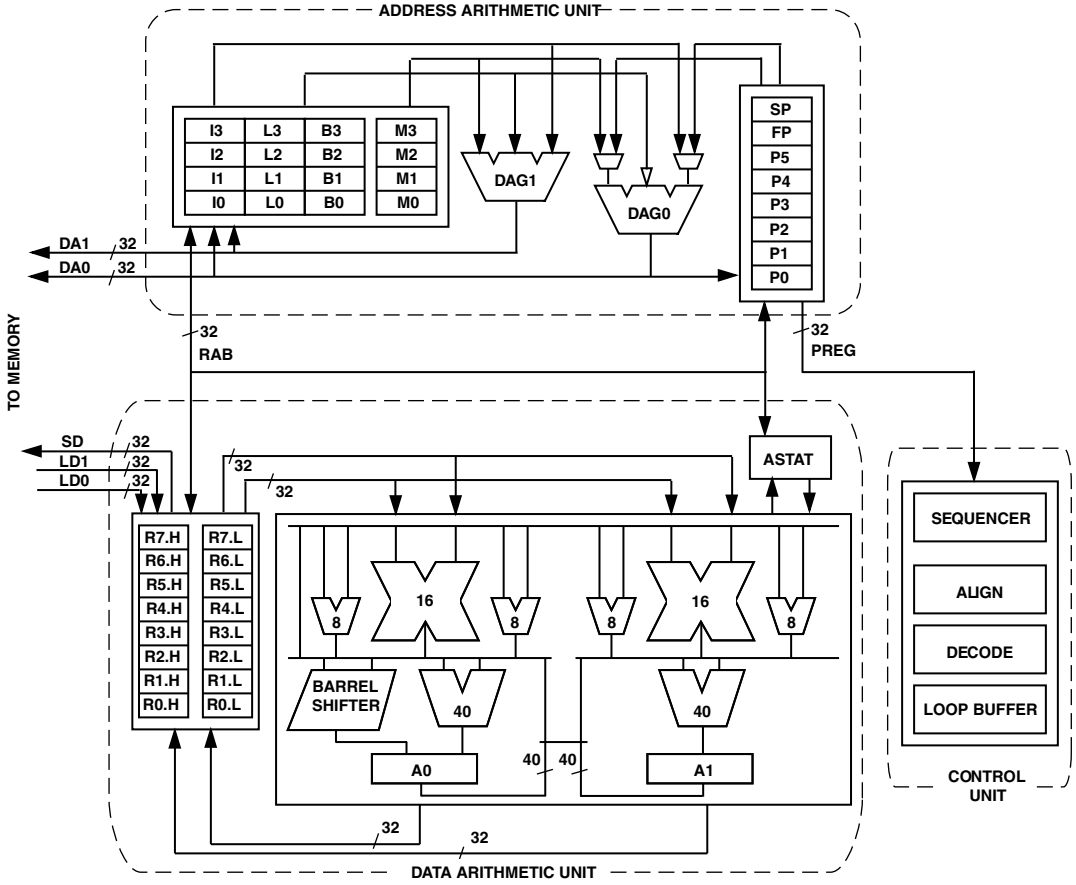


Figure 1-2: Blackfin Processor Core

Instruction Set Description

The Blackfin processor instruction set has been optimized so that 16-bit opcodes represent the most frequently used instructions, resulting in excellent compiled code density. Complex DSP instructions are encoded into 32-bit opcodes, representing fully featured multifunction instructions. Blackfin processors support a limited multi-issue capability, where a 32-bit instruction can be issued in parallel with two 16-bit instructions, allowing the programmer to use many of the core resources in a single instruction cycle.

The Blackfin processor family assembly language instruction set employs an algebraic syntax designed for ease of coding and readability. The instructions have been specifically tuned to provide a flexible, densely encoded instruction set that compiles to a very small final memory size. The instruction set also provides fully featured multifunction instructions that allow the programmer to use many of the processor core resources in a single instruction. Coupled with many features more often seen on micro controllers, this instruction set is very efficient when compiling C and C++ source code. In addition, the architecture supports both user (algorithm/application code) and supervisor (O/S kernel, device drivers, debuggers, ISRs) modes of operation, allowing multiple levels of access to core processor resources.

The assembly language, which takes advantage of the processor's unique architecture, offers the following advantages:

- Seamlessly integrated DSP/MCU features are optimized for both 8-bit and 16-bit operations.
- A multi-issue load/store modified-Harvard architecture, which supports two 16-bit MAC or four 8-bit ALU + two load/store + two pointer updates per cycle.
- All registers, I/O, and memory are mapped into a unified 4G byte memory space, providing a simplified programming model.
- Control of all asynchronous and synchronous events to the processor is handled by two subsystems: the Core Event Controller (CEC) and the System Event Controller (SEC).
- Micro controller features, such as arbitrary bit and bit-field manipulation, insertion, and extraction; integer operations on 8-, 16-, and 32-bit data-types; and separate user and supervisor stack pointers.
- Code density enhancements, which include intermixing of 16-bit and 32-bit instructions (no mode switching, no code segregation). Frequently used instructions are encoded in 16 bits.

NOTE: For more information about the Blackfin instruction set, see the *Blackfin Programming Reference*.

Processor Safety Features

The ADSP-BF60x processor has been designed for functional safety applications. While the level of safety is mainly dominated by the system concept, the following primitives are provided by the devices to build a robust safety concept.

Dual Core Supervision

The processor has been implemented as dual-core devices to separate critical tasks to large independency. Software models support mutual supervision of the cores in symmetrical fashion.

Fault Management

The fault management unit is part of the system event controller (SEC). Any system event, whether a dual-bit uncorrectable ECC error, or any peripheral status interrupt, can be defined as being a "fault". Additionally, the system events can be defined as an interrupt to the cores. If defined as such, the SEC forwards the event to the fault management unit which may automatically reset the entire device for reboot, or simply toggle the SYS_FAULT output pins to signal off-chip hardware. Optionally, the fault management unit can delay the action taken via a keyed sequence, to provide a final chance for the Blackfin cores to resolve the crisis and to prevent the fault action from being taken.

System Protection

All system resources and L2 memory banks can be controlled by either the processor cores, memory-to-memory DMA, or the system debug unit (SDU). A system protection unit (SPU) enables write accesses to specific resources that are locked to any of four masters: Core 0, Core 1, Memory DMA, and the System Debug Unit. System protection is enabled in greater granularity for some modules (L2, SEC and GPIO controllers) through a *global lock* concept.

Bandwidth Monitor

All DMA channels that operate in memory-to-memory mode (Memory DMA, PVP Memory Pipe DMA, PIXC DMA) are equipped with a bandwidth monitor mechanism. They can signal a system event or fault when transactions tend to starve because system buses are fully loaded with higher-priority traffic.

Private (to each Core) Memory

The L1 memory system is the highest-performance memory available to the Blackfin processor cores.

Each core has its own private L1 memory. The modified Harvard architecture supports two concurrent 32-bit data accesses along with an instruction fetch at full processor speed which provides high bandwidth processor performance. Two separate 64K-byte of data memory blocks partner with an 80K-byte memory block for instruction storage. Each block is multi banked for efficient data exchange through DMA and can be configured as SRAM. Alternatively, 16K bytes of each block can be configured in L1 cache mode. The four-way set-associative instruction cache and the 2 two-way set-associative data caches greatly accelerate memory access performance, especially when accessing external memories.

The L1 memory domain also features a 4K-byte scratchpad SRAM block which is ideal for storing local variables and the software stack. All L1 memory is protected by a multi-parity bit concept, regardless of whether the memory is operating in SRAM or cache mode.

Shared (by both Cores) Memory

Outside of the L1 domain, L2 and L3 memories are arranged using a Von Neumann topology. The L2 memory domain is a unified instruction and data memory and can hold any mixture of code and data required by the system design. The L2 memory domain is accessible by both Blackfin cores through a dedicated 64-bit interface. It operates at half the frequency of the cores. The processor features 256K bytes of L2 SRAM which is ECC-protected and organized in eight banks. Individual banks can be made private to any of the cores or the DMA subsystem. There is also a 32K-byte single-bank ROM in the L2 domain. It contains boot code and safety functions.

I/O Memory Space

The processor does not define a separate I/O space. All resources are mapped through the flat 32-bit address space. On-chip I/O devices have their control registers mapped into memory-mapped registers (MMRs) at addresses near the top of the 4G byte address space. These are separated into two smaller blocks, one which contains the control MMRs for all core functions, and the other which contains the registers needed for setup and control of the on-chip peripherals outside of the core. The MMRs are accessible only in supervisor mode and appear as reserved space to on-chip peripherals.

Memory Protection

The Blackfin cores feature a memory protection concept, which grants data and/or instruction accesses from enabled memory regions only. A supervisor mode vs. user mode programming model supports dynamically varying access rights. Increased flexibility in memory page size options supports a simple method of static memory partitioning.

Multi-Parity-Bit-Protected L1 Memories

In the processor's L1 memory space, whether SRAM or cache, each word is protected by multiple parity bits to detect the single event upsets that occur in all RAMs. This applies both to L1 instruction and data memory spaces.

ECC-Protected L2 Memory

Error correcting codes (ECC) are used to correct single event upsets. The L2 memory is protected with a Single Error Correct-Double Error Detect (SEC-DED) code. By default ECC is enabled, but it can be disabled on a per-bank basis. Single-bit errors are transparently corrected. Dual-bit errors can issue a system event or fault if enabled. ECC protection is fully transparent to the user, even if L2 memory is read or written by 8-bit or 16-bit entities.

CRC-Protected Memories

While parity bit and ECC protection mainly protect against random soft errors in L1 and L2 memory cells, the CRC engines can be used to protect against systematic errors (pointer errors) and static content (instruction code) of L1, L2 and even L3 memories (DDR2, LPDDR). The processors feature two CRC engines which are embedded in the memory-to-memory DMA controllers. CRC check sums can be calculated or compared on the fly during memory transfers, or one or multiple memory regions can be continuously scrubbed by single DMA work unit as per DMA descriptor chain instructions. The CRC engine also protects data loaded during the boot process.

Watchpoint Protection

The primary purpose of watchpoints and hardware breakpoints is to serve emulator needs. When enabled, they signal an emulator event whenever user-defined system resources are accessed or a core executes from user-defined addresses. Watchdog events can be configured such that they signal the events to the other Blackfin core or to the fault management unit.

Pin Multiplexing

The processor supports a flexible multiplexing scheme that multiplexes the GPIO pins with various peripherals. A maximum of 4 peripherals plus GPIO functionality is shared by each GPIO pin. All GPIO pins have a bypass path feature - that is, when the output enable and the input enable of a GPIO pin are both active, the data signal before the pad driver is looped back to the receive path for the same GPIO pin.

Processor Infrastructure

The *ADSP-BF60x Functional Block Diagram* shows a number of system control blocks. Some of these blocks provide system control operations, such as event handling and managing the memory sub-system interface. Other of these blocks provide processor core integration (infrastructure), such as bus arbitration, clock generation, and dynamic power management. The following sections provide information on the primary infrastructure components of the ADSP-BF60x processors.

System Crossbar (SCB)

The system crossbars (SCB) appear as buses in the *ADSP-BF60x Functional Block Diagram*. These buses constitute a switch fabric, providing concurrent data access between on-chip bus master and slave memory spaces. Each piece of fabric consists of a matrix for each of master interfaces, and each matrix has a number of slave interfaces. Each matrix with programmable round robin arbitration has a matching number of master slots. Arbitration may be programmed by writing each slot register with a corresponding slave value.

Clock Generation

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 multiple times the CLKIN 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.

Multiplication factors are programmed to the PLLs to define the PLLCLK frequency. Programmable values divide the PLLCLK frequency to generate the core clock (CCLK), the system clocks (SYSCLK, SCLK or SCLK0/1 for multi-core products), the LPDDR or DDR2 clock (DCLK) and the output clock (OCLK). This is illustrated in the Clock Relationships and Divider Values figure.

Writing to the CGU control registers does not affect the behavior of the PLL immediately. Registers are first programmed with a new value, and the PLL logic executes the changes so that it transitions smoothly from the current conditions to the new ones.

The SYS_CLKIN pin oscillations start when power is applied to the VDD_EXT pins. The rising edge of SYS_HWRST can be applied as soon as all voltage supplies are within specifications, and SYS_CLKIN oscillations are stable.

A SYS_CLKOUT output pin has programmable options to output divided-down versions of the on-chip clocks. By default, the SYS_CLKOUT pin drives a buffered version of the SYS_CLKIN input. Clock generation faults (for example PLL unlock) may trigger a reset by hardware.

Crystal Oscillator (SYS_XTAL)

The processor can be clocked by an external crystal, a sine wave input, or a buffered, shaped clock derived from an external clock oscillator. If an external clock is used, it should be a TTL compatible signal and must not be halted, changed, or operated below the specified frequency during normal operation. This signal is connected to the processor's SYS_CLKIN pin. When an external clock is used, the SYS_XTAL pin must be left unconnected. Alternatively, because the processor includes an on-chip oscillator circuit, an external crystal may be used.

Clock Out/External Clock

The SYS_CLKOUT pin can be used to output one of several different clocks used on the processor. The clocks shown in Clock Sources and Dividers can be outputs from SYS_CLKOUT.

NOTE: In the following table *SCLK0* and *SCLK1* are used. However these clocks are referred to as *SCLK* in this manual.

Table 1-1: Clock Sources and Dividers

Clock Source	Divider
CCLK (core clock)	By 4
SYSCLK (System clock)	By 2
SCLK0 (system clock for PVP, all peripherals not covered by SCLK1)	None
SCLK1 (system clock for SPORTS, SPI, ACM)	None
DCLK (LPDDR/DDR2 clock)	By 2
OCLK (output clock)	Programmable
CLKBUF	None, direct from SYS_CLKIN

System Protection Unit (SPU)

The system protection unit (SPU) provides features that let you protect system resources from errant writes. A number of protection categories (types of registers to protect) are available.

Dynamic Power Management (DPM)

The dynamic power management (DPM) feature of the processor lets you control the processor's core clock frequency (f_{CCLK}) dynamically.

As shown in clock relationships and divider values figure, the processor supports four different power domains, which maximizes flexibility while maintaining compliance with industry standards and conventions. By isolating the internal logic of the processor into its own power domain, separate from other I/O, the processor can take advantage of dynamic power management without affecting the other I/O devices.

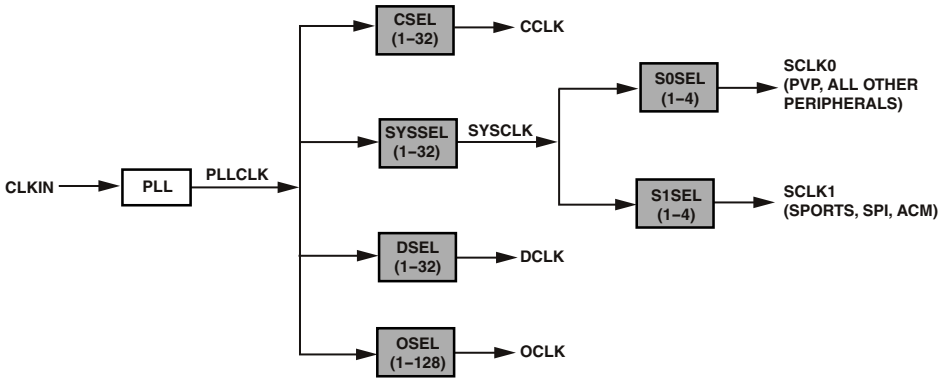


Figure 1-3: Clock Relationships and Divider Values

There are no sequencing requirements for the various power domains, but all domains must be powered according to the appropriate Power Domains and Ranges table for processor operating conditions; even if the feature/peripheral is not used.

Table 1-2: Power Domains and Ranges

Power Domain	VDD Range
All internal logic	V_{DD_INT}
DDR2/LPDDR	V_{DD_DDR}
USB	V_{DD_USB}
All other I/O	V_{DD_EXT}

The power dissipated by a processor is largely a function of its clock frequency and the square of the operating voltage. For example, reducing the clock frequency by 25% results in a 25% reduction in dynamic power dissipation.

Core Timers

Each processor core has its own dedicated timer. The core timer is clocked by the internal processor clock and is typically used as a system tick clock for generating periodic operating system interrupts.

Event Handling

The processor provides event handling that supports both nesting and prioritization. Nesting allows multiple event service routines to be active simultaneously. Prioritization ensures that servicing of a higher-priority event takes precedence over servicing of a lower-priority event. The processor provides support for five different types of events:

- Emulation - An emulation event causes the processor to enter emulation mode, allowing command and control of the processor via the JTAG interface.
- Reset - This event resets the processor.
- Non maskable Interrupt (NMI) - The NMI event can be generated either by the software watchdog timer, by the `NMI` input signal to the processor, or by software. The NMI event is frequently used as a power-down indicator to initiate an orderly shutdown of the system.
- Exceptions - Events that occur synchronously to program flow (in other words, the exception is taken before the instruction is allowed to complete). Conditions such as data alignment violations and undefined instructions cause exceptions.
- Interrupts - Events that occur asynchronously to program flow. They are caused by input signals, timers, and other peripherals, as well as by an explicit software instruction.

The *ADSP-BF60x Functional Block Diagram* shows a number of system control blocks. Some of these blocks provide system control operations, such as event handling and managing the memory sub-system interface. The following sections provide information on the event handling components of the ADSP-BF60x processors.

Core Event Controller (CEC)

The CEC supports nine general-purpose interrupts (IVG15-7), in addition to the dedicated interrupt and exception events. Of these general-purpose interrupts, the two lowest-priority interrupts (IVG15-14) are recommended to be reserved for software interrupt handlers. For more information about general-purpose interrupts, see the *Blackfin Processor Programmer's Reference*.

System Event Controller (SEC)

The SEC manages the enabling, prioritization, and routing of events from each system interrupt or fault source. Additionally, it provides notification and identification of the highest priority active system interrupt request to each core and routes system fault sources to its integrated fault management unit.

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. 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

Pin Interrupts

Every port pin on the processor can request interrupts in either an edge-sensitive or a level-sensitive manner with programmable polarity. Interrupt functionality is decoupled from GPIO operation. Six system-level interrupt channels (PINT0-5) are reserved for this purpose. Each of these interrupt channels can manage up to 32 interrupt pins. The assignment from pin to interrupt is not performed on a pin-by-pin basis. Rather, groups of eight pins (half ports) can be flexibly assigned to interrupt channels.

Every pin interrupt channel features a special set of 32-bit memory-mapped registers that enable half-port assignment and interrupt management. This includes masking, identification, and clearing of requests. These registers also enable access to the respective pin states and use of the interrupt latches, regardless of whether the interrupt is masked or not. Most control registers feature multiple MMR address entries to write-one-to-set or write-one-to-clear them individually.

Memory Architecture

The ADSP-BF60x processor views memory as a single unified 4G byte address space, using 32-bit addresses. All resources, including internal memory, external memory, and I/O control registers, occupy separate sections of this common address space. The memory portions of this address space are arranged in a hierarchical structure to provide a good cost/performance balance of some very fast, low-latency core-accessible memory as cache or SRAM, and larger, lower-cost and performance interface-accessible memory systems.

The processor block diagram shows a number of system control blocks. Some of these blocks provide system control operations, such as event handling and managing the memory sub-system interface. The following sections provide information on managing the memory sub-system interface of the ADSP-BF60x processors.

See the product specific data sheet for the proper external and internal memory configurations.

Static Memory Controller (SMC)

The static memory controller (SMC) is a protocol converter and data transfer interface between the internal processor bus and the external L3 memory.

The SMC can be programmed to control up to four banks of external memories or memory-mapped devices, with very flexible timing parameters. Each bank occupies a 64M byte segment regardless of the size of the device used, so that these banks are only contiguous if each is fully populated with 64M bytes of memory.

SMC acts as an SCB slave and accesses to SMC are arbitrated by the processor SCB interconnect fabric. On the chip boundary, the SMC is connected to a 25-bit external memory address bus, a 16-bit data bus and memory control signal pins (read, write) including four chip selects via the chip pads. The SMC can support 64 MB of external memory connected to four different banks—each bank is controlled by the chip select signal.

L2 Memory Controller

The L2 memory controller is an SCB slave with multiple SCB ports that access banks of RAM and/or ROM. These interfaces and the memory subsystem run at the`L2CLK` frequency. The L2 memory controller also has a bus interface to provide access to the L2 controller's control and status registers. Each bank of L2 memory is protected by ECC (error correction and control) logic which ensures that single bit errors can be corrected and multi-bit errors can be detected. For each 32KB bank, a 4Kx14 bits ECC RAM stores the parity bits.

Dynamic Memory Controller (DMC)

The dynamic memory controller (DMC) provides a glue-less interface between DDR2/LPDDR SDRAMs and the system crossbar interface (SCB) on-chip interconnect. This controller supports JESD79-2E compatible double data rate (DDR2) SDRAM and JESD209A low power DDR (LPDDR) SDRAM devices. The DMC enables execution of instructions from (as well as transfer of data to-and-from) external DDR2 SDRAM and external LPDDR SDRAM.

The DMC supports access to the external memory by core and DMA accesses. The external memory address space is divided in to four banks.

The DMC is partitioned in a manner that allows con figuration and maintainability. The memory access protocol state machine along with JEDEC standard specific logic is embedded in the protocol controller. An access and operation re-ordering 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 controller can be accessed using the MMR access bus.

Cyclic Redundancy Check (CRC)

The cyclic redundancy check (CRC) peripheral performs the CRC operation of the block of data that is presented to the peripheral. The peripheral provides a means to periodically verify the integrity of the system memory, the contents of memory-mapped registers (MMRs), or communication message objects.

The CRC is a hardware module based on a CRC32 engine that computes the CRC value of the 32-bit data words presented to it. Data is provided by the source channel of the memory-to-memory DMA (in memory scan mode) and is optionally forwarded to the destination channel (memory transfer mode). The main features of the CRC peripheral are:

- Memory scan, memory transfer, data verify, and data fill modes
- User-programmable CRC32 polynomial
- Bit/byte mirroring option (endianness)
- Fault/error interrupt mechanisms
- 1D and 2D fill block to initialize array with constants.
- 32-bit CRC signature of a block of a memory or MMR block.

The dedicated hardware compares the calculated signature of the operation to a pre-loaded expected signature and if the two fail to match, the peripheral generates an error.

Data may be provided by the source channel of the memory-to-memory DMA channels and optionally forwarded to memory via the destination DMA channel. Alternatively, the peripheral also supports data presented by core write transactions.

The CRC peripheral implements a reduced table-lookup algorithm to compute the signature of the data. A programmable 32-bit CRC polynomial is used to automatically generate the lookup table (LUT) contents.

Additional 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.

The two CRC protection modules allow system software to periodically calculate the signature of code and/or data in memory, the content of memory-mapped registers, or communication message objects. Dedicated hardware circuitry compares the signature with pre calculated values and triggers appropriate fault events.

Direct Memory Access (DMA)

The direct memory access (DMA) channels are dispersed throughout the infrastructure and may be clustered together over SCBs, sharing a single interface with the main system crossbar.

The processor uses 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 controller carries out the data transfers independent of processor activity.

The DMA channels can perform transfers between memory and a peripheral or between one memory and another memory. Two DMA channels are required for memory to memory DMA transfers (MDMA). One channel is the source channel, and the second, the destination channel.

The DMA channel does not connect external memories and devices directly. Rather, data is passed through an external memory interface port. Any kind of device that is supported by the external memory interface can also be accessed by DMA operation. This is typically flash memory, SRAM, DDR SDRAM, FIFOs, or memory-mapped peripheral devices.

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 processor 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.

The DMA controller supports the following DMA operations.

- A single linear buffer that stops on completion.
- A linear buffer with negative, positive or zero stride length.
- A circular, auto-refreshing buffer that interrupts when each buffer becomes full.
- A similar buffer that interrupts on fractional buffers (for example, 1/2, 1/4).
- 1D DMA - uses a set of identical ping-pong buffers defined by a linked ring of two-word descriptor sets, each containing a link pointer and an address.
- 1D DMA - uses a linked list of 4 word descriptor sets containing a link pointer, an address, a length, and a configuration.
- 2D DMA - uses an array of one-word descriptor sets, specifying only the base DMA address.
- 2D DMA - uses a linked list of multi-word descriptor sets, specifying everything.

On Chip Peripherals

The processor contains a set of on chip peripherals connected to the core over several high-bandwidth buses. These system interface peripherals provide flexibility in system configuration and system performance (see the *ADSP-BF60x Functional Block Diagram*). The following sections describe the on chip peripherals that provide the system interface.

General-Purpose I/O (GPIO)

The general-purpose I/O (GPIO) ports provide the following functions.

- Pin multiplexing scheme
- GPIO functionality
- Pin interrupts

The GPIO port pins can be individually controlled using the port control, status, and interrupt registers. These register let you:

- Specify the direction of each individual GPIO pin as input or output
- Use the "write one to modify" mechanism modify any combination of individual GPIO pins in a single instruction, without affecting the level of any other GPIO pins.
- Treat each individual GPIO pin as an interrupt to the processor; GPIO pins defined as inputs can be configured to generate hardware interrupts, while output pins can be triggered by software interrupts
- Specify whether individual pins are level- or edge-sensitive and specify (if edge-sensitive) whether just the rising edge or both the rising and falling edges of the signal are significant

General-Purpose Timers

The general-purpose timer provides eight general-purpose programmable timers. Each timer has an external pin that can be configured either as a pulse width modulator (PWM) or timer output, as an input to clock the timer, or as a mechanism for measuring pulse widths and periods of external events. These timers can be synchronized to an external clock input on the`TMRx` pins, an external clock`TMRCLK` input pin, or to the internal SCLK.

Additionally, a variety of interrupts can be generated upon completion of timer events. Moreover, GP timers can act both as trigger masters and trigger slaves.

The timer units can be used in conjunction with the UARTs and the CAN controller to measure the width of the pulses in the data stream to provide a software auto-baud detect function for the respective serial channels.

Watchdog Timers

Each core includes a 32-bit timer, which may be used to implement a software watchdog function. A software watchdog can improve system availability by forcing the processor to a known state, via generation of a hardware reset, non maskable interrupt (NMI), or general-purpose interrupt, if the timer expires before being reset by software. The programmer initializes the count value of the timer, enables the appropriate interrupt, then enables the timer. Thereafter, the software must reload the counter before it counts to zero from the programmed value. This protects the system from remaining in an unknown state where

software, which would normally reset the timer, has stopped running due to an external noise condition or software error.

After a reset, software can determine if the watchdog was the source of the hardware reset by interrogating a status bit in the timer control register, which is set only upon a watchdog generated reset.

General-Purpose Counters

A 32-bit counter is provided that can operate in general-purpose up/down count modes and can sense 2-bit quadrature or binary codes as typically emitted by industrial drives or manual thumbwheels. Count direction is either controlled by a level-sensitive input pin or by two edge detectors.

A third counter input can provide flexible zero marker support and can alternatively be used to input the push-button signal of thumb wheels. All three pins have a programmable debouncing circuit.

Internal signals forwarded to each general-purpose timer enable these timers to measure the intervals between count events. Boundary registers enable auto-zero operation or simple system warning by interrupts when programmable count values are exceeded.

Using this feature, you can convert pulses from incremental position encoders into data that is representative of the actual position by integrating (counting) pulses on one or two inputs. Because integration provides relative position, some devices also feature a zero position input (zero marker) that can be used to establish a reference point to verify that the acquired position does not drift over time. The incremental position information also can be used to determine speed, if the time intervals are measured. The GP counter provides flexible ways to establish position information. When used in conjunction with the GP timer block, the GP counter lets you acquire coherent position and time-stamp information that enables speed calculation.

Pulsewidth Modulator (PWM)

Each pulsewidth modulator (PWM) block integrates a flexible and programmable 3-phase PWM waveform generator that can be programmed to generate the required switching patterns to drive a 3-phase voltage source inverter for ac induction motor (ACIM) or permanent magnet synchronous motor (PMSM) control. In addition, the PWM block contains special functions that considerably simplify the generation of the required PWM switching patterns for control of the electronically commutated motor (ECM) or brushless dc motor (BDCM). Software can enable a special mode for switched reluctance motors (SRM). The two 3-phase PWM generation units each feature:

- 16-bit center-based PWM generation unit
- Programmable PWM pulse width
- Single/double update modes
- Programmable dead time and switching frequency
- Twos-complement implementation which permits smooth transition to full ON and full OFF states

- Dedicated asynchronous PWM shutdown signal

The eight PWM output signals (per PWM unit) consist of four high-side drive signals and four low-side drive signals. The polarity of a generated PWM signal can be set with software, so that either active HI or active LO PWM patterns can be produced.

Pulses synchronous to the switching frequency can be generated internally and output on the PWM_SYNC pin. The PWM unit can also accept externally generated synchronization pulses through PWM_SYNC.

Each PWM unit features a dedicated asynchronous shutdown pin which (when brought low) instantaneously places all six PWM outputs in the OFF state.

Universal Asynchronous Receiver/Transmitter (UART)

The processors provide two full-duplex universal asynchronous receiver/transmitter (UART) ports, which are fully compatible with PC-standard UARTs. Each UART port provides a simplified UART interface to other peripherals or hosts, supporting full-duplex, DMA-supported, asynchronous transfers of serial data.

A UART port includes support for five to eight data bits, and none, even, or odd parity. Optionally, an additional address bit can be transferred to interrupt only addressed nodes in multi-drop bus (MDB) systems. A frame is terminates by one, one and a half, two or two and a half stop bits. The UARTs also include interrupt-handling hardware. Interrupts can be generated from multiple events.

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

Partial modem status and control functionality is supported by the UART module to allow for hardware flow control. The UART ports support automatic hardware flow control through the Clear To Send (CTS) input and Request To Send (RTS) output with programmable assertion FIFO levels.

To help support the Local Interconnect Network (LIN) protocols, a special command causes the transmitter to queue a break command of programmable bit length into the transmit buffer. Similarly, the number of stop bits can be extended by a programmable inter-frame space.

The capabilities of the UARTs are further extended with support for the Infrared Data Association (IrDA) serial infrared physical layer link specification (SIR) protocol.

The UARTs are DMA-capable peripherals with support for separate transmit and receive DMA master channels. They can be used in either DMA or programmed core mode of operation. 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. Each UART has its own separate transmit and receive DMA channels.

One of the peripheral timers can be used to provide a hardware-assisted auto-baud detection mechanism for use with the UART. The timers are external to the UART.

2-Wire Interface (TWI)

The processors include a 2-wire interface (TWI), providing a simple exchange method of control data between multiple devices. The TWI module is compatible with the widely used I^2C bus standard. 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 utilizes 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 5 V logic levels.

Additionally, the TWI module is fully compatible with serial camera control bus (SCCB) functionality for easier control of various CMOS camera sensor devices.

To preserve processor bandwidth, the TWI module can be set up with transfer initiated interrupts only to service FIFO buffer data reads and writes. Protocol related interrupts are optional. The TWI externally moves 8-bit data while maintaining compliance with the I^2C bus protocol.

Controller Area Network (CAN)

The processor includes a controller area network (CAN) module, which implements the CAN 2.0B (active) protocol. This protocol is an asynchronous communications protocol used in both industrial and automotive control systems. The CAN protocol is well suited for control applications due to its capability to communicate reliably over a network. This is because the protocol incorporates CRC checking, message error tracking, and fault node confinement.

NOTE: This document assumes reader familiarity with the CAN standard. For more information, refer to Version 2.0 of the CAN Specification from Robert Bosch GmbH.

The CAN module provides the following features:

- 32 mailboxes (8 receive only, 8 transmit only, 16 configurable for receive or transmit)
- Dedicated acceptance masks for each mailbox
- Additional data filtering on first two bytes.
- Support for both the standard (11-bit) and extended (29-bit) identifier (ID) message formats
- Support for remote frames
- Active or passive network support
- CAN wakeup from hibernation mode (lowest static power consumption mode)
- Interrupts, including: TX complete, RX complete, error and global

NOTE: An additional crystal is not required to supply the CAN clock, as the CAN clock is derived from a system clock through a programmable divider.

Universal Serial Bus (USB)

The processor's universal serial bus (USB) module is a USB 2.0 compliant, dual-role device controller. This interface provides a low-cost connectivity solution for the growing adoption of this bus standard in industrial applications and consumer mobile devices, such as cell phones, digital still cameras, and MP3 players. The USB module lets these devices transfer data using a point-to-point USB connection without the need for a PC host.

The USB controller can operate in a traditional USB peripheral-only mode as well as the host mode presented in the on-the-go (OTG) supplement[1]to the USB 2.0 Specification[2]. In host mode, the USB module supports transfers at high-speed (480Mbps), full-speed (12Mbps), and low-speed (1.5Mbps) rates. Peripheral mode supports the high- and full-speed transfer rates.

The USB controller uses a slave bus interface to access its control and status registers as well as read and write to the endpoint packet buffers. Data is transferred to and from the USB controller through any of the transmit and receive endpoint FIFOs. A DMA bus master interface provides numerous DMA channels to provide a more efficient means of transferring large amounts of data between the controller and the Blackfin processor's memory map.

The USB clock (USB_CLKIN) is provided through a dedicated external crystal or crystal oscillator. Using an included phase locked loop with programmable multipliers, the USB on-the-go dual-role device controller generates the necessary internal clocking frequency for USB.

Ethernet Media Access Controller (MAC)

The processor can directly connect to a network by way of an embedded fast Ethernet media access controller (MAC) that supports both 10-BaseT (10M bits/sec) and 100-BaseT (100M bits/sec) operation. The 10/100 Ethernet MAC peripheral on the processor is fully compliant to the IEEE 802.3-2002 standard and it provides programmable features designed to minimize supervision, bus use, or message processing by the rest of the processor system.

Some standard Ethernet MAC features are:

- Support and RMII protocols for external PHYs
- Full duplex and half duplex modes
- Media access management (in half-duplex operation)
- Flow control
- Station management: generation of MDC/MDIO frames for read-write access to PHY registers

1.On-The-Go Supplement to the USB 2.0 Specification, Rev 1.0a; June 24, 2003; USB-IF
2.Universal Serial Bus Specification 2.0

Some advanced Ethernet MAC features are:

- Automatic checksum computation of IP header and IP payload fields of Rx frames
- Independent 32-bit descriptor-driven receive and transmit DMA channels
- Frame status delivery to memory through DMA, including frame completion semaphores for efficient buffer queue management in software
- Tx DMA support for separate descriptors for MAC header and payload to eliminate buffer copy operations
- Convenient frame alignment modes
- 47 MAC management statistics counters with selectable clear-on-read behavior and programmable interrupts on half maximum value
- Advanced power management
- Magic packet detection and wakeup frame filtering
- Support for 802.3Q tagged VLAN frames
- Programmable MDC clock rate and preamble suppression

The Ethernet MAC includes support for the IEEE 1588 standard. This standard is a precision clock synchronization protocol for networked measurement and control systems. The processor includes hardware support for IEEE 1588 with an integrated precision time protocol synchronization engine (PTP_TSYNC). This engine provides hardware assisted time stamping to improve the accuracy of clock synchronization between PTP nodes.

The main IEEE 1588 standard features of the engine are:

- Support for both IEEE 1588-2002 and IEEE 1588-2008 protocol standards
- Hardware assisted time stamping capable of up to 12.5 ns resolution
- Lock adjustment
- Automatic detection of IPv4 and IPv6 packets, as well as PTP messages
- Multiple input clock sources (SCLK, RMII clock, external clock)
- Programmable pulse per second (PPS) output
- Auxiliary snapshot to time stamp external events

Removable Storage Interface (RSI)

The removable storage interface (RSI) controller acts as the host interface for multimedia cards (MMC), secure digital memory cards (SD) and secure digital input/output cards (SDIO). The following list describes the main features of the RSI controller.

- Support for a single MMC, SD memory or SDIO card
- Support for 1-bit and 4-bit SD modes
- Support for 1-bit, 4-bit, and 8-bit MMC modes
- Support for eMMC 4.3 embedded NAND flash devices
- A ten-signal external interface with clock, command, and up to eight data lines
- Card interface clock generation from SCLK
- SDIO interrupt and read wait features

Serial Peripheral Interface (SPI)

The processors have two serial peripheral interface (SPI) compatible ports that allow the processor to communicate with multiple SPI compatible devices.

In its simplest mode, the SPI interface uses three pins for transferring data: two data pins (Master Output-Slave Input, MOSI, and Master Input-Slave Output, MISO) and a clock pin (serial clock, SCK). An SPI chip select input pin (`SPISS`) lets other SPI devices select the processor, and seven SPI chip select output pins (`SPISEL7-1`) let the processor select other SPI devices. The SPI select pins are reconfigured general-purpose I/O pins. Using these pins, the SPI port provides a full-duplex, synchronous serial interface, which supports both master/slave modes and multi master environments.

The SPI port's baud rate and clock phase/polarities are programmable, and it has integrated DMA channels for both transmit and receive data streams.

Serial Port (SPORT)

The processor includes three synchronous serial ports (SPORTs) that provide an inexpensive interface to a wide variety of digital and mixed-signal peripheral devices such as the AD183x family of audio CODECs, ADCs, and DACs from Analog Devices. The SPORTs consist of two data lines, a clock, and frame sync. The data lines can be programmed to either transmit or receive and each data line has a dedicated DMA channel.

SPORT data can be automatically transferred to and from on-chip memory/external memory over dedicated DMA channels. Each of the SPORTs can work in conjunction with another SPORT to provide TDM support. In this configuration, one SPORT provides two transmit signals while the other SPORT provides the two receive signals. The frame sync and clock are shared.

Serial ports may operate in any of the following modes:

- Standard DSP serial mode
- Multichannel (TDM) mode
- I^2S mode
- Packed I^2S mode
- Left-justified mode

ADC Control Module (ACM)

The processor includes an ADC control module (ACM) that provides an interface that synchronizes the controls between the processor and an analog-to-digital converter (ADC). The analog-to-digital conversions are initiated by the processor, based on external or internal events.

The ACM permits flexible scheduling of sampling instants and provides precise sampling signals to the ADC.

The ADC, ACM, and SPORT Connections diagram shows how to connect an external ADC to the ACM and one of the three SPORTs.

The ACM synchronizes the ADC conversion process, generating the ADC controls, the ADC conversion start signal, and other signals. The actual data acquisition from the ADC is done by a peripheral, such as a SPORT or a SPI.

The processor interfaces directly to many ADCs without any glue logic required.

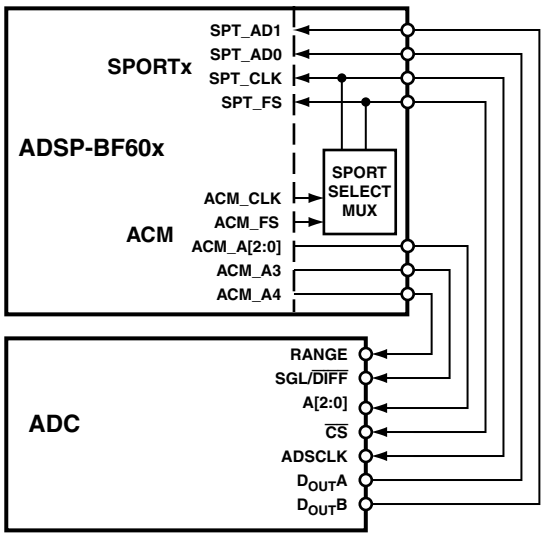


Figure 1-4: ADC, ACM, and SPORT Connections

Link Port (LP)

Four DMA-enabled, 8-bit-wide link ports can connect to the link ports of other Blackfin processors or other link-enabled processors. Link ports are bi-directional ports with eight data lines, an acknowledge line, and a clock line.

Video Sub-System and Pixel Pipeline (PxP)

The processor's video sub-system provides a connectivity matrix that interconnects the video related peripherals. The pixel pipeline (PxP) protocol is used for data transfer among these video peripherals.

- Three enhanced parallel peripheral interface (PPI) ports
- The pixel compositor (PIXC)
- The pipelined vision processor (PVP)
- The video interconnect (bus) and pixel pipeline (PxP)
- 18 DMA channels

The PPIs can operate in video input and video output modes, and also support several general-purpose modes of operation. These parallel ports are aware of video frame synchronization, blanking concepts and color formats. They can receive up to 16-bit video data directly from video sensors (cameras) and also directly control displays on the output.

The PIXC supports color space conversion and alpha blending for video overlays.

The PVP provides a framework for various vision processing elements, which are targeting mainly edge and object detection strategies.

The video interconnect is a local, distributed bus system, which interconnects the PPI ports, the PIXC, and the PVP.

Pipelined Vision Processor (PVP)

The pipelined vision processor (PVP) provides hardware implementation of signal and image processing algorithms that are required for co-processing and pre-processing of monochrome video frames in ADAS applications, robotic systems, and other machine applications.

The PVP works in conjunction with the Blackfin cores. It is optimized for convolution and wavelet based object detection and classification, and tracking and verification algorithms. The PVP has the following processing blocks.

- Four 5x5 16-bit convolution blocks optionally followed by downscaling
- A 16-bit cartesian-to-polar coordinate conversion block

- A pixel edge classifier that supports 1st and 2nd derivative modes
- An arithmetic unit with 32-bit addition, multiply and divide
- A 32-bit threshold block with 16 thresholds, a histogram, and run-length encoding
- Two 32-bit integral blocks that support regular and diagonal integrals
- An up- and down-scaling unit with independent scaling ratios for horizontal and vertical components
- Input and output formatters for compatibility with many data formats, including Bayer input format

The PVP can form a pipe of all the constituent algorithmic modules and is dynamically configurable to form different pipeline structures.

The PVP supports the simultaneous processing of up to four data streams. The memory pipe stream operates on data received by DMA from any L1, L2, or L3 memory. The three camera pipe streams operate on a common input received directly from any of the three PPI inputs. Optionally, the PIXC can convert color data received by the PPI and forward luma values to the PVP's monochrome engine. Each stream has a dedicated DMA output. This preprocessing concept ensures careful use of available power and bandwidth budgets and frees up the processor cores for other tasks.

The PVP provides for direct core MMR access to all control/status registers. Two hardware interrupts interface to the system event controller. For optimal performance, the PVP allows register programming through its control DMA interface, as well as outputting selected status registers through the status DMA interface. This mechanism enables the PVP to automatically process job lists completely independent of the Blackfin cores.

Parallel Peripheral Interface (PPI)

The enhanced parallel peripheral interface (PPI) is a half-duplex, bidirectional port with a dedicated clock pin and three frame sync (FS) pins. It can support direct connections to active TFT LCD, parallel A/D and D/A converters, video encoders and decoders, image sensor modules and other general-purpose peripherals. Each PPI has a DMA channel associated with it. Moreover, in some modes, a PPI may use an additional DMA channel.

The processor provides up to three parallel peripheral interfaces (PPIs), supporting data widths up to 24 bits. The PPI supports direct connection to TFT LCD panels, parallel analog-to-digital and digital-to-analog converters, video encoders and decoders, image sensor modules and other general-purpose peripherals.

The following features are supported in the PPI module:

- Programmable data length: 8 bits, 10 bits, 12 bits, 14 bits, 16 bits, 18 bits, and 24 bits per clock.
- Various framed, non-framed, and general-purpose operating modes. Frame syncs can be generated internally or can be supplied by an external device.

- ITU-656 status word error detection and correction for ITU-656 receive modes and ITU-656 preamble and status word decode.
- Optional packing and unpacking of data to/from 32 bits from/to 8 bits, 16 bits and 24 bits. If packing/unpacking is enabled, endianness can be configured to change the order of packing/unpacking of bytes/words.
- RGB888 can be converted to RGB666 or RGB565 for transmit modes.
- Various de-interleaving/interleaving modes for receiving/transmitting 4:2:2 YCrCb data.
- Configurable LCD data enable (DEN) output available on Frame Sync 3.

Pixel Compositor (PIXC)

The pixel compositor (PIXC) provides data overlay, transparent color, and color space conversion support. This allows for supporting different video outputs including active (TFT) flat-panel digital color/monochrome LCD displays and analog NTSC/PAL. The color space conversion and text/graphic overlay capabilities, along with visual effect controls, such as transparency control, shorten the processing time on an image data stream, reduce power consumption and save system board space by removing the need for external glue logic.

The PIXC is used to combine and format the data streams required by a wide variety of digital LCD panels and NTSC/PAL analog encoders. It provides all the control needed to allow two data streams from two separate data buffers to be combined and converted into appropriate formats for both LCD panels and video output displays. The main image buffer provides the basic background image presented in the data stream. The overlay image buffer allows the user to add foreground text and graphics on top of the main image data stream. This feature is useful for printing additional graphical or textual information on the screen, such as symbols or a menu, while showing the main image in the background.

Overlay is an option and can be enabled or disabled. If it is disabled, the blender/compositor is bypassed and the data stream from the main image buffer goes directly to memory with optional color space conversion.

Transparent color is just a special case of blending, masking off the blend operation on a pixel-by-pixel basis. In other words, the overlay region consists of sub-regions in any particular color convenient to the programmer and then, if the color data for a given overlay pixel matches the specified transparent color, the overlay function is masked for that pixel and its data is taken solely from the main image buffer, which is stored in memory in either YUV 4:2:2 interleaved format or RGB888 format

Regardless of the data format or buffer structure, each color element is 8 bits wide. If overlay is enabled, a graphics/text overlay data buffer is defined in memory. The color space converter can switch positions among any of the three locations; it can be in the image data path, the overlay data path, or after the blender. The exact position of the color space converter depends on the input and output data formats.

Since the end display may be a TV (NTSC/PAL) or an LCD panel, and since the image/overlay input buffers may be in either RGB888 or YUV4:2:2 format, a color space conversion may be needed. The color

space conversion is selected according to the input data stream format of the PIXC. A YUV-to-RGB format conversion is necessary if the end display is an LCD and if either of the PIXC input data streams is in YUV 4:2:2 format. Similarly, an RGB-to-YUV format conversion is necessary if the end display is a TV and if either of the PIXC input data streams is in RGB888 format.

If the final display device is an LCD, the output RGB data stream is always packed in RGB 8-bit serial format when transferring back to memory. Similarly, if the final display device is a TV, the YUV data stream is always packed in YUV 4:2:2 interleaved format when transferring back to memory.

Reset Control Unit (RCU)

Reset is the initial state of the whole processor or one of the cores and is the result of a hardware or software triggered event. In this state, all control registers are set to their default values and functional units are idle. Exiting a full system reset starts with Core-0 only being ready to boot. Exiting a Core-n only reset starts with this Core-n being ready to boot.

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. This is particularly important when only one of the cores is reset (programs must ensure that there is no pending system activity involving the core that is being reset).

From a system perspective reset is defined by both the reset target and the reset source as described below.

Target defined:

- Hardware Reset - All functional units are set to their default states without exception. History is lost.
- System Reset - All functional units except the RCU are set to their default states.
- Core-n only Reset - Affects Core-n only. The system software should guarantee that the core in reset state is not accessed by any bus master.

Source defined:

- Hardware Reset - The`SYS_HWRST`input signal is asserted active (pulled down).
- System Reset - May be triggered by software (writing to the RCU_CTL register) or by another functional unit such as the dynamic power management (DPM) unit (Hibernate) or any of the system event controller (SEC), trigger routing unit (TRU), or emulator. inputs.
- Core-n-only reset - Triggered by software.
- Trigger request (peripheral).

Booting

The processor has several mechanisms for automatically loading internal and external memory after a reset. The boot mode is defined by the SYS_BMODE input pins dedicated for this purpose. There are two categories of boot modes. In master boot modes, the processor actively loads data from parallel or serial memories. In slave boot modes, the processor receives data from external host devices.

The boot modes are shown in the SYS_BMODE Selections and Boot Modes table. These modes are implemented by the SYS_BMODE bits of the reset configuration register and are sampled during power-on resets and software-initiated resets.

Table 1-3: SYS_BMODE Selections and Boot Modes

SYS_BMODE Setting	Boot Mode
000	No boot/Idle
001	Flash
010	RSI0 Master
011	SPI0 Master
100	SPI0 Slave
101	Reserved
110	LP0 Slave
111	UART0 Slave

System Debug Unit

The System Debug Unit (SDU) provides IEEE-1149.1 support through its JTAG interface. In addition to traditional JTAG features, present in legacy Blackfin products, the SDU adds more features for debugging the chip without halting the core processors.

System Watchpoint Unit

The System Watchpoint Unit (SWU) is a single module which connects to a single system bus and provides for transaction monitoring. One SWU is attached to the bus going to each system slave. The SWU provides ports for all system bus address channel signals. Each SWU contains four match groups of registers with associated hardware. These four SWU match groups operate independently, but share common event (interrupt, trigger, etc.) outputs.

2 System Crossbars (SCB)

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. 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:

- Highly 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/secure) support for selective bus interconnect protection
- Programmable bus arbitration model for bandwidth and latency management

SCB Functional Description

The following sections provide a functional description of the SCB:

- *ADSP-BF60x SCB Register List*
- *SCB Definitions*
- *SCB Block Diagram*

ADSP-BF60x SCB Register List

The system cross bar (SCB), which is often referred to as the system interconnect fabric, is a collection of interconnection units connecting system masters to slave memory spaces. Each unit in the fabric consists of a matrix of master interfaces (MIn). Each of these matrices has a number of slave interfaces (SIn). The SCBs (units in the fabric) with multiple SI for each MI have programmable round-robin arbitration to manage access to slots. A set of registers govern SCB operations. For more information on SCB functionality, see the SCB register descriptions.

Table 2-1: ADSP-BF60x SCB Register List

Name	Description
SCB_ARBRn	Arbitration Read Channel Master Interface n Register
SCB_ARBWn	Arbitration Write Channel Master Interface n Register
SCB_SLAVES	Slave Interfaces Number Register
SCB_MASTERS	Master Interfaces Number Register

SCB Definitions

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

MI (Master Interface)

SCB master interface connected to system bus interconnect slave (for example, L2, sMMR, SCB, and others).

SI (Slave Interface)

SCB slave interface connected to system bus interconnect master (for example, Core, DDE, SCB, and others).

SCB Block Diagram

The SCB architectural model is illustrated in the following figure. This figure shows a high-level overview of the SCB and associated connections to system masters and slaves. A variable number of masters may be connected to a variable number of slaves in each SCB. In this example, all SIs are connected to all MIs as indicated by the lines connecting them.

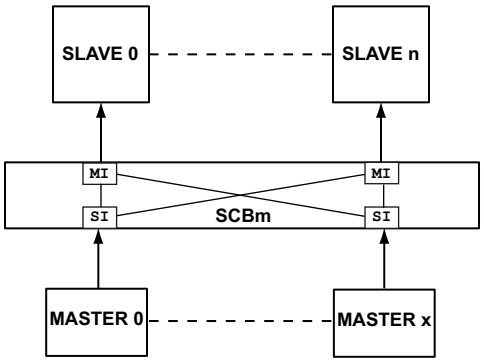


Figure 2-1: SCB Overview

NOTE: For an overall diagram of all SCB interconnections, see the *ADSP-BF60x SCB Block Diagram*.

SCB Hierarchy Block Diagram

A system interconnect built from multiple SCBs in a hierarchical model is illustrated in the following figure. The system master node level SCBs master multiple SIs to a single MI, which in turn connects to an SI of the system slave level node SCB. In this example, all SIs are connected to all MIs.

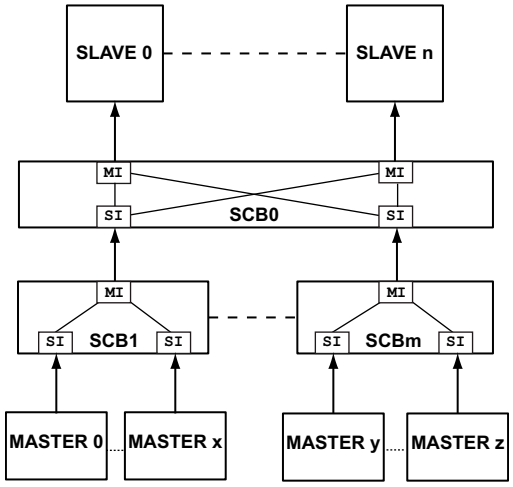


Figure 2-2: SCB Hierarchy Overview

NOTE: For an overall diagram of all SCB interconnections, see the *ADSP-BF60x SCB Block Diagram*.

ADSP-BF60x SCB Block Diagram

The following figure shows the SCB block diagram for the ADSP-BF60x processors.

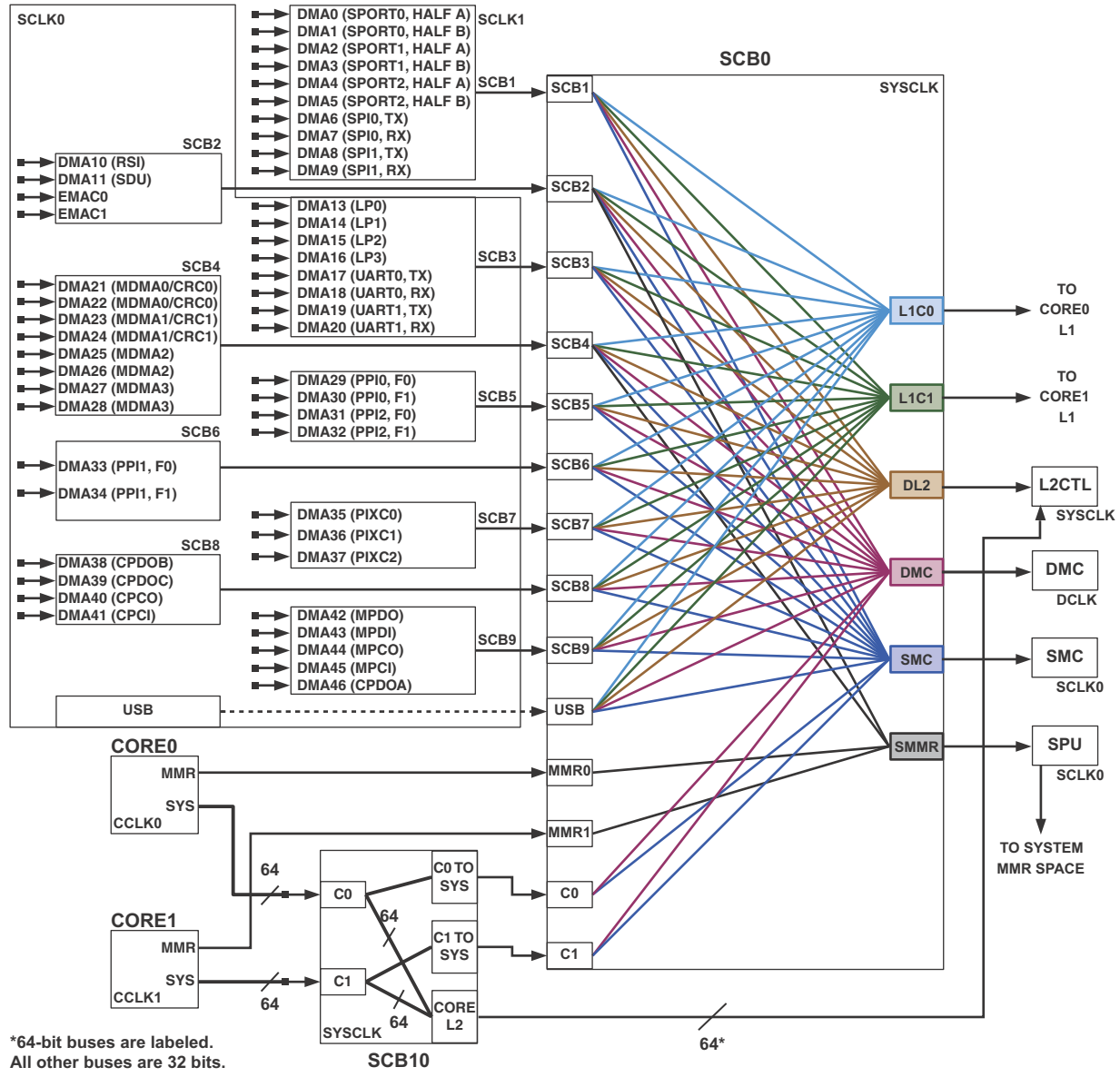


Figure 2-3: ADSP-BF60x SCB Block Diagram

While this figure is useful just for the overview it provides, 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 cores and peripherals on the processor.
- The SCBs connections support DMA channels for some peripherals, support dedicated connections for others (such as USB), and support memory mapped register access for internal memory (L1 and L2) and for external memory (DDR, FLASH, and others).
- The peripherals (and their SCBs) are in the *SCLK0* clock domain; except the SPORT and SPI peripherals (and their SCBs), which are in the *SCLK1* clock domain. SCB0 and SCB10 are in the *SYSCLK*

domain. The processor cores are each in their own clock domain. Synchronization across clock domains affect SCB performance.

- 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.

The following definitions of acronyms (appearing in the figure) may be helpful:

DMA0-DMA46

These indicate DMA channels for peripherals supporting DMA transfers.

SCB0-SCB10

These indicate SCB interfaces, connecting the system bus masters and slaves.

SCLK0, SYSCLK, CCLK0, CCLK1

These indicate clock domains in which the specific SCBs operate. For more information on clock domains, see the Clock Generation Unit chapter and the product data sheet.

L1, L2

These indicate on-core (L1) internal memory and off-core (L2) internal memory.

C0, C1

These indicate processor core 0 (C0) and core 1 (C1).

SMC, L2CTL, DMC

These indicate the static memory controller (SMC), off-core (L2) memory controller, and dynamic memory controller (DMC) interfaces.

PPI0, PPI1, PPI2 - F0/F1

These indicate the parallel peripheral port interfaces, using either internal or external frame sync.

SPORT0, SPORT1, SPORT2 - Half A/B

These indicate the serial port interfaces and their full-duplex halves.

SPI0, SPI1 - RX/TX

These indicate the serial peripheral interfaces ports with receive or transmit paths.

RSI

This indicates the removable storage interface (RSI) interface.

SDU

This indicates the system debug unit (SDU) interface.

SPU

This indicates the system protection unit (SPU).

EMAC0, EMAC1

These indicate the Ethernet MAC 0 and 1 interfaces.

MDMA0, MDMA1, MDMA2, MDMA3

These indicate memory DMA 0 through 3 interfaces.

CRC0, CRC1

These indicate the cyclic redundancy check (CRC) 0 and 1 interfaces.

PIXC0, PIXC1

These indicate the pixel compositor (PIXC) 0 and 1 interfaces

CPCI, CPCO, CPDOC, CPDOB, CPDOA

These indicate the PVP Camera Pipe Control I/O and Camera Pipe Data output A/B/C interfaces.

MPCI, MPCO, MPDI, MPDO

These indicate the PVP Memory Pipe Control I/O and Memory Pipe Data I/O interfaces.

USB

This indicates the universal serial bus (USB) interface.

SMMR

This indicates the system memory-mapped register interface.

ADSP-BF60x SCB Bus Master IDs

The SCB bus master ID indicates which SCB is the current master of a particular bus. While this information is useful for some advanced debugging of SCB arbitration programming, it is deemed to provide too much insight into proprietary design methodology to make publicly available.

Note the following about bus master IDs:

- MMR0 and MMR1 IDs are only seen on the SMMR bus.
- The CL2 bus only has IDs from C0 and C1.
- Non-CL2 C0 and C1 IDs are only seen on the DDR and SMC buses. SCB1, SCB3, SCB5, SCB6, SCB7, SCB8, SCB9 and SCB2 (except for SDU) IDs are seen on DDR, SMC, DL2, L1C0, L1C1.
- SCB4 and SCB2 (SDU and SDU DMA) IDs are seen on DDR, SMC, DL2, L1C0, L1C1, and SMMR.

NOTE: For an overall diagram of all SCB interconnections, see the *ADSP-BF60x SCB Block Diagram*.

Table 2-2: ADSP-BF60x Bus IDs at the slaves (DDR, SMC, CL2, DL2, L1C0, L1C1, SMMR)

Ports	Masters	Hex ID Values Visible on CL2 Bus
SCB1	sp0 hs-A	0x009,0x109
	sp0 hs-B	0x019,0x119
	sp1 hs-A	0x029,0x129
	sp1 hs-B	0x039,0x139
	sp2 hs-A	0x049,0x149
	sp2 hs-B	0x059,0x159
	spi0 tx	0x069,0x169
	spi0 rx	0x079,0x179
	spi1 tx	0x089,0x189
	spi1 rx	0x099,0x199
SCB2	RSI	0x00a,0x08a
	SDU DMA	0x01a,0x09a
	SDU	0x02a
	EMAC0	0x03a,0x0ba
	EMAC1	0x04a,0x0ca
SCB3	LP0	0x00b,0x08b
	LP1	0x01b,0x09b
	LP2	0x02b,0x0ab
	LP3	0x03b,0x0bb
	uart0 tx	0x04b,0x0cb
	uart0 rx	0x05b,0x0db
	uart1 tx	0x06b,0x0eb
	uart1 rx	0x07b,0x0fb
SCB4	DMA21	0x00c,0x08c
	DMA22	0x01c,0x09c
	DMA23	0x02c,0x0ac
	DMA24	0x03c,0x0bc
	DMA25	0x04c,0x0cc
	DMA26	0x05c,0x0dc
	DMA27	0x06c,0x0ec
	DMA28	0x07c,0x0fc

Table 2-2: ADSP-BF60x Bus IDs at the slaves (DDR, SMC, CL2, DL2, L1C0, L1C1, SMMR) (Continued)

Ports	Masters	Hex ID Values Visible on CL2 Bus
SCB5	ppi0	0x005,0x045
	ppi0	0x015,0x055
	ppi2	0x025,0x065
	ppi2	0x035,0x075
SCB6	ppi1	0x006,0x026
	ppi1	0x016,0x036
SCB7	PIXC0	0x008,0x048
	PIXC1	0x018,0x058
	PIXC2	0x028,0x068
SCB8	cpdoB	0x007,0x047
	cpdoC	0x017,0x057
	cpco	0x027,0x067
	cpci	0x037,0x077
SCB9	mpdo/uddo	0x004,0x084
	mpdi/uddi	0x014,0x094
	mpco/upco	0x024,0x0a4
	mpci/upci	0x034,0x0b4
	cpdoA	0x044,0x0c4
USB		0x00d
C0		0x000,0x020,0x040,0x060,0x080,0x0a0,0x0c0,0x0e0
MMR0		0x001
C1		0x012,0x032,0x052,0x072,0x092,0x0b2,0x0d2,0x0f2
MMR1		0x003

Table 2-3: ADSP-BF60x Bus IDs at the slaves (DDR, SMC, CL2, DL2, L1C0, L1C1, SMMR)

Ports	Masters	Hex ID Values Visible on CL2 Bus
C0		0x000,0x002,0x004,0x006, 0x008,0x00a,0x00c,0x00e

Table 2-3: ADSP-BF60x Bus IDs at the slaves (DDR, SMC, CL2, DL2, L1C0, L1C1, SMMR) (Continued)

Ports	Masters	Hex ID Values Visible on CL2 Bus
C1		0x001,0x003,0x005,0x007, 0x009,0x00b,0x00d,0x00f

ADSP-BF60x SCB Arbitration Tables

The SCB uses round robin arbitration to prioritize each slave's interface to masters (Master Interface) and each master's interface to slaves (Slave Interface). This section provides reference information about these interfaces.

NOTE: For an overall diagram of all SCB interconnections, see the ADSP-BF60x SCB Block Diagram.

For SCB programing information, see the SCB Programming Model.

SCB Units, Master Interfaces, and Arbitration Types

Table 2-4: SCB Units, Master Interfaces, and Arbitration Types

SCB	Master Interface No.	Arbitration Type	Programmable Slots
SCB0	MI0 (DDR)	Prog Round Robin	32
	MI1 (SMC)	Prog Round Robin	32
	MI2 (DL2)	Prog Round Robin	32
	MI3 (L1C0)	Prog Round Robin	32
	MI4 (L1C1)	Prog Round Robin	32
	MI5 (SMMR)	Prog Round Robin	16
SCB1	MI0 (SCB0)	Prog Round Robin	20
SCB2	MI0 (SCB0)	Prog Round Robin	10
SCB3	MI0 (SCB0)	Prog Round Robin	16
SCB4	MI0 (SCB0)	Prog Round Robin	16
SCB5	MI0 (SCB0)	Prog Round Robin	8
SCB6	MI0 (SCB0)	Prog Round Robin	4
SCB7	MI0 (SCB0)	Prog Round Robin	6
SCB8	MI0 (SCB0)	Prog Round Robin	8
SCB9	MI0 (SCB0)	Prog Round Robin	10
SCB10	MI0 (CL2)	Prog Round Robin	16
	MI1 (C02Sys)	Fixed Round Robin	0
	MI2 (C12Sys)	Fixed Round Robin	0

SCB0 Slots and Masters

Table 2-5: SCB0 Arbitration Default Settings for MI0 (DDR)

Slot	Master
0	SI0 (C0)
1	SI2 (C1)
2	SI4 (SCB9)
3	SI5 (SCB5)
4	SI6 (SCB6)
5	SI7 (SCB8)
6	SI8 (SCB7)
7	SI9 (SCB1)
8	SI10 (SCB2)
9	SI11 (SCB3)
10	SI13 (USB)
11	SI12 (SCB4)
12	SI0 (C0)
13	SI2 (C1)
14	SI4 (SCB9)
15	SI5 (SCB5)
16	SI6 (SCB6)
17	SI7 (SCB8)
18	SI8 (SCB7)
19	SI9 (SCB1)
20	SI10 (SCB2)
21	SI11 (SCB3)
22	SI13 (USB)
23	SI12 (SCB4)
24	SI0 (C0)
25	SI2 (C1)
26	SI4 (SCB9)
27	SI5 (SCB5)
28	SI6 (SCB6)

Table 2-5: SCB0 Arbitration Default Settings for MI0 (DDR) (Continued)

Slot	Master
29	SI7 (SCB8)
30	SI8 (SCB7)
31	SI13 (USB)

Table 2-6: SCB0 Arbitration Default Settings for MI1 (SMC)

Slot	Master
0	SI0 (C0)
1	SI2 (C1)
2	SI4 (SCB9)
3	SI5 (SCB5)
4	SI6 (SCB6)
5	SI7 (SCB8)
6	SI8 (SCB7)
7	SI9 (SCB1)
8	SI10 (SCB2)
9	SI11 (SCB3)
10	SI13 (USB)
11	SI12 (SCB4)
12	SI0 (C0)
13	SI2 (C1)
14	SI4 (SCB9)
15	SI5 (SCB5)
16	SI6 (SCB6)
17	SI7 (SCB8)
18	SI8 (SCB7)
19	SI9 (SCB1)
20	SI10 (SCB2)
21	SI11 (SCB3)
22	SI13 (USB)
23	SI12 (SCB4)
24	SI0 (C0)

Table 2-6: SCB0 Arbitration Default Settings for MI1 (SMC) (Continued)

Slot	Master
25	SI2 (C1)
26	SI4 (SCB9)
27	SI5 (SCB5)
28	SI6 (SCB6)
29	SI7 (SCB8)
30	SI8 (SCB7)
31	SI13 (USB)

Table 2-7: SCB0 Arbitration Default Settings for MI2 (DL2)

Slot	Master
0	SI4 (SCB9)
1	SI5 (SCB5)
2	SI6 (SCB6)
3	SI7 (SCB8)
4	SI8 (SCB7)
5	SI9 (SCB1)
6	SI10 (SCB2
7	SI11 (SCB3)
8	SI13 (USB)
9	SI12 (SCB4)
10	SI4 (SCB9)
11	SI5 (SCB5)
12	SI6 (SCB6)
13	SI7 (SCB8)
14	SI8 (SCB7)
15	SI9 (SCB1)
16	SI10 (SCB2)
17	SI11 (SCB3)
18	SI13 (USB)
19	SI12 (SCB4)
20	SI4 (SCB9)

Table 2-7: SCB0 Arbitration Default Settings for MI2 (DL2) (Continued)

Slot	Master
21	SI5 (SCB5)
22	SI6 (SCB6)
23	SI7 (SCB8)
24	SI8 (SCB7)
25	SI9 (SCB1)
26	SI10 (SCB2)
27	SI11 (SCB3)
28	SI13 (USB)
29	SI12 (SCB4)
30	SI4 (SCB9)
31	SI7 (SCB8)

Table 2-8: MI3 (L1A)

Slot	Master
0	SI4 (SCB9)
1	SI5 (SCB5)
2	SI6 (SCB6)
3	SI7 (SCB8)
4	SI8 (SCB7)
5	SI9 (SCB1)
6	SI10 (SCB2)
7	SI11 (SCB3)
8	SI13 (USB)
9	SI12 (SCB4)
10	SI4 (SCB9)
11	SI5 (SCB5)
12	SI6 (SCB6)
13	SI7 (SCB8)
14	SI8 (SCB7)
15	SI9 (SCB1)
16	SI10 (SCB2)

Table 2-8: MI3 (L1A) (Continued)

Slot	Master
17	SI11 (SCB3)
18	SI13 (USB)
19	SI12 (SCB4)
20	SI4 (SCB9)
21	SI5 (SCB5)
22	SI6 (SCB6)
23	SI7 (SCB8)
24	SI8 (SCB7)
25	SI9 (SCB1)
26	SI10 (SCB2)
27	SI11 (SCB3)
28	SI13 (USB)
29	SI12 (SCB4)
30	SI4 (SCB9)
31	SI7 (SCB8)

Table 2-9: MI4 (L1B)

Slot	Master
0	SI4 (SCB9)
1	SI5 (SCB5)
2	SI6 (SCB6)
3	SI7 (SCB8)
4	SI8 (SCB7)
5	SI9 (SCB1)
6	SI10 (SCB2)
7	SI11 (SCB3)
8	SI13 (USB)
9	SI12 (SCB4)
10	SI4 (SCB9)
11	SI5 (SCB5)
12	SI6 (SCB6)

Table 2-9: MI4 (L1B) (Continued)

Slot	Master
13	SI7 (SCB8)
14	SI8 (SCB7)
15	SI9 (SCB1)
16	SI10 (SCB2)
17	SI11 (SCB3)
18	SI13 (USB)
19	SI12 (SCB4)
20	SI4 (SCB9)
21	SI5 (SCB5)
22	SI6 (SCB6)
23	SI7 (SCB8)
24	SI8 (SCB7)
25	SI9 (SCB1)
26	SI10 (SCB2)
27	SI11 (SCB3)
28	SI13 (USB)
29	SI12 (SCB4)
30	SI4 (SCB9)
31	SI7 (SCB8)

Table 2-10: MI5 (SMMR)

Slot	Master
0	SI1 (MMR0)
1	SI3 (MMR1)
2	SI10 (SCB2)
3	SI12 (SCB4)
4	SI1 (MMR0)
5	SI3 (MMR1)
6	SI10 (SCB2)
7	SI12 (SCB4)
8	SI1 (MMR0)

Table 2-10: MI5 (SMMR) (Continued)

Slot	Master
9	SI3 (MMR1)
10	SI10 (SCB2)
11	SI12 (SCB4)
12	SI1 (MMR0)
13	SI3 (MMR1)
14	SI10 (SCB2)
15	SI12 (SCB4)

SCB1 Slots and Masters

Table 2-11: MI0 (SCB1)

Slot	Master
0	SI0 (SPORT0A)
1	SI1 (SPORT0B)
2	SI2 (SPORT1A)
3	SI3 (SPORT1B)
4	SI4 (SPORT2A)
5	SI5 (SPORT2B)
6	SI6 (SPI0TX)
7	SI7 (SPI0RX)
8	SI8 (SPI1TX)
9	SI9 (SPI1RX)
10	SI0 (SPORT0A)
11	SI1 (SPORT0B)
12	SI2 (SPORT1A)
13	SI3 (SPORT1B)
14	SI4 (SPORT2A)
15	SI5 (SPORT2B)
16	SI6 (SPI0TX)
17	SI7 (SPI0RX)
18	SI8 (SPI1TX)
19	SI9 (SPI1RX)

SCB2 Slots and Masters

Table 2-12: MI0 (SCB2)

Slot	Master
0	SI0 (RSI)
1	SI1 (SDU DMA)
2	SI2 (SDU)
3	SI3 (EMAC0)
4	SI4 (EMAC1)
5	SI0 (RSI)
6	SI1 (SDU DMA)
7	SI2 (SDU)
8	SI3 (EMAC0)
9	SI4 (EMAC1)

SCB3 Slots and Masters

Table 2-13: MI0 (SCB3)

Slot	Master
0	SI0 (LP0)
1	SI1 (LP1)
2	SI2 (LP2)
3	SI3 (LP3)
4	SI4 (UART0TX)
5	SI5 (UART0RX)
6	SI6 (UART1TX)
7	SI7 (UART1RX)
8	SI0 (LP0)
9	SI1 (LP1)
10	SI2 (LP2)
11	SI3 (LP3)
12	SI4 (UART0TX)
13	SI5 (UART0RX)
14	SI6 (UART1TX)

Table 2-13: MI0 (SCB3) (Continued)

Slot	Master
15	SI7 (UART1RX)

SCB4 Slots and Masters

Table 2-14: MI0 (SCB4)

Slot	Master
0	SI0 (DMA21)
1	SI1 (DMA22)
2	SI2 (DMA23)
3	SI3 (DMA24)
4	SI4 (DMA25)
5	SI5 (DMA26)
6	SI6 (DMA27)
7	SI7 (DMA28)
8	SI0 (DMA21)
9	SI1 (DMA22)
10	SI2 (DMA23)
11	SI3 (DMA24)
12	SI4 (DMA25)
13	SI5 (DMA26)
14	SI6 (DMA27)
15	SI7 (DMA28)

SCB5 Slots and Masters

Table 2-15: MI0 (SCB5)

Slot	Master
0	SI0 (PPI0 -- DMA29)
1	SI1 (PPI0 -- DMA30)
2	SI2 (PPI2 -- DMA31)
3	SI3 (PPI2 -- DMA32)
4	SI0 (PPI0 -- DMA29)

Table 2-15: MI0 (SCB5) (Continued)

Slot	Master
5	SI1 (PPI0 -- DMA30)
6	SI2 (PPI2 -- DMA31)
7	SI3 (PPI2 -- DMA32)

SCB6 Slots and Masters

Table 2-16: MI0 (SCB6)

Slot	Master
0	SI0 (PPI1 -- DMA33)
1	SI1 (PPI1 -- DMA34)
2	SI0 (PPI1 -- DMA33)
3	SI1 (PPI1 -- DMA34)

SCB7 Slots and Masters

Table 2-17: MI0 (SCB7)

Slot	Master
0	SI0 (PIXC0)
1	SI1 (PIXC1)
2	SI2 (PIXC2)
3	SI0 (PIXC0)
4	SI1 (PIXC1)
5	SI2 (PIXC2)

SCB8 Slots and Masters

Table 2-18: MI0 (SCB8)

Slot	Master
0	SI0 (PVP CPDOB)
1	SI1 (PVP CPDOC)
2	SI2 (PVP CPCO)
3	SI3 (PVP CPCI)
4	SI0 (PVP CPDOB)

Table 2-18: MI0 (SCB8) (Continued)

Slot	Master
5	SI1 (PVP CPDOC)
6	SI2 (PVP CPCO)
7	SI3 (PVP CPCI)

SCB9 Slots and Masters

Table 2-19: MI0 (SCB9)

Slot	Master
0	SI0 (PVP MPDO/UDDO)
1	SI1 (PVP MPDI/UDDI)
2	SI2 (PVP MPCO/UPCO)
3	SI3 (PVP MPCI/UPCI)
4	SI4 (PVP CPDOA)
5	SI0 (PVP MPDO/UDDO)
6	SI1 (PVP MPDI/UDDI)
7	SI2 (PVP MPCO/UPCO)
8	SI3 (PVP MPCI/UPCI)
9	SI4 (PVP CPDOA)

SCB Programming Model

The SCB arbitration model is programmable. Before modifying the default read or write arbitration settings, it is important to know and follow the following SCB register access restrictions:

1. Do not attempt to access reserved or unused address locations. Attempting to access these locations can result in unpredictable behavior of the SCB.
2. Unless otherwise stated in the accompanying text:
    - Do not modify undefined register bits.
    - Ignore undefined register bits on reads.
    - All register bits are reset to a logic 0 by a system reset.

NOTE: No protection is imposed when you program the slots in the arbitration scheme. It is possible to remove an SI from all the slots (making the SI inaccessible).

For information about the SCBs in the processor and their slot numbers (for modifying read/write arbitration settings), see the *ADSP-BF60x SCB Arbitration Tables*.

Reading Arbitration Settings

To read arbitration parameter data (for example, get the identity of the SI assigned to an arbitration slot), use a write-followed-by-read process with bits 31-24 set to FF:

1. Write `SCB_ARBWn` with the value 0xFF0000mm (where mm is the slot index for the following read).
2. Read the value in `SCB_ARBWn`, the value is 0x000000nn (where nn is identity of the SI assigned to slot mm).

Writing Arbitration Settings

To write arbitration parameter data (for example, assign an SI to an SCB slot), use a write with bits 31-24 set to the slot number.

SCB Programming Concepts

The SCB arbitration model is programmable. Through register configuration, the arbitration model can be adjusted for the specific bandwidth and latency requirements. The arbitration model is programmable round robin with the following features:

- Each MI has its own set of arbitration slots.
- The total number of arbitration slots is specific to the SCB and fixed in hardware.
- Arbitration slot priority rotates each cycle.
- SIs are assigned to each slot. (See *ADSP-BF60x SCB Arbitration Tables*.)
- Transactions are granted to highest priority requesting master each cycle.

ADSP-BF60x SCB Register Descriptions

System Cross Bar (SCB) contains the following registers.

Table 2-20: ADSP-BF60x SCB Register List

Name	Description
`SCB_ARBRn`	Arbitration Read Channel Master Interface n Register

Table 2-20: ADSP-BF60x SCB Register List (Continued)

Name	Description
SCB_ARBWn	Arbitration Write Channel Master Interface n Register
SCB_SLAVES	Slave Interfaces Number Register
SCB_MASTERS	Master Interfaces Number Register

Arbitration Read Channel Master Interface n Register

Each master interface (MI) of an SCB has an SCB_ARBRn register, providing indexed access to the read arbitration parameters of the slave interfaces (SIn) connected to that MI.

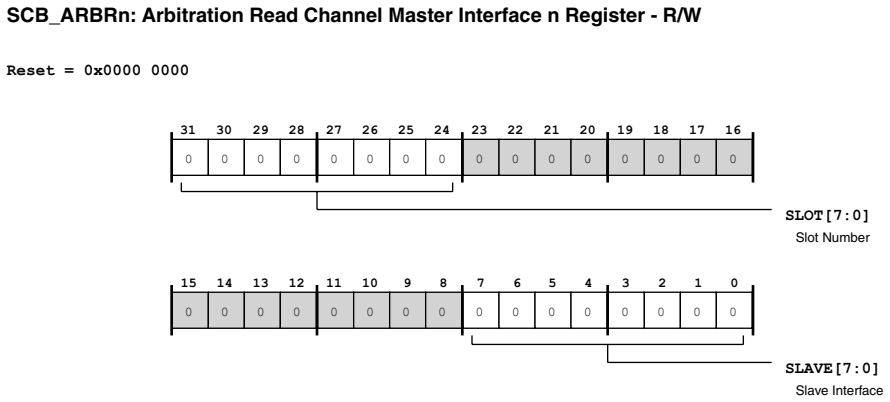


Figure 2-4: SCB_ARBRn Register Diagram

Table 2-21: SCB_ARBRn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R0/W)	SLOT	Slot Number. The SCB_ARBRn.SLOT bits either hold the SCB slot number (for writing arbitration data) or hold the value FF (for write-followed-by-read of arbitration data). For a list of slot numbers for specific SCBs, see the SCB function description.
7:0 (R/W)	SLAVE	Slave Interface. The SCB_ARBRn.SLAVE bits either hold the SCB slave interface (SI) to be assigned to the slot number in SCB_ARBRn.SLOT (for writing arbitration data) or hold the slot number (for write-followed-by-read of arbitration data). For a list of slot numbers of specific SCBs, see the SCB functional description.

Arbitration Write Channel Master Interface n Register

Each master interface (MI) SCB has an SCB_ARBWn register, providing indexed access to the write arbitration parameters of the slave interfaces (SIn) connected to that MI.

SCB_ARBWn: Arbitration Write Channel Master Interface n Register - R/W

Reset = 0x0000 0000

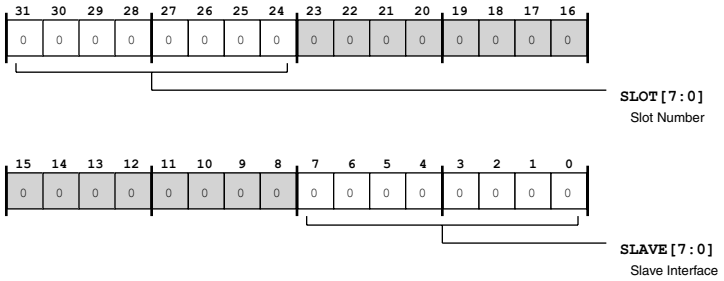


Figure 2-5: SCB_ARBWn Register Diagram

Table 2-22: SCB_ARBWn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R0/W)	SLOT	Slot Number. The SCB_ARBWn.SLOT bits either hold the SCB slot number (for writing arbitration data) or hold the value FF (for write-followed-by-read of arbitration data). For a list of slot numbers for specific SCBs, see the SCB function description.
7:0 (R/W)	SLAVE	Slave Interface. The SCB_ARBWn.SLAVE bits either hold the SCB slave interface (SI) to be assigned to the slot number in SCB_ARBWn.SLOT (for writing arbitration data) or hold the slot number (for write-followed-by-read of arbitration data). For a list of slot numbers of specific SCBs, see the SCB functional description.

Slave Interfaces Number Register

The SCB_SLAVES register holds the number of slave interfaces (SI) connected to the SCB.

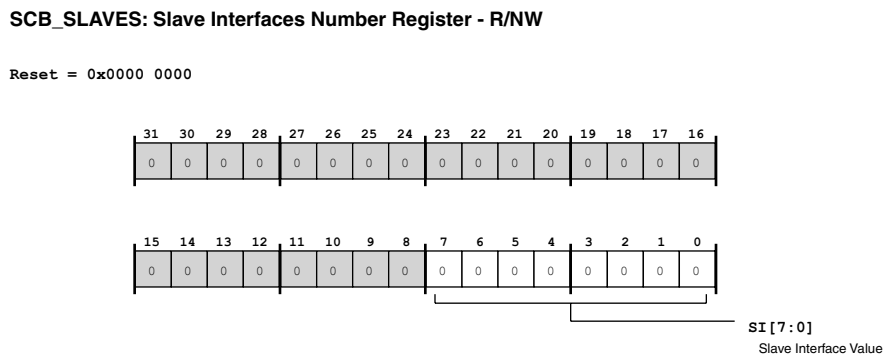


Figure 2-6: SCB_SLAVES Register Diagram

Table 2-23: SCB_SLAVES Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	SI	Slave Interface Value. The SCB_SLAVES.SI bits hold the number of slave interfaces connected to the SCB.

Master Interfaces Number Register

The SCB_MASTERS register holds the number of master interfaces (SI) connected to the SCB.

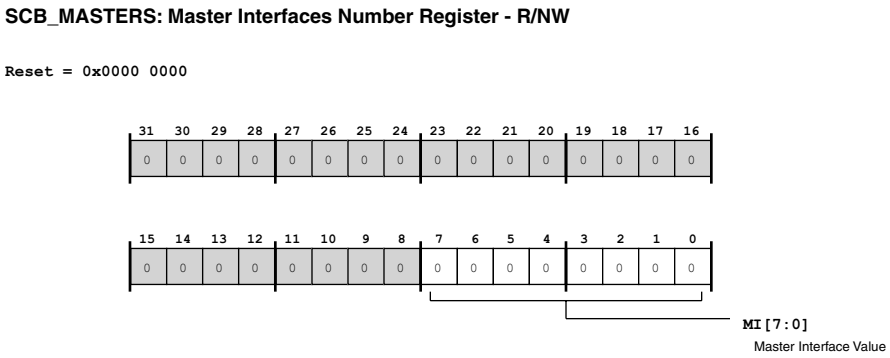


Figure 2-7: SCB_MASTERS Register Diagram

Table 2-24: SCB_MASTERS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	MI	Master Interface Value. The SCB_MASTERS.MI bits hold the number of master interfaces connected to the SCB.

3 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 CLKIN 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.

CGU Features

The following features are supported in the CGU module:

- Generates all on-chip clocks and synchronization signals; programmable values divide the PLL clock frequency to generate the core clock (*CCLK*), the system clocks (*SYSCLK*, *SCLK0* and *SCLK1*), the LPDDR or DDR2 clock (*DCLK*) and the output clock (*OCLK*)
- Provides smooth transitions from current clock condition to new condition with PLL logic, executes the changes to clocks due to register programming
- Supports programmable options for the `SYS_CLKOUT` output, which may output divided-down versions of the on-chip clocks; by default, the `SYS_CLKOUT` pin drives a buffered version of the `SYS_CLKIN` input
- Provides PLL and clock domain status reporting for event management
- Maximizes power management flexibility in conjunction with the DPM
- Manages power dynamically, allowing the processor's core clock frequency (f_{CCLK}) to be dynamically controlled
- Provides clock generation support for multiple operating/sleep modes to permit a custom power usage model; modes include full-on mode, active mode, deep sleep mode, and hibernate mode

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

For more information about CGU/DPM integrated features, see the Dynamic Power Management (DPM) chapter.

CGU Functional Description

The CGU (clock generation unit) generates all on-chip clocks and synchronization signals based on the programmed PLL multiplication factor and dividers. The following sections describe the CGU features:

- *ADSP-BF60x CGU Register List*
- *ADSP-BF60x CGU Interrupt List*
- *ADSP-BF60x CGU Trigger List*
- *CGU Definitions*
- *CGU PLL Block Diagram*

ADSP-BF60x 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 3-1: ADSP-BF60x CGU Register List

Name	Description
CGU_CTL	Control Register
CGU_STAT	Status Register
CGU_DIV	Clocks Divisor Register
CGU_CLKOUTSEL	CLKOUT Select Register

ADSP-BF60x CGU Interrupt List

Table 3-2: ADSP-BF60x CGU Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
CGU0 Event	1		PULSE/EDGE
CGU0 Error	129		LEVEL

ADSP-BF60x CGU Trigger List

Table 3-3: ADSP-BF60x CGU Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
CGU0 Event	1	PULSE/EDGE

Table 3-4: ADSP-BF60x CGU Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
None		

CGU Definitions

DPM

The dynamic power management (DPM) works with the CGU to provide flexible power disssapation models for the processor.

PCU

The PLL control unit (PCU) in the CGU controls PLL operations.

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.

CGU

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

Table 3-5: Clock Descriptions

Clock	Description
PLLCLK	Phase-locked loop clock provides the source from which all clocks below are derived from unless the PLL is bypassed
CCLK0	Core Clock 0
CCLK1	Core Clock 1

Table 3-5: Clock Descriptions (Continued)

Clock	Description
SYSCLK	Clock for system buses and provides the source from which SCLK0 and SCLK1 are derived
SCLK0	PVP and all other peripherals not clocked by SCLK1
SCLK1	SPORT, SPI, and ACM peripherals clock
DCLK	Dynamic memory clock
OCLK	Output clock is a possible source for SYS_CLKOUT

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) which multiplies the SYS_CLKINinput to a higher frequency.

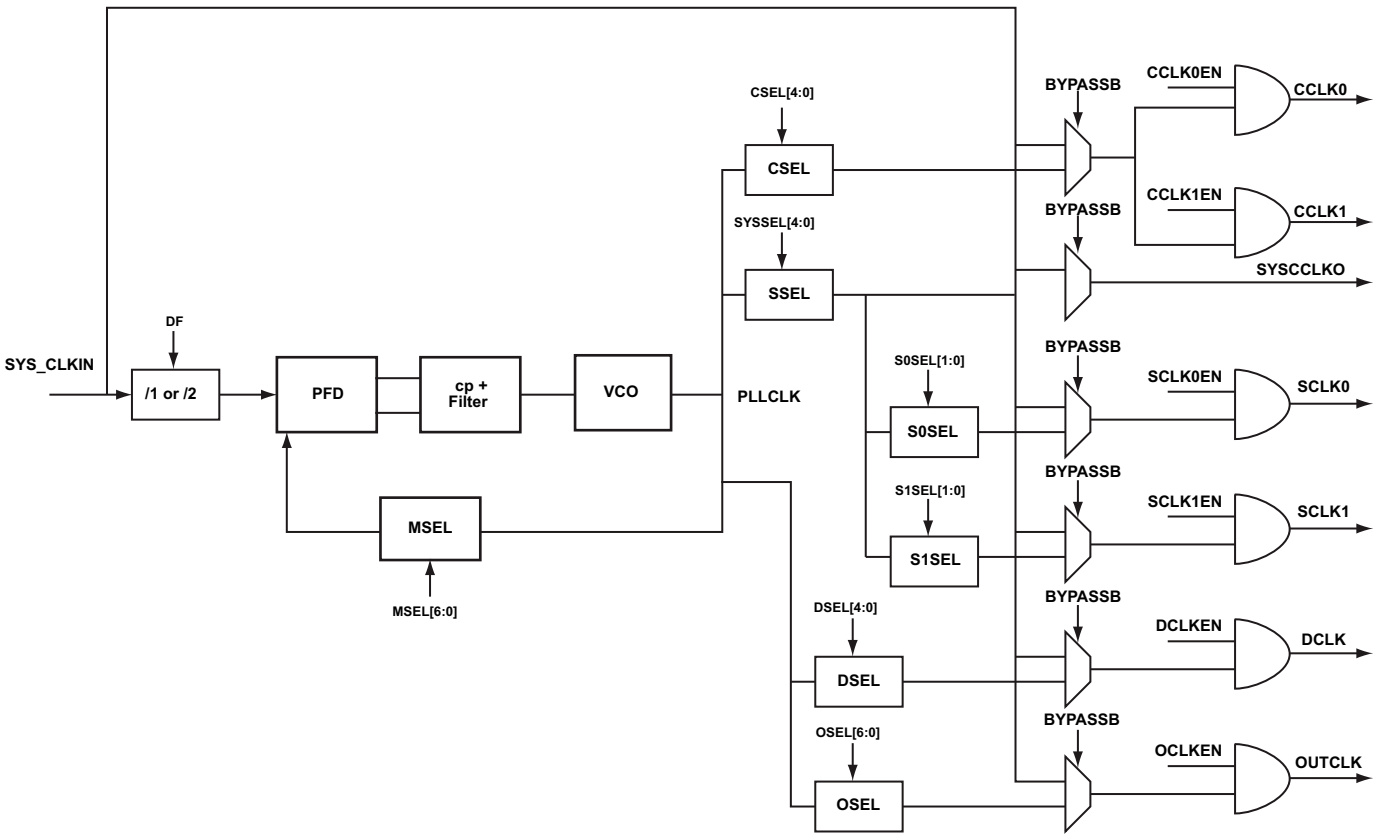


Figure 3-1: CGU PLL Block Diagram

The output of these blocks is called *PLLCLK*. The *PLLCLK*is divided to form *CCLK0*, *CCLK1*, *SYSCLK*, *DCLK*, and *OCLK*. The *SYSCLK* is further divided to form *SCLK0* and *SCLK1*.

The *OCLK* (shown in the CGU PLL block diagram) is routed to the *CLKOUT* block (shown in the SYS_CLKOUTgeneration figure), so *OCLK* can be selected as one of the SYS_CLKOUTsources.

The SYS_CLKOUTgeneration figure is a conceptual representation of the CLKOUT module. As shown in the CGU PLL block diagram, many clocks are available on the SYS_CLKOUToutput pin. The selection of which clock outputs on the SYS_CLKOUTpin is controlled by CGU_CLKOUTSEL.CLKOUTSEL.

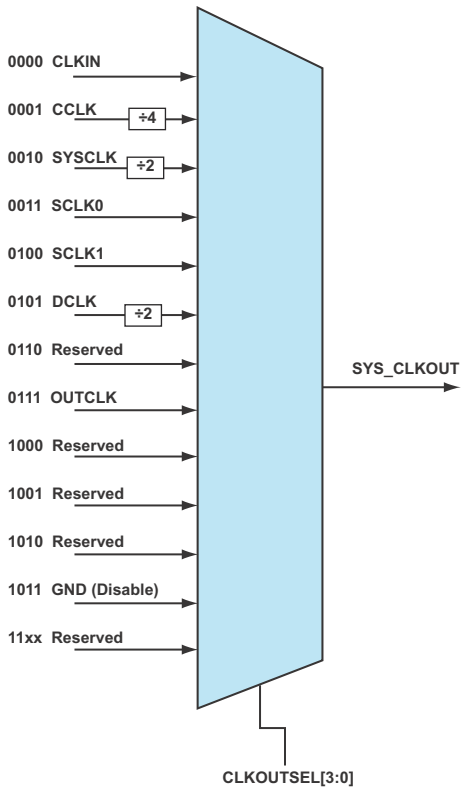


Figure 3-2: SYS_CLKOUT Generation

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 CGU's PLL 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 the System Crossbars (SCB) chapter.
- The DPM uses the CGU for clock management as power state transitions occur. For more information, see the Dynamic Power Management (DPM) chapter.

CGU Event Control

The CGU is capable of generating a CGU Event or CGU Error for several different causes.

CGU Event

The CGU event interrupt is used for different purposes. After a frequency change, a CGU event indicates that the PLL has locked and clocks are synchronized. The CGU event interrupt can be used to break a core idle if a core was idled while changing frequencies. While in active mode, a CGU event indicates that the PLL has locked.

CGU Error

If the PLL fails to lock, the PLL is disabled, and a CGU error is triggered. In addition, CGU_STAT.PLOCKERR is set. The PLL resets itself when it is disabled. If the lock error happens during reset, system reset will be exited and a CGU event is triggered. If the failed lock occurs during a frequency change, the cores exit idle. In order to clear the CGU event in an interrupt service routine, write to CGU_CTL.MSEL or CGU_CTL.DF. This write makes the PLL exit the error state (the CGU_STAT.PLOCKERR bit is cleared and the CGU_EVENT signal is de-asserted) and re-lock. This PLL operation occurs even if the new values for the CGU_CTL.MSEL field and the CGU_CTL.DF field are the same as the previous values. The CGU_CTL.LOCK bit and the CGU_CTL.WFI bit settings still apply. If the PLL lock error occurred during a hardware or software triggered system reset, after out of reset, the system is functional and can boot. If the PLL lock error occurs during a PLL frequency change, the CGU event interrupt makes the cores exit idle, as in the non error case.

The CGU_STAT.WDIVERR bit indicates a write access to the CGU_DIV register (to trigger an alignment sequence or to change CGU_DIV.CSEL, CGU_DIV.SYSSEL, CGU_DIV.SOSEL, CGU_DIV.S1SEL, or CGU_DIV.DSEL) while the PLL is locked, but still aligning the clocks. This condition generates a CGU error. If this error occurs, it should be cleared and the desired values should be written to the CGU_DIV register again.

The CGU_STAT.WDFMSERR bit indicates a write access to the CGU_CTL register to change the CGU_CTL.DF field or the CGU_CTL.MSEL field while the PLL is locking. This condition generates a CGU error. If this error occurs, wait until the PLL has finished locking, clear the error, and write again the desired value change to the CGU_CTL.DF field or the CGU_CTL.MSEL field.

The CGU_STAT.DIVERR indicates a clock divisor value error, occurring when a *CCLK* divisor is greater than the *SYSCLK* divisor, as in: CGU_DIV.CSEL> CGU_DIV.SYSSEL. The CGU issues a CGU error for this condition. If this error occurs, it should be cleared and the new values should be written to the CGU_DIV register, such that the CGU_DIV.CSEL field value is less than or equal to the CGU_DIV.SYSSEL field value.

CGU Generated Bus Errors

The CGU generates a bus error if a read or write transaction is attempted to an unused address within the CGU address range or if 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, a bus error also is generated. In addition, the CGU_STAT.LWERR bit is set.

CGU Programming Model

This section describes the programming concepts and mode configuration techniques for the CGU.

CGU Mode Configuration

This section provides procedures related to clock and PLL configuration.

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 re locks, 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), or the CGU_STAT.PLOCKERR bit =1 (PLL lock error, the PLL failed to lock).
   c. The CGU_STAT.CLKSALGN bit =0 (clocks aligned).
2. Write the desired values to the CGU_DIV register's clock divisor select (SEL) fields 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 cores are idle, write to the CGU_CTL register with the CGU_CTL.WFI bit =1.
   b. To change the PLL frequency while the cores are active, write to the CGU_CTL register with the CGU_CTL.WFI bit =0.

AFTER COMPLETING THIS TASK:

This sequence updates the corresponding CGU registers; bypasses the PLL; makes the PLL lock to the new values in the CGU_CTL.MSEL or CGU_CTL.DF fields; changes the clock frequencies; and exits 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. The CGU_STAT.PLOCK bit, CGU_STAT.PLLBP bit, and CGU_STAT.CLKSALGN bit may be polled to discover when the PLL is locked and the clocks are aligned.

Changing the PLL's frequency is allowed while the PLL is bypassed but the new PLLCLK frequency is not used until the PLL is no longer bypassed.

CAUTION: Changing the PLL frequency causes the *DCLK* frequency to change. Either accessing dynamic memory (for example, DDR) or accessing the dynamic memory controller (DMC) registers while PLL and/or clock frequency changes are in progress may have unpredictable results.

Changing the CCLKn, SYSCLK, or SCLKn frequency Without Modifying the PLLCLK Frequency

To change the clock frequencies is done by writing new CGU_DIV.CnSEL, CGU_DIV.SYSSEL, or CGU_DIV.SnSELvalues. The frequency change occurs only when the PLL is not bypassed. Any time the CCLKn, SYSCLK, or SCLKn clock frequencies is changed, they all exit the frequency change sequence aligned. The CGU_SYSDCLK_ALGN is not asserted even if the SYSCLK and the DCLK frequencies are equal.

1. Read the CGU_STAT register. Verify that CLKSALGN = 0 (clocks aligned)
2. Write the desired CSEL, SOSEL, SYSSEL, S1SEL, DSELand OSEL values to the CGUO_DIV registers with the UPDT bit = 1.

   ADDITIONAL INFORMATION: This write updates the CGU_DIV register and changes the CCLKn, SYSCLK or SCLKn frequencies and aligns these clocks. When the clocks are aligned a CGU Event occurs.

AFTER COMPLETING THIS TASK:

The CGU_STAT register exits this sequence with the CLKSALGN bit cleared. The CLKSALGN bit can be polled to discover when the clocks are aligned. Any write to the CGU_DIV register intended to change an xSEL field while the CLKSALGN bit = 1 (clocks alignment in progress) triggers a bus error and the CGU_DIV register is not modified.

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

Writing to the CGU_DIV register is allowed while the processor is in active (PLL bypassed) mode but 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 SYSCLK frequency is not supported and may have unpredictable results.

Changing the DCLK Clock Frequency

To change the DCLK clock frequency write a new CGU_DIV.DSEL value. The frequency change occurs only when the PLL is not bypassed. Any time DCLK clock frequency is changed, DCLK, CCLKn, SYSCLK and SCLKn clocks exit the frequency change sequence aligned. The CGU_SYSDCLK_ALGN bit = 1 if the SYSCLK and the DCLK frequencies are equal.

1. Read the CGUO_STAT register. Verify that CLKSALGN = 0 (clocks aligned)

2. Write the desired DSEL value to the CGU0_DIV register, with the UPDT bit = 1.

   ADDITIONAL INFORMATION: This write updates the CGU_DIV register, changes the DCLK frequency, and aligns all clocks except OUTCLK.

AFTER COMPLETING THIS TASK:

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

Writing to CGU0_DIV.DSEL is allowed while the processor is in active (PLL bypassed) mode but the effect of the write is visible only after the transition to full-on (PLL not bypassed) mode.

Accessing the DDR memory, or the DDR memory controller's registers while changing the DCLK frequency is not supported and may have unpredictable results.

Changing the OUTCLK Frequency

To change the OUTCLK clock frequency, write a new CGU_DIV.OSEL value. Any time OUTCLK clock frequency is changed, the OUTCLK, CCLKn, SYSCLK and SCLKn clocks exit the frequency change sequence aligned. The CGU_SYSDCLK_ALGN signal is not modified.

1. Read CGU_STAT register. Verify that CLKSALGN = 0 (clocks aligned)
2. Write desired OSEL value with the UPDT bit 1 to the CGU0_DIV register.

   ADDITIONAL INFORMATION: This write updates the CGU_DIV register, changes the DCLK frequency, and aligns all clocks except OUTCLK.

AFTER COMPLETING THIS TASK:

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

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

Aligning All Clocks

To align CCLKn, SYSCLK, SCLKn, DCLK and OUTCLK clocks write 1 to CGUO_DIV.ALGN. The CSEL, SYSSEL, SnSEL, DSEL or OSEL may be changed if a frequency change is also required. The CGU_SYSDCLK_ALGN is asserted if SYSCLK and DCLK frequencies are equal.

1. Read the CGU_STAT register. Verify that CLKSALGN = 0 (clocks aligned).
2. Write 1 to the ALGN bit in CGU_DIV register. All other fields may or may not change.

   ADDITIONAL INFORMATION: This write does not alter the CGU_DIV register unless any of the xSEL fields is modified. When clocks are aligned a CGU event occurs.

AFTER COMPLETING THIS TASK:

The CGU_STAT register exits this sequence with the CLKSALGN bit = 0. The CLKSALGN bit can be polled to discover when the clocks are aligned. Any write to the CGUO_DIV register intended to align clocks or to change an xSEL field while CLKSALGN = 1 (clocks alignment in progress) triggers an MMR access bus error and the CGUO_DIV register is not modified.

Writing 1 to CGUO_DIV.ALGN has no effect while the processor is in active (PLL bypassed) mode. Accessing the DDR memory while changing the DCLK or SYSCLK frequencies is not supported and may have unpredictable results.

ADSP-BF60x Valid Clock Multiplier Settings

Processor operations depend on valid settings in the CGU_CTL and CGU_DIVregisters. These registers control clock multiplier and divisor values. These registers must be set such that the minimum and maximum clock 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 part.

ADSP-BF60x CGU Register Descriptions

Clock Generation Unit (CGU) contains the following registers.

Table 3-6: ADSP-BF60x CGU Register List

Name	Description
CGU_CTL	Control Register
CGU_STAT	Status Register
CGU_DIV	Clocks Divisor Register
CGU_CLKOUTSEL	CLKOUT Select Register

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.

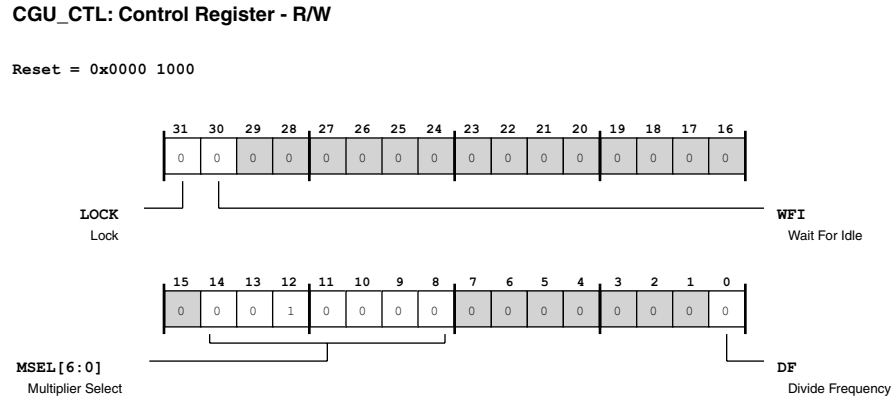


Figure 3-3: CGU_CTL Register Diagram

Table 3-7: CGU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the CGU_CTL.LOCK bit is set, the CGU_CTL 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 CGU_CTL.MSEL and the CGU_CTL.DF result in bypassing the PLL. The CGU_CTL.WFI force the PLL to wait for all processor cores to be in an idle or reset state before changing frequencies as a result of change to the CGU_CTL.MSEL or CGU_CTL.DF fields. Write accesses to CGU_CTL to change CGU_CTL.DF or CGU_CTL.MSEL while the PLL is locking sets the CGU_STAT.WDFMSERR bit.	
		0	Update Immediately
		1	Wait for Idle

Table 3-7: CGU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14:8 (R/W)	MSEL	Multiplier Select. The CGU_CTL.MSEL selects the multiplier in the PLLCLK equation: PLLCLK frequency = (SYS_CLKIN frequency / (DF+1)) * MSEL Where the value of MSEL may be between 1 and 127. Note that supported MSEL values are product specific. See the product specific section of the CGU for additional notes.	
		xxxxxxx	MSEL = 1 to 127
		0	Reserved
0 (R/W)	DF	Divide Frequency. The CGU_CTL.DF selects whether or not the SYS_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

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.

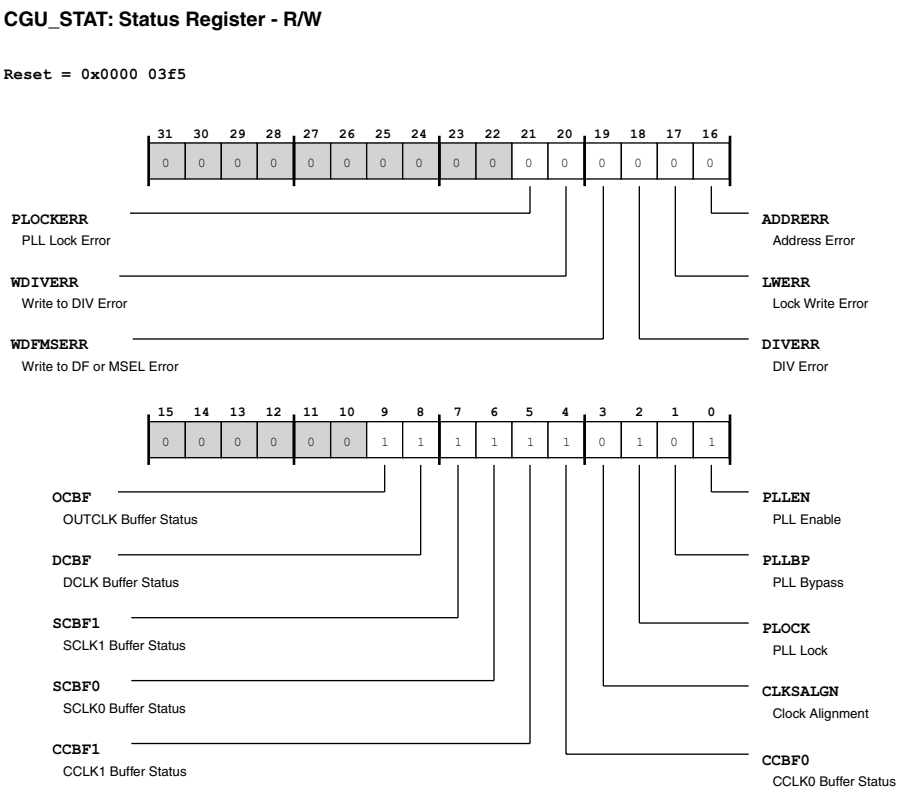


Figure 3-4: CGU_STAT Register Diagram

Table 3-8: CGU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W1C)	PLOCKERR	PLL Lock Error. The CGU_STAT.PLOCKERR indicates that the PLL failed to lock.	
		0	No Error
		1	PLL Lock Error
20 (R/W1C)	WDIVERR	Write to DIV Error. The CGU_STAT.WDIVERR indicates a write access to the CGU_DIV register (to trigger an alignment sequence or to change CGU_DIV.CSEL, CGU_DIV.SYSSEL, CGU_DIV.SOSEL, CGU_DIV.S1SEL, or CGU_DIV.DSEL) 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

Table 3-8: CGU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W1C)	WDFMSERR	Write to DF or MSEL Error. The CGU_STAT.WDFMSERR indicates a write access to the CGU_CTL register to change CGU_CTL.DF or CGU_CTL.MSEL while the PLL is locking.	
		0	No Error
		1	Write DF/MSEL Error
18 (R/W1C)	DIVERR	DIV Error. The CGU_STAT.DIVERR indicates a clock divisor value error, occurring when the CCLK clock divisor is greater than the SYSCLK clock divisor, as in: CGU_DIV.CSEL> CGU_DIV.SYSSEL The CGU issues a CGU error for this condition.	
		0	No Error
		1	DIV Error
17 (R/W1C)	LWERR	Lock Write Error. The CGU_STAT.LWERR 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 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
9 (R/NW)	OCBF	OUTCLK Buffer Status. The CGU_STAT.OCBF indicates whether the OUTCLK buffer is enabled.	
		0	Disabled
		1	Enabled
8 (R/NW)	DCBF	DCLK Buffer Status. The CGU_STAT.DCBF indicates whether the DCLK buffer is enabled.	
		0	Disabled
		1	Enabled

Table 3-8: CGU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	SCBF1	SCLK1 Buffer Status. The CGU_STAT.SCBF1 indicates whether the SCLK1 buffer is enabled.	
		0	Disabled
		1	Enabled
6 (R/NW)	SCBF0	SCLK0 Buffer Status. The CGU_STAT.SCBF0 indicates whether the SCLK0 buffer is enabled.	
		0	Disabled
		1	Enabled
5 (R/NW)	CCBF1	CCLK1 Buffer Status. The CGU_STAT.CCBF1 indicates whether the CCLK1 buffer is enabled.	
		0	Disabled
		1	Enabled
4 (R/NW)	CCBF0	CCLK0 Buffer Status. The CGU_STAT.CCBF0 indicates whether the CCLK0 buffer is enabled.	
		0	Disabled
		1	Enabled
3 (R/NW)	CLKSALGN	Clock Alignment. The CGU_STAT.CLKSALGN indicates whether a clock alignment sequence is in progress. This bit is set when clocks alignment is required by changes to CGU_DIV.CSEL, CGU_DIV.SOSEL, CGU_DIV.S1SEL, CGU_DIV.DSEL, or CGU_DIV.OSEL. The CGU_STAT.CLKSALGN bit is cleared when clocks are aligned. Note that (after a PLL frequency change in active state) the CGU_STAT.CLKSALGN bit may indicate that clocks are not aligned even though the clocks are aligned (all clocks are aligned and running at the SCLKIN frequency).	
		0	Clocks are Aligned
		1	Clocks not Aligned (alignment in progress)

Table 3-8: CGU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/NW)	PLOCK	PLL Lock. The CGU_STAT.PLOCK indicates whether the PLL is locked. This bit is set when the PLL locks (PLL lock counter end-of-count). The CGU_STAT.PLOCK bit is cleared when requested PLL frequency change (for PLL reset, PLL disable-to-enable transition, or change to CGU_CTL.MSEL or CGU_CTL.DF) is in progress.	
		0	PLL not Locked (PLL frequency change in progress)
		1	PLL Locked
1 (R/NW)	PLLBP	PLL Bypass. The CGU_STAT.PLLBP indicates whether the PLL is bypassed. The default value for CGU_STAT.PLLBP is determined by the PLL bypass state.	
		0	PLL not Bypassed
		1	PLL Bypassed
0 (R/NW)	PLLEN	PLL Enable. The CGU_STAT.PLLEN indicates whether the PLL is enabled.	
		0	Disabled
		1	Enabled

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.

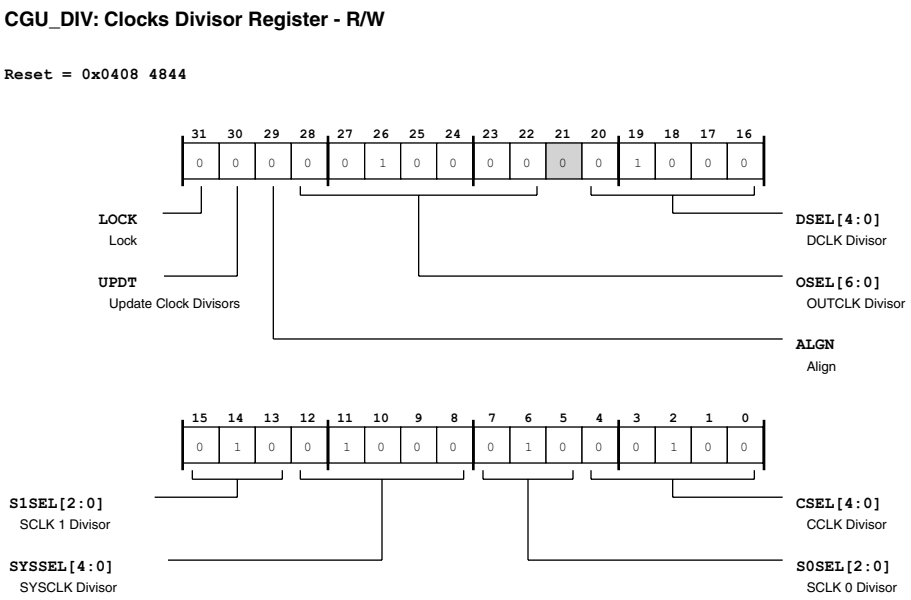


Figure 3-5: CGU_DIV Register Diagram

Table 3-9: CGU_DIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the CGU_DIV.LOCK bit is set, the CGU_DIV register is read only (locked).	
		0	Unlock
		1	Lock
30 (R/W)	UPDT	Update Clock Divisors. The CGU_DIV.UPDT controls whether the CGU drives new CGU_DIV.CSEL, CGU_DIV.SYSSEL, CGU_DIV.SOSEL, CGU_DIV.S1SEL, CGU_DIV.DSEL, and CGU_DIV.OSEL values to PLL after CGU_DIV register update.	
		0	No PLL Update
		1	Drive Updated SEL Values to PLL
29 (R0/W1A)	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.SOSEL, CGU_DIV.S1SEL, CGU_DIV.DSEL, and/or CGU_DIV.OSEL) do not have to change.	
		0	No Action
		1	Align PLL Clocks

Table 3-9: CGU_DIV Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
28:22 (R/W)	OSEL	OUTCLK Divisor. The CGU_DIV.OSEL selects the divisor in the OUTCLK equation: OUTCLK frequency = (SYS_CLKIN frequency / (DF+1)) * MSEL / CGU_DIV.OSEL Where the value of CGU_DIV.OSEL is between 1 and 127.	
		xxxxxxx	OSEL = 1 to 127
		0	Reserved
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	Reserved
		xxxxx	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	Reserved
		xxx	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	Reserved
		xxxxx	SYSSEL = 1 to 31
7:5 (R/W)	S0SEL	SCLK 0 Divisor. The CGU_DIV.S0SEL selects the divisor in the SCLK0 equation: SCLK0 frequency = (SYSCLK frequency) / CGU_DIV.S0SEL Where the value of CGU_DIV.S0SEL is between 1 and 7.	
		0	Reserved
		xxx	S0SEL = 1 to 7

Table 3-9: CGU_DIV Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4:0 (R/W)	CSEL	CCLK Divisor. The CGU_DIV.CSEL selects the divisor in the CCLK equation: CCLK frequency = (SYS_CLKIN frequency / (DF+1)) * MSEL / CGU_DIV.CSEL Where the value of CGU_DIV.CSEL is between 1 and 31.	
		0	Reserved
		xxxxx	CSEL= 1 to 31

CLKOUT Select Register

The CGU_CLKOUTSEL selects the signal that the CGU drives through the CLKOUT multiplexer.

CGU_CLKOUTSEL: CLKOUT Select Register - R/W

Reset = 0x0000 0000

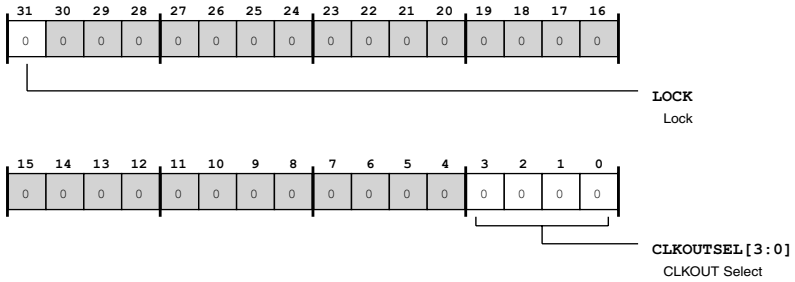


Figure 3-6: CGU_CLKOUTSEL Register Diagram

Table 3-10: CGU_CLKOUTSEL Register Fields

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

Table 3-10: CGU_CLKOUTSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/W)	CLKOUTSEL	CLKOUT Select. The CGU_CLKOUTSEL.CLKOUTSEL selects the signal that the CGU drives through the CLKOUT pin multiplexer.	
		0	CLKIN
		1	CCLKn/4
		2	SYSCLK/2
		3	SCLK0
		4	SCLK1
		5	DCLK/2
		6	Reserved
		7	OUTCLK
		8	Reserved
		9	Reserved
		10	Reserved
		11	GND (Disable OUTCLK)
		11xx	Reserved

4 System Protection Unit (SPU)

The system protection unit (SPU) provides features that let you protect system resources from errant writes. A number of protection categories (types of registers to protect) are available.

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 lets the user restrict access to certain MMRs, similar to the functionality of a semaphore.

SPU Features

The System Protection Unit has the following features.

- Write-protect system MMR from certain system masters.
- Simultaneously lock multiple peripheral configuration registers.
- Write-protect and block access to its own write-protection registers from other system masters.

SPU Functional Description

The SPU has a register associated with each peripheral. Each of these write-protection registers has the exact same bits that correspond to a particular SMMR master (Core 0, Core 1, MDMA, for example). When the bits are set, the corresponding SMMR masters are locked out of accessing the associated peripheral's register address space. The bits in the register can be cleared to allow access to the peripheral's registers again. Any writes that are in progress when write-protection is initiated are completed before subsequent writes are blocked.

In the following figure, each write-protect register in the SPU is associated with a particular peripheral.

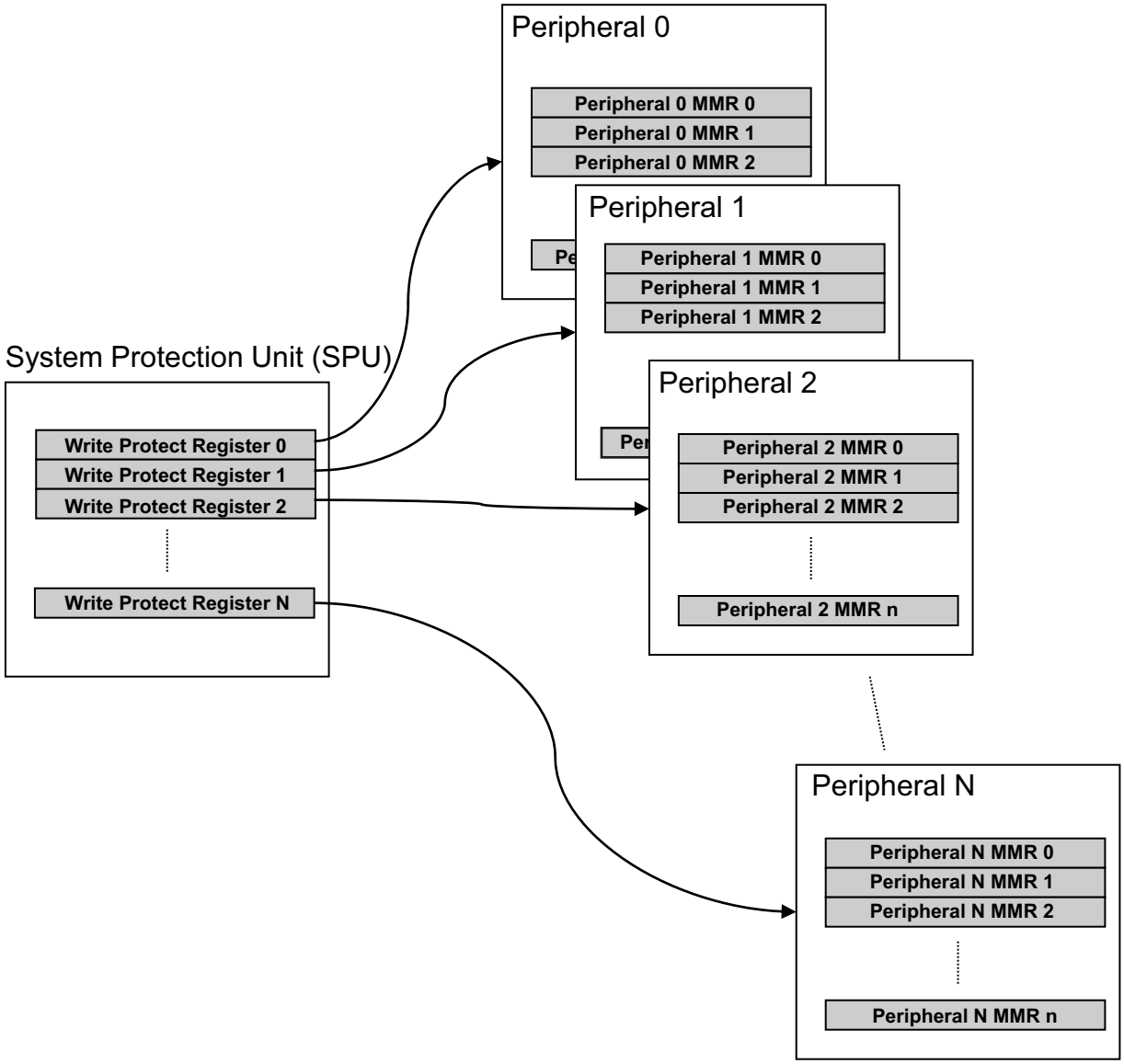


Figure 4-1: SPU Write Protect Registers

The SPU also has global locking capability. When enabled, 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 peripheral's configuration register is re-enabled when global lock is turned off in the SPU.

The following figure is a conceptual diagram where a peripheral blocks any write attempts to its control register if the global lock signal from the SPU is active AND the global lock enable bit is set in the peripheral's control register.

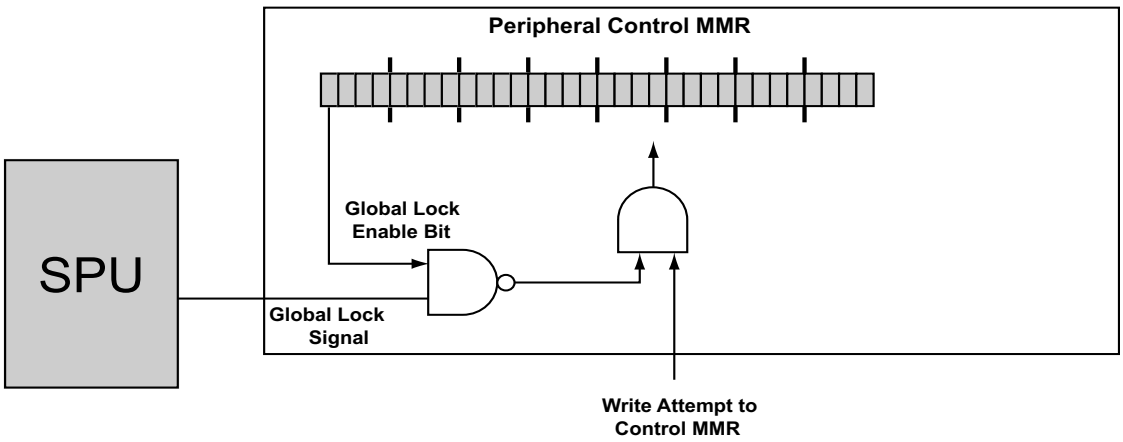


Figure 4-2: Global Locking

The SPU can write protect its own registers. When the write protection register lock bit is set and global locking is enabled, accesses to the SPU write-protection registers are blocked. To re-enable write access to the write-protection registers in the SPU, global locking must be disabled.

In the following figure a write-protect register in the SPU blocks write-attempts to the associated peripheral's MMR space. The bits in the write-protect register specify which masters to block write-access from.

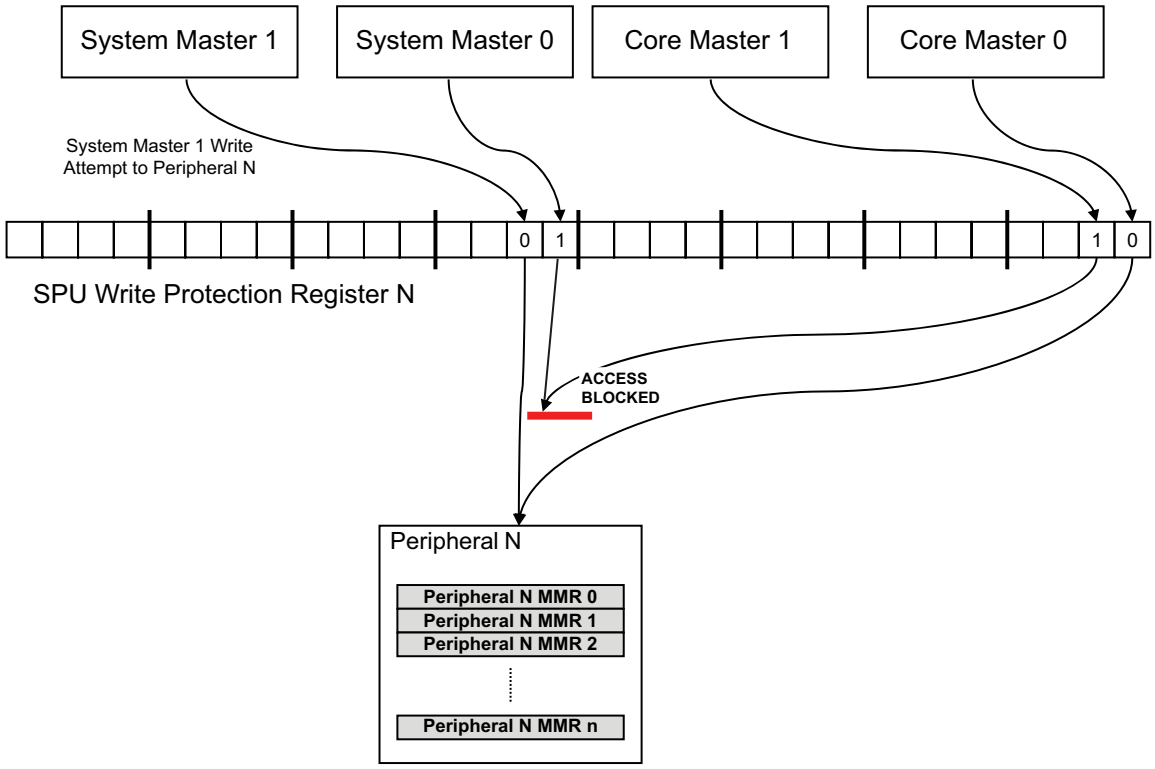


Figure 4-3: SPU Write-Protect Register Blocking Access from System Master 0 and Core Master 1

ADSP-BF60x SPU Register List

The system protection unit (SPU) provides a set of registers that allow you to 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 4-1: ADSP-BF60x SPU Register List

Name	Description
SPU_CTL	Control Register
SPU_STAT	Status Register
SPU_WPn	Write Protect Register n

SPU Definitions

Write-Protect Register

Memory mapped registers in the SPU. Each register correlates to a specific peripheral instance. It controls the write access to the peripheral's register set.

Global Locking

SPU's ability to prevent write access to multiple peripheral's control register at once.

SPU Block Diagram

The figure below shows a system level block diagram of where the SPU is in the system. It sits in between the SMMR interface and the system crossbar. Depending on the configuration of the SPU write-protect registers, it can block access to certain peripherals from certain SMMR masters.

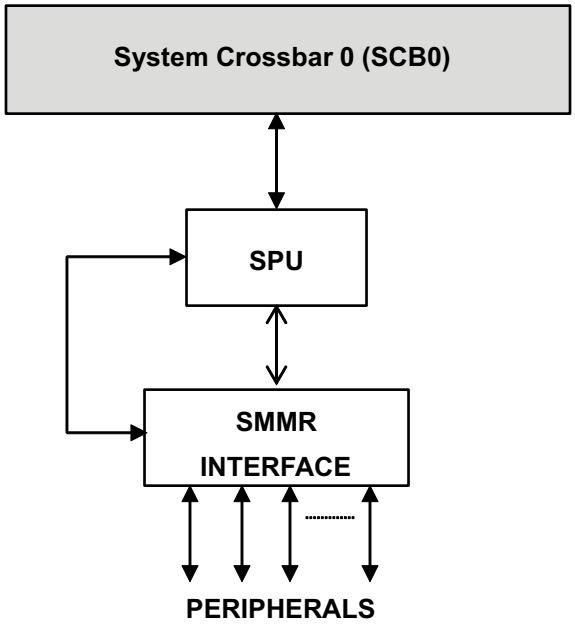


Figure 4-4: SPU System Level Block Diagram

SPU Architectural Concepts

As shown in the block diagram, the SPU sits between the System Crossbar (SCB) and the SMMR interface to the peripherals. Any MMR access to any peripheral from any master comes through the SCB and is gated by the SPU. Depending on the configuration of the write-protection registers in the SPU, the SPU may or may not allow the MMR write to go through.

SPU Event Control

The system protection unit provides write protection against a peripheral's MMRs and its own write-protect registers. If a write attempt is made to any peripheral's MMR and was locked, the SPU will block the write and generate a bus error to the master that attempted the write. That master may or may not generate an event based upon the returned error. The SPU does not generate an event for blocked write attempts.

The SPU can also lock its own registers from write attempts. If a write-attempt was made to a locked register in the SPU, the SPU blocks it and records it as an error in `SPU_STAT.LWERR`. Again, the SPU generates a bus error to the master that attempted the write. The master may or may not generate an event based upon the returned error. The SPU does not generate an event for a blocked write access to an SPU register.

SPU Programming Model

The system protection unit (SPU) consists of write-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 peripheral's memory-mapped registers are write protected against masters whose bits in the write-protect register have been set.

Another capability of the SPU is to globally lock peripherals' control register. Peripherals that support this feature have a lock enable bit in their control register. When the global lock signal is active from the SPU and the peripheral's lock enable bit is set, the peripheral blocks any more write attempts to its control register from any master. If the lock enable bit of a peripheral is not set and the global lock signal is active, access to that peripheral's control register is still allowed. To grant access again, the global lock signal from the SPU must be disabled 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 SPU's write-protect registers will be blocked. To re-enable access to the write-protect registers in the SPU, the global lock signal must be deactivated by writing 0xAD into the `SPU_CTL.GLCK` bit field.

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 by using Global Locking. Peripherals that support this feature can enable it their respective configuration register. When the SPU enables the Global Lock signal, all subsequent writes to the peripheral's configuration register are blocked until the Global Lock signal is deasserted. Similarly, the SPU's own write-protection registers can be write protected using the Global Lock signal as well. All these modes of operation can be used in conjunction.

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.

RESULT:

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 in the SPU.
2. Determine which SMMR master(s) the peripheral needs to be protected from and set the corresponding bit(s) in the write-protect register for the peripheral in the SPU.

RESULT:

After setting the write-protect register for the particular peripheral, the SMMR master(s) will be blocked from writing to any MMR in the peripheral's address space until the bits in the write-protect register are cleared.

ADSP-BF60x SPU Register Descriptions

System Protection Unit (SPU) contains the following registers.

Table 4-2: ADSP-BF60x SPU Register List

Name	Description
SPU_CTL	Control Register
SPU_STAT	Status Register
SPU_WPn	Write Protect Register n

Control Register

The SPU control register (SPU_CTL) provides a global lock for configuration registers and write protection for registers.

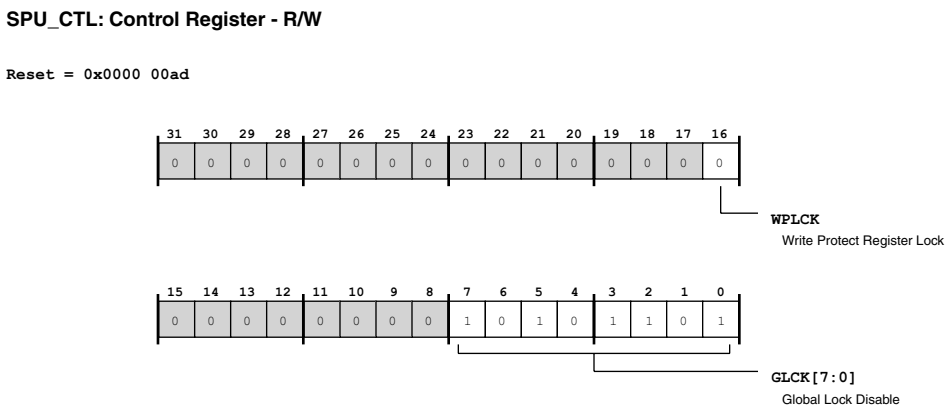


Figure 4-5: SPU_CTL Register Diagram

Table 4-3: SPU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	WPLCK	Write Protect Register Lock. The SPU_CTL.WPLCK works with the SPU_CTL.GLCK field. If the write protect register lock is enabled (SPU_CTL.WPLCK bit =1) and the global lock is enabled, writes to the SPU_WPn register are disabled (locked out).	
		0	Disable
		1	Enable
7:0 (R/W)	GLCK	Global Lock Disable. The SPU_CTL.GLCK controls the global lock of configuration registers. Writing 0xAD to this field disables the lock, and writing any other value enables the lock.	

Status Register

The SPU_STAT register indicates the error and lock status for the SPU.

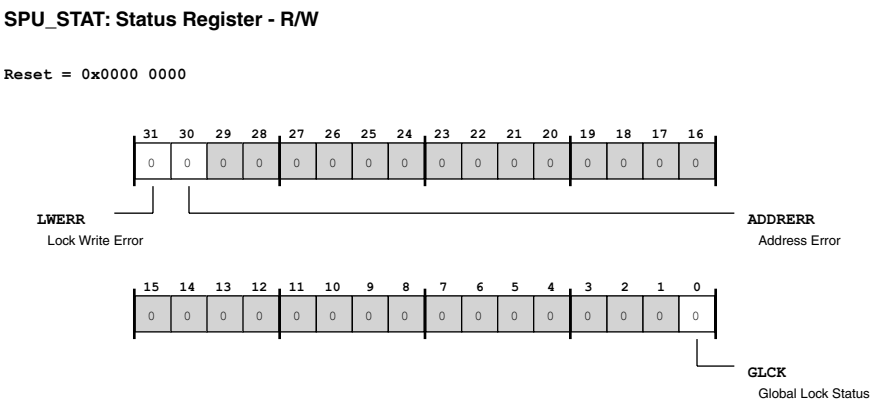


Figure 4-6: SPU_STAT Register Diagram

Table 4-4: SPU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W1C)	LWERR	Lock Write Error. The SPU_STAT.LWERR is write-1-to-clear and indicates whether there has been an attempted write to a register with its LOCK bit set while SPU_CTL.GLCK was asserted.	
		0	Inactive
		1	Active
30 (R/W1C)	ADDRERR	Address Error. The SPU_STAT.ADDRERR is write-1-to-clear and indicates whether there has been an attempted write to a read-only register or an access an invalid address.	
		0	Inactive
		1	Active
0 (R/NW)	GLCK	Global Lock Status. The SPU_STAT.GLCK indicates whether the global lock is enabled or disabled.	
		0	Disabled (global_lock=0)
		1	Enabled (global_lock=1)

Write Protect Register n

In the system, each SPU_WPn 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_WPn register.

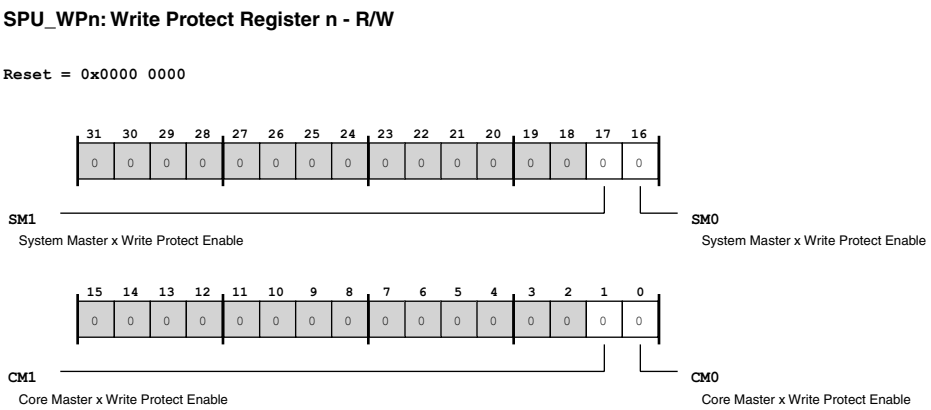


Figure 4-7: SPU_WPn Register Diagram

Table 4-5: SPU_WPn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17:16 (R/W)	SMn	System Master x Write Protect Enable.
1:0 (R/W)	CMn	Core Master x Write Protect Enable.

ADSP-BF60x SPU_WPn Additional Information

The SPU consists of a collection of Write Protect Registers each of which are associated with a specific peripheral or slave. The table gives the Write Protect Register number for each of the peripherals that are provided with write protection through the SPU. The SPU for ADSP-BF60x is configured with 86 Write Protect Registers.

For each processor, there will be different number of masters that will be able to access the SMMR space. Accordingly the bit definitions for the Write Protect Registers will be different for each processor. The ADSP-BF60x processor has four masters that can access the SMMR space. The table below shows which bits enable the protection against which master.

Table 4-6: SPU_WPn.CMn and SPU_WPn.SMn Bits

Bit Number	Bit Name	Description
0	CM0_WP (Core Master 0)	CoreA
1	CM1_WP (Core Master 1)	CoreB
16	SM0_WP (System Master 0)	SDU
17	SM1_WP (System Master 1)	MDMA

For each peripheral, there will be a corresponding write-protect register, SPU_WPn. The table shows the Write Protect Register number for each peripheral.

Table 4-7: SPU_WPn Registers and Related Peripherals

Write Protect Register Number (n)	Peripheral
0	Counter
1	RSI
2	CAN
3	LP0
4	LP1
5	LP2
6	LP3
7	TIM
8	CRC0
9	CRC1
10	TWI0
11	TWI1
12	UART0
13	UART1
14	PORTA/B
15	PORTC/D
16	PORTE/F
17	PORTG
18	PINT0
19	PINT1
20	PINT2
21	PINT3
22	PINT4
23	PINT5
24	SCB2
25	SCB2_ARB
26	SCB3
27	SCB3_ARB
28	SCB4

Table 4-7: SPU_WPn Registers and Related Peripherals (Continued)

Write Protect Register Number (n)	Peripheral
29	SCB4_ARB
30	SCB5
31	SCB5_ARB
32	SCB6
33	SCB6_ARB
34	SCB7
35	SCB7_ARB
36	SCB8
37	SCB8_ARB
38	SCB9
39	SCB9_ARB
40	SMC
41	WDT0
42	WDT1
43	EPPI0
44	EPPI1
45	EPPI2
46	PIXC
47	PVP
48	EPWM0
49	EPWM1
50	VSS_Crossbar
51	SWU 0 (SMC)
52	SDU
53	EMAC0
54	EMAC1
55	SPORT0_A
56	SPORT0_B
57	SPORT1_A
58	SPORT1_B
59	SPORT2_A

Table 4-7: SPU_WPn Registers and Related Peripherals (Continued)

Write Protect Register Number (n)	Peripheral
60	SPORT2_B
61	SPI0
62	SPI1
63	SCB1
64	SCB1_ARB
65	ACM
66	DDR2
67	SCB12_ARB
68	SWU6 (DDR2)
69	SCB11_ARB
70	SCB10_ARB
71	SCB0_ARB
72	L2
73	SEC
74	TRU
75	RCU
76	SPU
77	PCU
78	DPM
79	SWU1 (L2_S)
80	SWU2 (L2_C)
81	SWU3 (Core0)
82	SWU4 (Core1)
83	SWU5 (SMMR)
85	USB

5 Dynamic Power Management (DPM)

The dynamic power management (DPM) unit of the processor controls transitions between different power saving modes. The DPM also allows individual clock domains to be enabled and disabled.

DPM Features

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

- Provides capability to shut off individual clock domains to save power
- Supports capability to bypass the PLL for power savings
- Aids power savings through hibernate mode, which allows `VDD_INT` supply to be shut off
- Permits operation of multiple, external wake-up sources

DPM Functional Description

The processor supports a number of power domains, which maximizes flexibility while maintaining compliance with industry standards and conventions. By isolating the internal logic of the processor into its own power domain, separate from other I/O, the processor can take advantage of dynamic power management without affecting the other I/O devices. There are no sequencing requirements for the various power domains, but all domains must be powered according to the appropriate specifications, even if the feature/peripheral is not used. For more information on power domains, see the processor data sheet.

The dynamic power management feature of the processor allows the processor's core clock frequency (f_{CCLK}) to be dynamically controlled.

ADSP-BF60x DPM Register List

The dynamic power management (DPM) unit includes the phase locked loop (PLL) enable/disable features, deep sleep and hibernate mode controls, and clock domain enable/disable features. The combination of these features and controls provide selective and flexible power management. A set of registers govern DPM operations. For more information on DPM functionality, see the DPM register descriptions.

Table 5-1: ADSP-BF60x DPM Register List

Name	Description
DPM_CTL	Control Register
DPM_STAT	Status Register
DPM_CCBF_DIS	Core Clock Buffer Disable Register
DPM_CCBF_EN	Core Clock Buffer Enable Register
DPM_CCBF_STAT	Core Clock Buffer Status Register
DPM_CCBF_STAT_STKY	Core Clock Buffer Status Sticky Register
DPM_SCBF_DIS	System Clock Buffer Disable Register
DPM_WAKE_EN	Wakeup Enable Register
DPM_WAKE_POL	Wakeup Polarity Register
DPM_WAKE_STAT	Wakeup Status Register
DPM_HIB_DIS	Hibernate Disable Register
DPM_PGCNTR	Power Good Counter Register
DPM_RESTOREn	Restore Registers

ADSP-BF60x DPM Interrupt List

Table 5-2: ADSP-BF60x DPM Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
DPM0 Event	131		LEVEL

DPM Definitions

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

Active mode

A power saving mode in which the PLL is bypassed but still enabled.

Active mode with PLL disabled

A power saving mode in which the PLL is bypassed and disabled.

CGU

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

Deep sleep mode

A power saving mode in which all CCLKs are gated.

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.

Hibernate mode

A power saving mode in which the `VDD_INT` supply can be shut off, and the contents of on-chip memory are not retained.

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
- ACTIVE
- ACTIVE with PLL Disabled

- DEEP SLEEP
- HIBERNATE

The operating modes and transitions figure shows the relationships between DPM modes for the ADSP-BF60x processor.

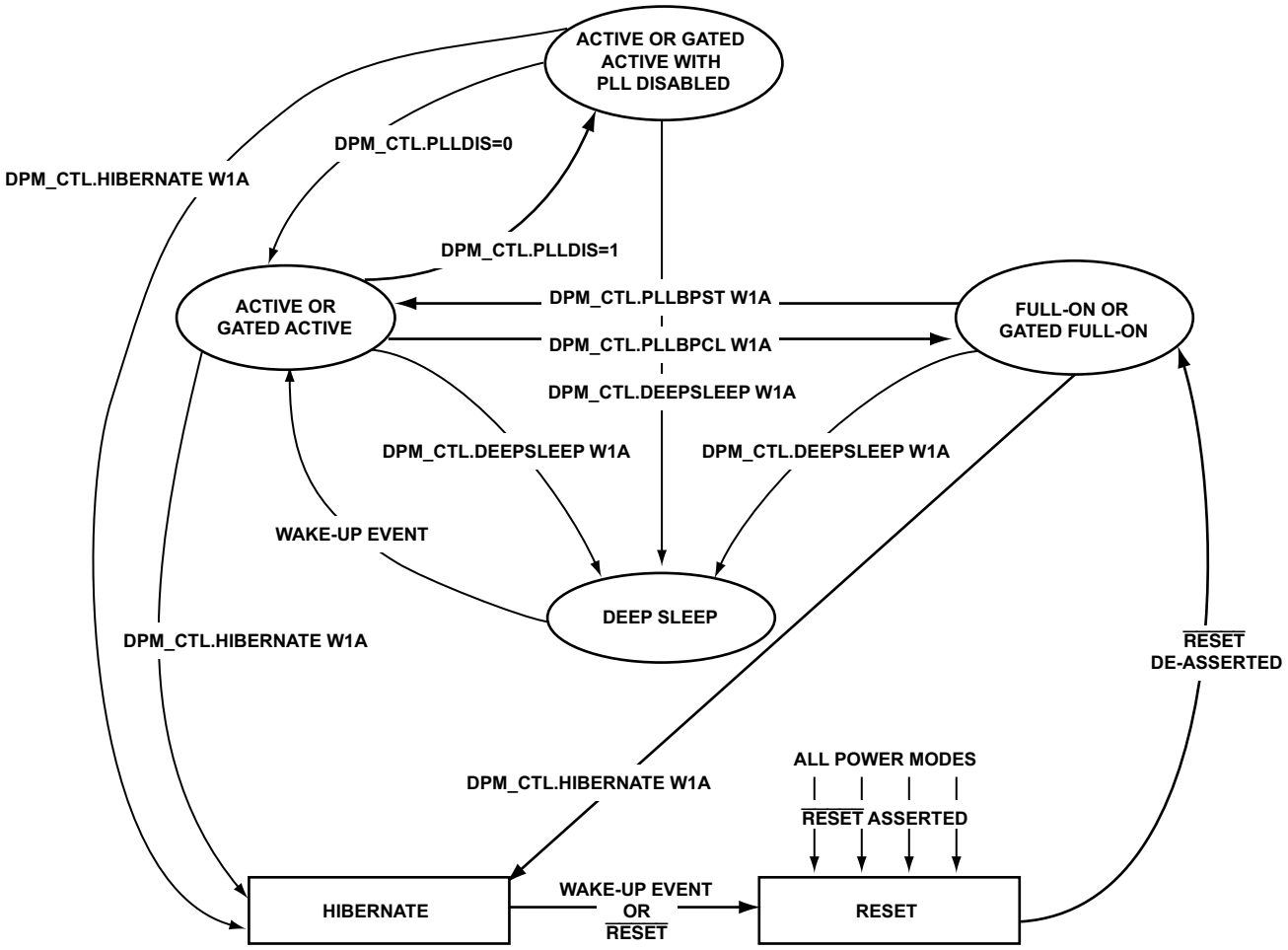


Figure 5-1: Operating Modes and Transitions

Reset State

Reset is the initial state of the processor and is the result of a hardware or software triggered event. Entering reset is not triggered by the DPM itself, but by the external SYS_HWRST pin or by the RCU. The DPM responds to reset by transitioning to its default state.

Certain registers (see HIBERNATE) are exceptions to this return to default state and are preserved if the DMP is returning from hibernate mode.

From RESET, the DPM always transitions to FULL-ON.

Full-on Mode

Full-on mode is the DPM's default state after RESET.

In full-on mode, the processor can reach its maximum clock rate and power dissipation can be at its highest. The DPM transitions from full-on mode to:

- Active mode if DPM_CTL. PLLBPST is set
- Deep sleep mode if DPM_CTL. DEEPSLEEP is set
- Hibernate mode if DPM_CTL. HIBERNATE is set

Active Mode

In active mode, power dissipation is reduced on the VDD_INT power domain (compared to full-on mode) by bypassing the PLL and running clock domains at the SYS_CLKIN pin frequency. The processor is fully functional. The DPM transitions from active mode to:

- Full-on mode if DPM_CTL. PLLBPCL is set
- Active with PLL disabled mode if DPM_CTL. PLLDIS is set
- Deep sleep mode if DPM_CTL. DEEPSLEEP is set
- Hibernate mode if DPM_CTL. HIBERNATE is set

ACTIVE with PLL Disabled

In active with PLL disabled mode, power dissipation is reduced on the VDD_INT power domain (compared to active mode) by disabling the PLL in addition to running all units at the at the SYS_CLKIN pin frequency. The processor is fully functional. The DPM transitions from active with PLL disabled mode to:

- Active mode if DPM_CTL.PLLDIS is cleared
- Deep sleep mode if DPM_CTL.DEEPSLEEP is set
- Hibernate mode if DPM_CTL.HIBERNATE is set

Deep Sleep Mode

To enter deep sleep mode, the processor sets the DPM_CTL.DEEPSLEEP bit, and all processor cores are in idle state. It is the programs responsibility in software to guarantee that system transfers including DMA are stopped before each processor core goes into idle state and the processor enters deep sleep mode. In this state, power dissipation on the VDD_INT power domain is reduced (compared to active mode or gated active mode) by gating all the core and system clocks and by disabling the PLL.

The enabled hardware wake-up signals or a hardware reset signal can make the processor exit deep sleep mode. The DPM_WAKE_EN.WSn bits and DPM_WAKE_POL.WSn bits work together to determine which hardware wake-up signals are enabled and the signals' polarity. Wake-up signal assertion is latched only when the signal is enabled. The enabled wake-up signal assertion occurring first is recorded in the DPM_WAKE_STAT register.

NOTE: To see which wake-up sources your processor reports, see *ADSP-BF60x Wake-Up Sources*.

When a wake-up occurs, the DPM does the following:

- Signals a DPM event interrupt to the SEC
- Transitions to ACTIVE mode
- Enables all clocks domains that are not disabled in the DPM_SCBF_DIS register

The DPM event interrupt will stay active until the user clears any bits that are set in DPM_WAKE_STAT. The DPM event interrupt is the first indication that the processor has exited DEEP SLEEP.

One option for waking up the core is to enable the CGU event interrupt, which asserts after the PLL locks.

Another option is to use the enable in the SEC to make a core exit idle and to enable the corresponding core clock buffer.

Hibernate Mode

In hibernate mode, there is no power dissipation on the VDD_INT power domain as long as the voltage regulator powering VDD_INT is shut off. If state information, data, or code needs to be preserved, it can be stored in the DPM_RESTOREn registers or in external memory. If dynamic memory is used the memory device needs to be placed into self-refresh mode before going to hibernate.

Hibernate mode is entered by setting the DPM_CTL.HIBERNATE bit. It is the programs responsibility in software to guarantee that system transfers including DMA are stopped before entering hibernate mode.

After the DPM_CTL.HIBERNATE bit has been set, the DPM preserves the state of the DPM_WAKE_EN, DPM_WAKE_POL, DPM_HIB_DIS, DPM_PGCNTR, and all of the DPM_RESTOREn registers in the DPM-HV, which is powered by the VDD_EXT power domain. All other registers (with the exception of those mentioned in the RCU chapter) and on-chip memory are erased when the VDD_INT supply is powered down. After the contents of the registers listed above are moved to storage on the VDD_EXT power domain, the SYS_EXTWAKE signal is de-asserted to indicate that it is safe to shut off the regulator providing the VDD_INT supply.

Enabled hardware wake-ups or a hardware reset can begin the process of exiting hibernate mode. The DPM_WAKE_EN.WSn bits and DPM_WAKE_POL.WSn bit determine which hardware wake ups are enabled and select their signal polarity. Wake-up assertions are latched only if they are enabled. The enabled wake-up assertion that occurs first is recorded in the DPM_WAKE_STAT register. After a wake-up source assertion is latched by the processor, the SYS_EXTWAKE signal is asserted to indicate that the regulator for the VDD_INT supply should begin to ramp up. The DPM waits for assertion of the SYS_PWRGD signal and waits for expiration of

the DPM_PGCNTR.CNT counter. These conditions indicate that the VDD_INT supply has been restored within the operating conditions specified in the data sheet. For more information, see *Ensuring Internal Logic Supply is Restored Before Booting*.

After the VDD_INT supply is restored, the SYS_PWRGD signal is asserted, and the DPM_PGCNTR.CNT is expired; the processor exits hibernate mode, goes into reset state, and the PLL begins locking. While the PLL is locking, the DPM_WAKE_EN, DPM_WAKE_POL, DPM_HIB_DIS, DPM_PGCNTR, and all of the DPM_RESTOREn registers are restored from the DPM-HV. After coming out of reset mode and locking the PLL, the processor begins executing the boot code. Some processors support memory boot when returning from hibernate (instead of the boot mode selected by the SYS_BMODE signals). For more information see the booting chapter.

DPM Event Control

The DPM event is triggered when an enabled wake-up is asserted. DPM bus errors are generated when a misaligned access to a registers occurs or 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_WAKE_STAT register is set. This indicates that an enabled wake-up was asserted. The DPM event interrupt will stay active until the user clears any bits that are set in the DPM_WAKE_STAT register.

DPM Errors

The DPM generates a bus error if a read or write transaction is attempted to an unused address within the DPM address range or if 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 following sections describe programming techniques, including verifying restoration of power supplies, managing power modes, and selecting wake-up sources.

- *Ensuring Internal Logic Supply is Restored Before Booting*
- *Configuring Deep Sleep Mode*
- *Configuring Hibernate Mode*

- *ADSP-BF60x Wake-Up Sources*
- *ADSP-BF60x Clock Buffer Disable Bit Assignments*
- *ADSP-BF60x Hibernate Disable Bit Assignments*

Ensuring Internal Logic Supply is Restored Before Booting

Before the processor boots after returning from hibernate mode, the internal logic supply (VDD_INT power domain) must be restored. This topic discusses methods for signalling the processor that the VDD_INT power domain has been restored.

After a wake-up occurrence is latched by the DPM while in hibernate mode, the DPM power good counter will begin decrementing and the SYS_EXTWAKE signal will be asserted to indicate to the regulator supplying the VDD_INT pins that it should begin to ramp up. It is the user's responsibility to then signal the DPM through software (the DPM power good counter in the DPM_PGCNTR register) or hardware (the SYS_PWRGD signal) that it is safe to begin the boot process. Note that the DPM does not start the boot process until SYS_PWRGD is asserted and the DPM_PGCNTR.CNT value is expired.

Using the PG Counter to Check Internal Logic Supply is Restored

In order to use the software approach, which utilizes the DPM power good counter in the DPM_PGCNTR register, it is suggested that SYS_PWRGD be pulled up to the VDD_EXT voltage. The DPM will then have to wait only for the DPM power good counter to expire before kicking off the boot process. It is the user's responsibility to select the appropriate setting for DPM_PGCNTR.CNT value such that when the counter expires, the VDD_INT power domain has been restored to within the operating conditions specified in the data sheet. The amount of time that the counter takes to expire is dependent on both the DPM_PGCNTR.CNT value and the SYS_CLKIN frequency according to this equation:

Expiration Time [ms] = Cycle Count * 16 * SYS_CLKIN period [ms]

The text in brackets indicates that both the expiration time and SYS_CLKIN period are expressed in milliseconds. To determine the cycle count, refer to the following table which shows cycle counts for the different DPM_PGCNTR.CNT value selections.

Table 5-3: Cycle Counts for DPM_PGCNTR.CNT Settings

DPM_PGCNTR.CNT **Values**	**Cycle Count**
0x0004	128 (Default)
0x0080	4,096
0x0100	8,192
0x0200	16,384
0x0400	32,768
0x0800	65,536

Table 5-3: Cycle Counts for DPM_PGCNTR.CNT Settings (Continued)

DPM_PGCNTR.CNT Values	Cycle Count
0x1000	131,072
0x2000	262,144
0x4000	524,288
0x8000	1,048,576
All other values not listed above	Reserved

The following table shows examples of different SYS_CLKIN frequencies and DPM_PGCNTR.CNT values and the resulting power good counter expiration times. Note that these are example calculations only and that the SYS_CLKIN frequency selected must be within the range allowed by the data sheet for your processor.

Table 5-4: Example DPM Counter Expiration Time Calculations

DPM_PGCNTR.CNT Setting	Cycle Count (Cycles)	SYS_CLKIN Frequency (MHz)	SYS_CLKIN Period (ms)	Expiration Time (ms)
0x0400	32,768	50	0.00002	10.45696
0x0800	65,536	40	0.000025	26.2144
0x4000	524,288	25	0.00004	335.54432

Using the PG Input to Check Internal Logic Supply is Restored

Voltage regulators commonly have a power good output, which can be directly connected to the SYS_PWRGD input of the processor. It is the user's responsibility to make sure that when the power good output of the regulator is asserted, the VDD_INT power domain has been restored to within the operating conditions specified in the processor's data sheet. If the regulator's power good output asserts before the VDD_INT power domain has been restored to within the operating conditions specified in the processor's data sheet, then the power good counter must be used to ensure that the VDD_INT power domain has been restored to within specifications.

When the SYS_PWRGD signal is used to indicate the VDD_INT power domain is restored, it is suggested that the DPM_PGCNTR.CNT value be left at its default setting. The DPM then waits until the default DPM_PGCNTR.CNT value is expired and the SYS_PWRGD signal is asserted before kicking off the boot process.

Configuring Deep Sleep Mode

The deep sleep mode gates all core and system clocks in order to save power.

PREREQUISITE:

The deep sleep mode can be entered from any state in which software can run. Reading the DPM_STAT.CURMODE field reveals the current power mode. Clocks do not stop immediately after entry to deep sleep mode is requested, but no further action is needed to guarantee that the mode transition occurs.

The processor cores need to be idle before the clocks are shut down.

1. If the DPM_STAT.CURMODE indicates full-on mode, wait for the CGU_STAT.PLLBP bit =0, the CGU_STAT.PLOCK bit =1, and the CGU_STAT.CLKSALGN bit =0.
2. If DPM_STAT.CURMODE indicates active mode with PLL disabled, wait for the CGU_STAT.PLLEN bit =0.
3. Enable the DPM event interrupt to wake up the desired core, directing exit from idle after exit from deep sleep mode.
4. Set the polarity of wake-up sources as needed with the DPM_WAKE_POL.WSn bits.
5. Enable the wake-up sources as needed with the DPM_WAKE_EN.WSn bits.
6. Set the DPM_CTL.DEEPSLEEP bit.
7. Clear all pending core transactions (for example, on Blackfin processors, use an SSYNC instruction), DMA transactions, and interrupts.
8. Place all processor cores in idle state.

RESULT:

The processor is now in deep sleep mode. To wake the processor, assert any of the enabled wake-up sources.

Configuring Hibernate Mode

The hibernate mode permits the VDD_INT supply to be shut off to reduce power dissipation.

PREREQUISITE:

The hibernate mode can be entered from any DPM state in which software can be run. Reading the DPM_STAT.CURMODE reveals the current power mode. Make sure to save any data that needs to be preserved to the DPM_RESTOREn registers or to external memory before placing the processor into hibernate mode.

The following steps should be performed in code running from L1 memory on Blackfin processors.

1. Set the polarity of wake-up sources as needed with the DPM_WAKE_POL.WSn bits.
2. Enable the wake-up sources as needed with the DPM_WAKE_EN.WSn bits.
3. Write to the DPM_HIB_DIS register if any blocks not on the VDD_INT domain need to be disabled.

   ADDITIONAL INFORMATION: The DPM_HIB_DIS register may not exist on your processor if there are no blocks on domains other than VDD_INT that are capable of being disabled during hibernate.
4. Write the power good count value to the DPM_PGCNTR.CNT field.
5. Clear all pending core transactions (this is done with an SSYNC instruction in Blackfin), DMA transactions, and interrupts.
6. Set the DPM_CTL.HIBERNATE bit

RESULT:

The DPM is now configured to hibernate mode.

ADSP-BF60x Wake-Up Sources

The table shows the hibernate mode and deep sleep mode wake-up sources for the ADSP-BF60x processor.

The first column shows which wake-up source bit (*WSn*) is used in the DPM_WAKE_EN, DPM_WAKE_POL register and the DPM_WAKE_STAT register. The *Assigned Source* column shows which peripheral or pin source is assigned to the *WSn* bit. Peripherals in parentheses indicate that the source can either be used as a general-purpose I/O wake-up or used as the specific peripheral wake-up listed. The *Deep Sleep Mode* column indicates whether or not the wake-up source can wake the processor from deep sleep mode. The *Hibernate Mode* column indicates whether or not the wake-up source can wake the processor from hibernate mode.

Table 5-5: ADSP-BF60x HIBERNATE and DEEP SLEEP Wake-up Sources

DPM_WAKE_EN.WSn, DPM_WAKE_POL.WSn, and DPM_WAKE_STAT.WSn bits	Assigned Source	Deep Sleep Mode?	Hibernate Mode?
WS0	PA_15	Yes	Yes
WS1	PB_15	Yes	Yes
WS2	PC_15	Yes	Yes
WS3	PD_06	Yes	Yes
WS4	PE_12	Yes	Yes
WS5	PG_04 (CAN0_RX)	Yes	Yes

Table 5-5: ADSP-BF60x HIBERNATE and DEEP SLEEP Wake-up Sources (Continued)

DPM_WAKE_EN.WSn, DPM_WAKE_POL.WSn, and DPM_WAKE_STAT.WSn bits	Assigned Source	Deep Sleep Mode?	Hibernate Mode?
WS6	PG_13	Yes	Yes
WS7	USB (DPM_WAKE_POL.WS7 must =1 for USB)	No	Yes
WS30:WS8	RESERVED	NA	NA

ADSP-BF60x Clock Buffer Disable Bit Assignments

The table shows the clock buffers that may be disabled on the ADSP-BF60x processor.

The first column shows which system clock buffer disable bits (DPM_SCBF_DIS.SCBFn). The *Assigned Clock* column shows which clock buffer is assigned to each DPM_SCBF_DIS.SCBFn bit.

Table 5-6: ADSP-BF60x Bit Assignments for DPM_SCBF_DIS.SCBFn

DPM_SCBF_DIS.SCBFn Bits	Assigned Clock
SCBF0	SCLK0
SCBF1	SCLK1
SCBF2	DCLK
SCBF3	OCLK

ADSP-BF60x Hibernate Disable Bit Assignments

The table shows the functional units that may be disabled during hibernate on the ADSP-BF60x processors.

The first column shows which hibernate disable bits (DPM_HIB_DIS.HDn). The *Functional Unit* column shows which functional unit is assigned to each DPM_HIB_DIS.HDn bit, permitting disable on entering hibernate mode.

Table 5-7: ADSP-BF60x Bit Assignments for DPM_HIB_DIS.HDn

DPM_HIB_DIS.HDn Bits	Functional Unit
HD0	System crystal oscillator
HD7:HD1	Reserved

ADSP-BF60x DPM Register Descriptions

Dynamic Power Management (DPM) contains the following registers.

Table 5-8: ADSP-BF60x DPM Register List

Name	Description
DPM_CTL	Control Register
DPM_STAT	Status Register
DPM_CCBF_DIS	Core Clock Buffer Disable Register
DPM_CCBF_EN	Core Clock Buffer Enable Register
DPM_CCBF_STAT	Core Clock Buffer Status Register
DPM_CCBF_STAT_STKY	Core Clock Buffer Status Sticky Register
DPM_SCBF_DIS	System Clock Buffer Disable Register
DPM_WAKE_EN	Wakeup Enable Register
DPM_WAKE_POL	Wakeup Polarity Register
DPM_WAKE_STAT	Wakeup Status Register
DPM_HIB_DIS	Hibernate Disable Register
DPM_PGCNTR	Power Good Counter Register
DPM_RESTOREn	Restore Registers

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.

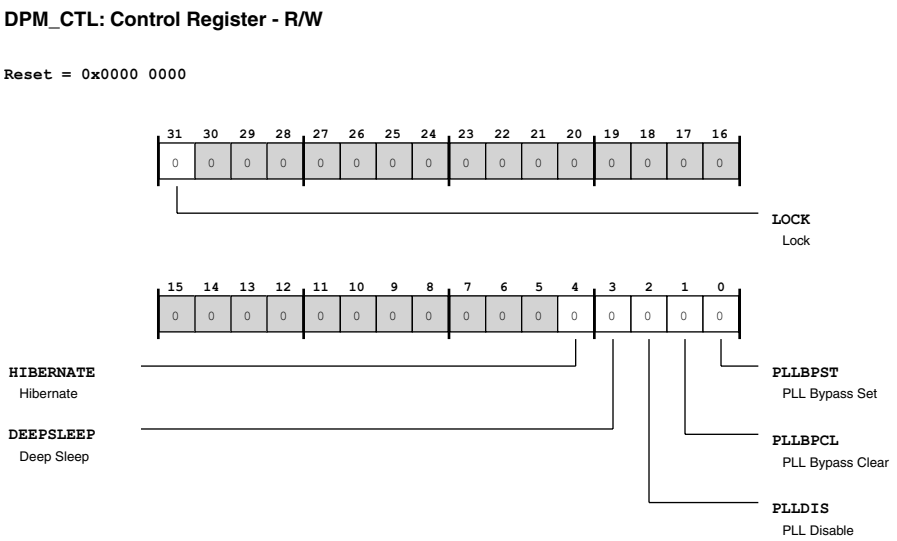


Figure 5-2: DPM_CTL Register Diagram

Table 5-9: DPM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the DPM_CTL.LOCK bit is set, the DPM_CTL register is read only (locked).	
		0	Unlock
		1	Lock
4 (R0/W1A)	HIBERNATE	Hibernate. The DPM_CTL.HIBERNATE bit puts the DPM into hibernate mode. The DPM stays in this mode until a wakeup event or reset occurs. For more information about DPM modes, see the operating modes.	
		0	No Action
		1	Hibernate
3 (R0/W1A)	DEEPSLEEP	Deep Sleep. The DPM_CTL.DEEPSLEEP bit puts the DPM into deep sleep mode. The DPM stays in this mode until a wakeup event occurs. For more information about DPM modes, see the functional description.	
		0	No Action
		1	Deep Sleep

Table 5-9: DPM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	PLLDIS	PLL Disable. While the DPM is in active mode, it is possible to disable the PLL with the DPM_CTL.PLLDIS bit, keeping the DPM active and running with lower power consumption. For more information about DPM modes, see the operating modes.	
		0	Enable
		1	Disable
1 (R0/W1A)	PLLBPCL	PLL Bypass Clear. While the DPM is in active mode, it is possible to disable the PLL bypass with the DPM_CTL.PLLBPCL bit, transitioning to DPM to full-on mode. For more information about DPM modes, see the operating modes.	
		0	No action
		1	Disable PLL Bypass
0 (R0/W1A)	PLLBPST	PLL Bypass Set. While the DPM is in full on mode, it is possible to enable the PLL bypass with the DPM_CTL.PLLBPST bit, transitioning the DPM to active mode. For more information about DPM modes, see the operating modes.	
		0	No action
		1	Enable PLL Bypass

Status Register

The DPM_STAT register indicates error status for DPM operations and indicates the current and previous DPM modes.

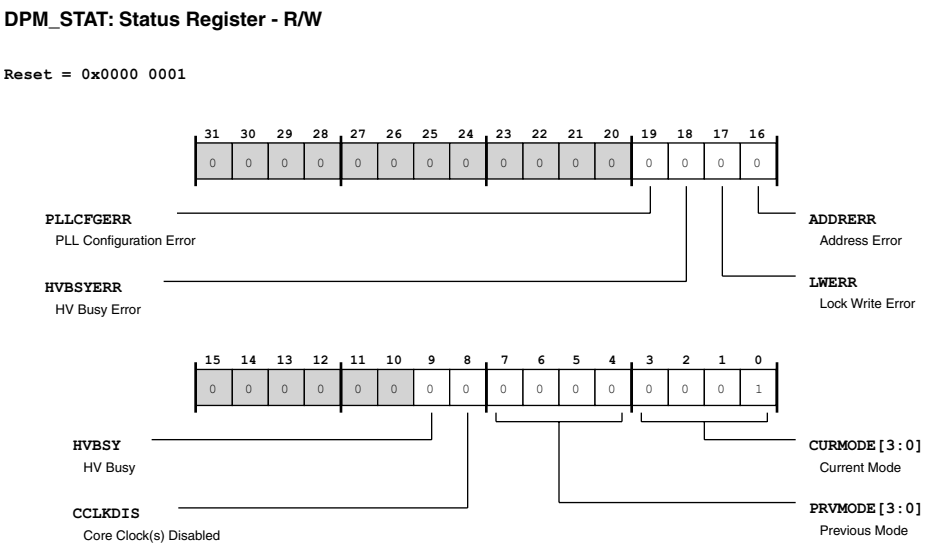


Figure 5-3: DPM_STAT Register Diagram

Table 5-10: DPM_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W1C)	PLLCFGERR	PLL Configuration Error. The DPM_STAT.PLLCFGERR bit indicates that either the DPM_CTL.PLLBPST bit and DPM_CTL.PLLBPCL bit have been set (=1) simultaneously or the DPM_CTL.PLLBPST has been set while the DPM is in full on mode or while the DPM is entering full on mode (DPM_CTL.PLLBPCL =1).	
		0	No Status
		1	PLL Configuration Error
18 (R/W1C)	HVBSYERR	HV Busy Error. The DPM_STAT.HVBSYERR bit indicates that a read access of DPM_WAKE_STAT or DPM_RESTOREn occurred during restore of DPM LV from DPM HV	
		0	No Status
		1	HV Busy Error
17 (R/W1C)	LWERR	Lock Write Error. The DPM_STAT.LWERR bit indicates an attempt to write to a locked DPM register while the global lock bit is set (SPU_CTL_GLCK bit =1).	
		0	No Status
		1	Lock Write Error

Table 5-10: DPM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W1C)	ADDRERR	Address Error. The DPM_STAT.ADDRERR bit indicates either an attempted read or write access to an address in the DPM MMR space that are not implemented addresses, an attempted write access to a DPM read only register, or a read or write access to a non aligned address in the DPM MMR space.	
		0	No Status
		1	Address Error
9 (R/NW)	HVBSY	HV Busy. The DPM_STAT.HVBSY bit indicates whether the DPM HV unit is ready (has completed all transactions) or is busy (transfers from HV have not yet completed).	
		0	Ready
		1	Busy
8 (R/NW)	CCLKDIS	Core Clock(s) Disabled. The DPM_STAT.CCLKDIS bit indicates whether or not one or more of the core clocks is disabled.	
		0	No Status
		1	Clock (1 or more) Disabled
7:4 (R/NW)	PRVMODE	Previous Mode. The DPM_STAT.PRVMODE bits indicate the previous DPM mode of operation. All values not shown are reserved.	
		0	Reset
		1	Full-On
		2	Active
		3	Active with PLL disabled
		4	Deep Sleep
		5	Hibernate
3:0 (R/NW)	CURMODE	Current Mode. The DPM_STAT.CURMODE bits indicate the current DPM mode of operation. All values not shown are reserved.	
		1	Full-On
		2	Active
		3	Active with PLL disabled

Core Clock Buffer Disable Register

The DPM_CCBF_DIS register controls the core n clock buffers. The number of clocks varies with the processor design, with bit 0 corresponding to CCLK0, bit 1 corresponding to CCLK1, and so on. This register includes a write protection lock.

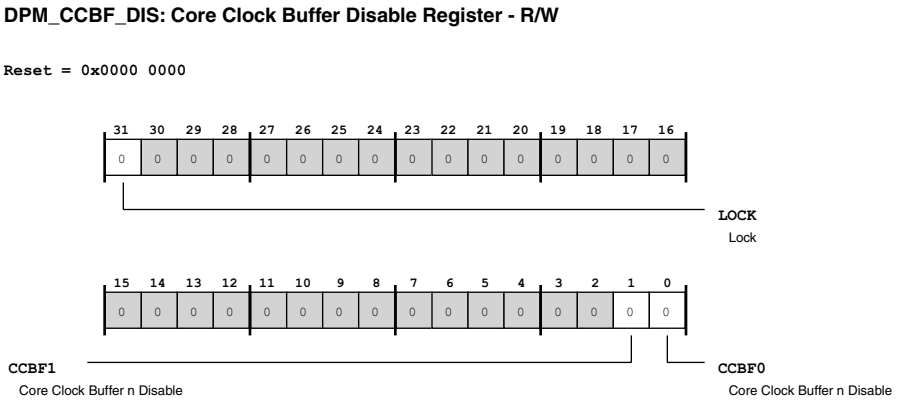


Figure 5-4: DPM_CCBF_DIS Register Diagram

Table 5-11: DPM_CCBF_DIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock.	
		0	Unlock
		1	Lock
1:0 (R0/W1A)	CCBFn	Core Clock Buffer n Disable. The DPM_CCBF_DIS.CCBFn bits provide a core clock buffer disable for each core on the processor with bit 0 corresponding to CCLK0, bit 1 corresponding to CCLK1, and so on.	
		0	No Action
		1	Disable Buffer

Core Clock Buffer Enable Register

The DPM_CCBF_EN register controls the core n clock buffers. The number of clocks varies with the processor design, with bit 0 corresponding to CCLK0, bit 1 corresponding to CCLK1, and so on. This register includes a write protection lock.

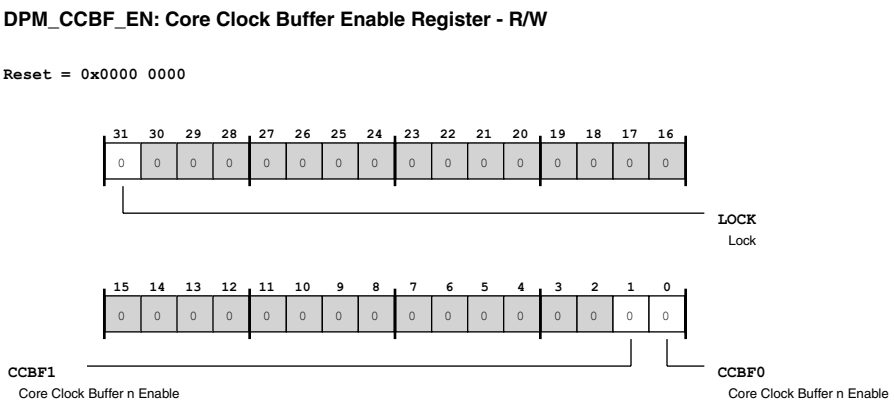


Figure 5-5: DPM_CCBF_EN Register Diagram

Table 5-12: DPM_CCBF_EN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the DPM_CCBF_EN.LOCK bit is set, the DPM_CCBF_EN register is read only (locked).	
		0	Unlock
		1	Lock
1:0 (R0/W1A)	CCBFn	Core Clock Buffer n Enable. The DPM_CCBF_EN.CCBFn bits provide a core clock buffer enable for each core on the processor with bit 0 corresponding to CCLK0, bit 1 corresponding to CCLK1, and so on.	
		0	No Action
		1	Enable Buffer

Core Clock Buffer Status Register

The DPM_CCBF_STAT register indicates core clock buffer enable or disable status for each core on the processor, with bit 0 corresponding to CCLK0, bit 1 corresponding to CCLK1, and so on.

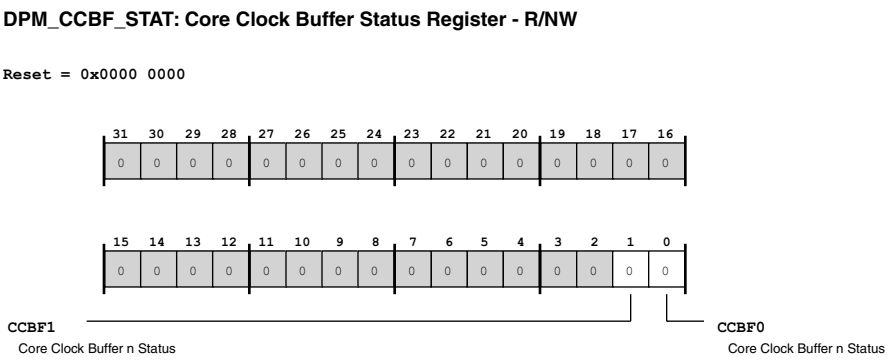


Figure 5-6: DPM_CCBF_STAT Register Diagram

Table 5-13: DPM_CCBF_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/NW)	CCBFn	Core Clock Buffer n Status. The DPM_CCBF_STAT.CCBFn bits indicates core clock buffer enabled or disabled status for each core on the processor with bit 0 corresponding to CCLK0, bit 1 corresponding to CCLK1, and so on.	
		0	Buffer Enabled
		1	Buffer Disabled

Core Clock Buffer Status Sticky Register

The DPM_CCBF_STAT_STKY register indicates core n clock buffer enable or disable sticky status for each core on the processor, with bit 0 corresponding to CCLK0, bit 1 corresponding to CCLK1, and so on.

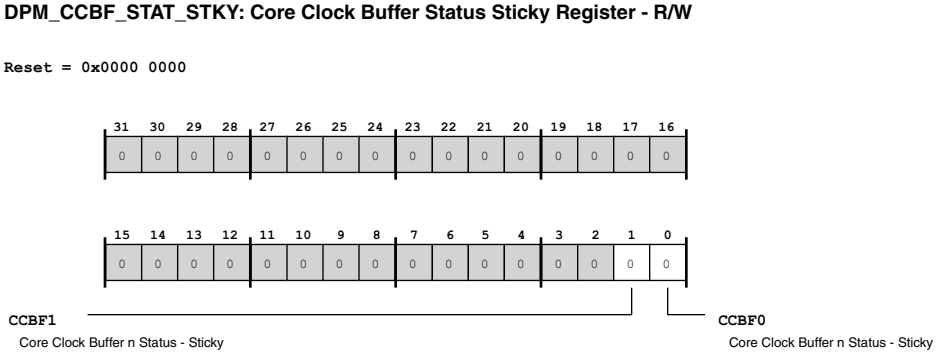


Figure 5-7: DPM_CCBF_STAT_STKY Register Diagram

Table 5-14: DPM_CCBF_STAT_STKY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W1C)	CCBFn	Core Clock Buffer n Status - Sticky. The DPM_CCBF_STAT_STKY.CCBFn bits indicates core clock buffer enabled or disabled sticky status for each core on the processor with bit 0 corresponding to CCLK0, bit 1 corresponding to CCLK1, and so on. The sticky status shows that the status was set since the last time the bit was cleared with a W1A or reset.	
		0	Buffer Enabled - Sticky
		1	Buffer Disabled - Sticky

System Clock Buffer Disable Register

The DPM_SCBF_DIS register controls the system n clock buffers. The number of clocks varies with the processor design. See the Clock Buffer Disable Bit Assignments topic for the bit assignments of this processor.

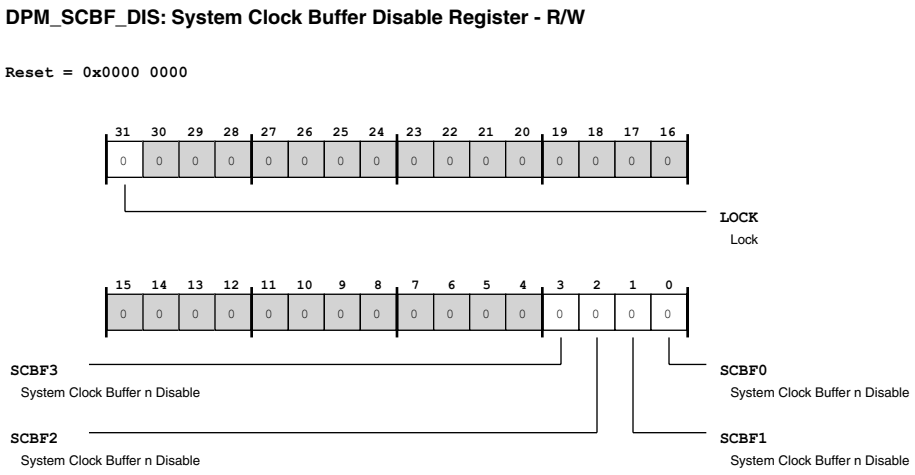


Figure 5-8: DPM_SCBF_DIS Register Diagram

Table 5-15: DPM_SCBF_DIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the DPM_SCBF_DIS.LOCK bit is set, the DPM_SCBF_DIS register is read only (locked).	
		0	Unlock
		1	Lock
3:0 (R/W)	SCBFn	System Clock Buffer n Disable. The DPM_SCBF_DIS.SCBFn bits provide a system clock buffer enable for each system clock domain on the processor. See the Clock Buffer Disable Bit Assignments topic for the bit assignments of this processor.	
		0	Enable Buffer
		1	Disable Buffer

Wakeup Enable Register

The DPM_WAKE_EN register enables the wakeup event sources for exiting deep sleep mode. The number of wakeup sources varies with the processor design, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wakeup source 1, and so on. This register includes a write protection lock. For information about wakeup source assignments, see the DPM Wakeup Sources topic.

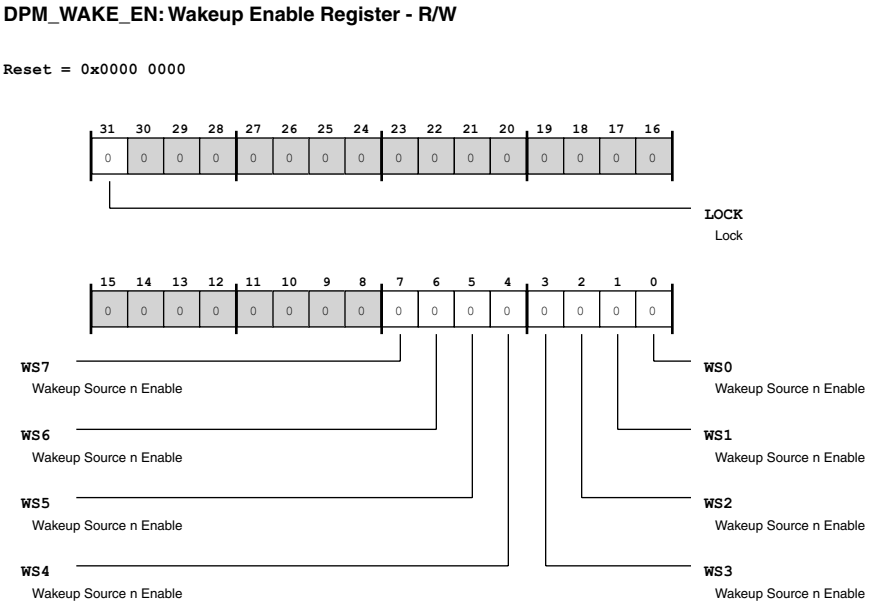


Figure 5-9: DPM_WAKE_EN Register Diagram

Table 5-16: DPM_WAKE_EN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the DPM_WAKE_EN.LOCK bit is set, the DPM_WAKE_EN register is read only (locked).	
		0	Unlock
		1	Lock
7:0 (R/W)	WSn	Wakeup Source n Enable. The DPM_WAKE_EN.WSn bits enable wakeup sources for exiting deep sleep mode, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wakeup source 1, and so on. For information about wakeup source assignments, see the DPM Wakeup Sources topic.	
		0	Disable Wakeup Source
		1	Enable Wakeup Source

Wakeup Polarity Register

The DPM_WAKE_POL register select polarity (active high or low) of the wakeup event sources for exiting deep sleep mode. The number of wakeup sources varies with the processor design, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wakeup source 1, and so on. This register includes a write protection lock. For information about wakeup source assignments, see the DPM Wakeup Sources topic.

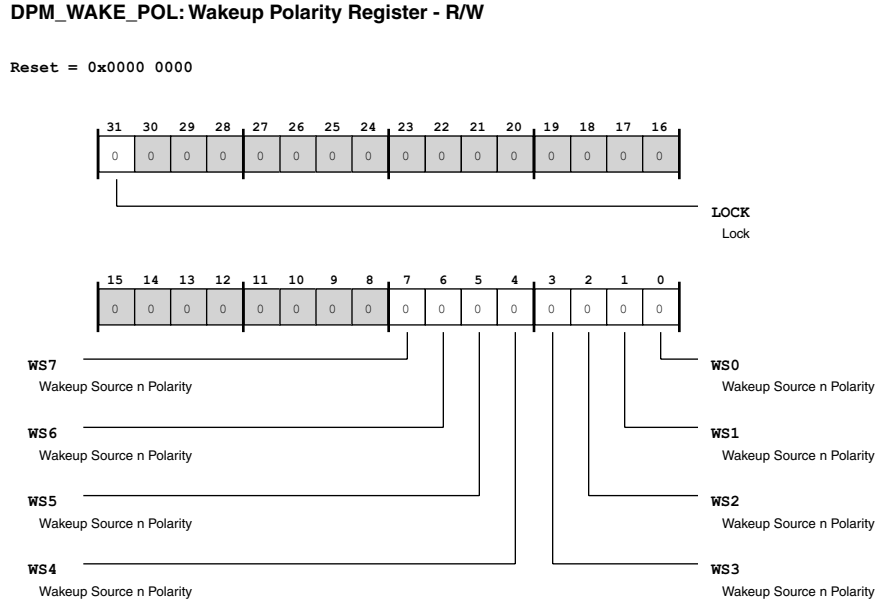


Figure 5-10: DPM_WAKE_POL Register Diagram

Table 5-17: DPM_WAKE_POL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the DPM_WAKE_POL.LOCK bit is set, the DPM_WAKE_POL register is read only (locked).	
		0	Unlock
		1	Lock
7:0 (R/W)	WSn	Wakeup Source n Polarity. The DPM_WAKE_POL.WSn bits select polarity (active high or low) of wakeup sources for exiting deep sleep mode, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wakeup source 1, and so on. For information about wakeup source assignments, see the DPM Wakeup Sources topic.	
		0	Low Active Wakeup
		1	High Active Wakeup

Wakeup Status Register

The DPM_WAKE_STAT register indicates the enabled and active status of the wakeup event sources for exiting deep sleep mode. The number of wakeup sources varies with the processor design, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wakeup source 1, and so on. For information about wakeup source assignments, see the DPM Wakeup Sources topic.

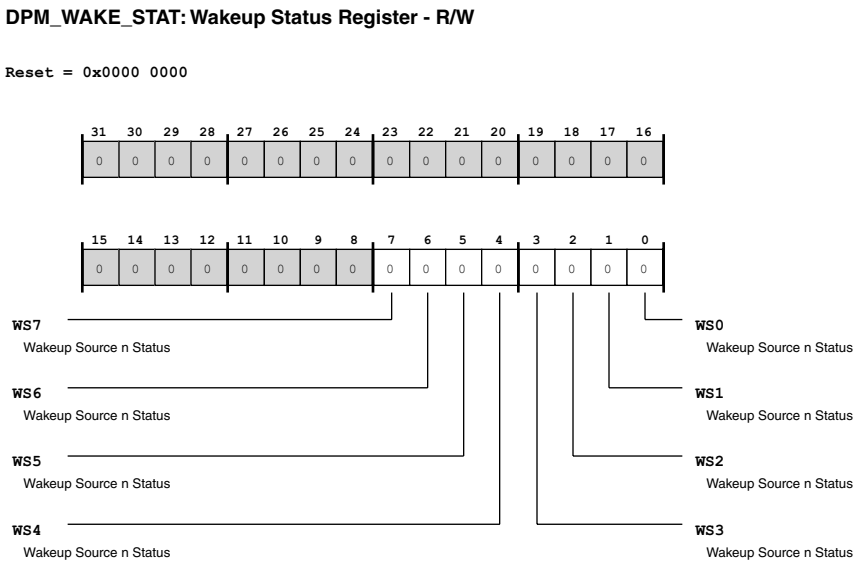


Figure 5-11: DPM_WAKE_STAT Register Diagram

Table 5-18: DPM_WAKE_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:0 (R/W1C)	WSn	Wakeup Source n Status. The DPM_WAKE_STAT.WSn bits indicate the enabled and active status of wakeup sources for exiting deep sleep mode, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wakeup source 1, and so on. For information about wakeup source assignments, see the DPM Wakeup Sources topic.	
		0	No Status
		1	Enabled and Active

Hibernate Disable Register

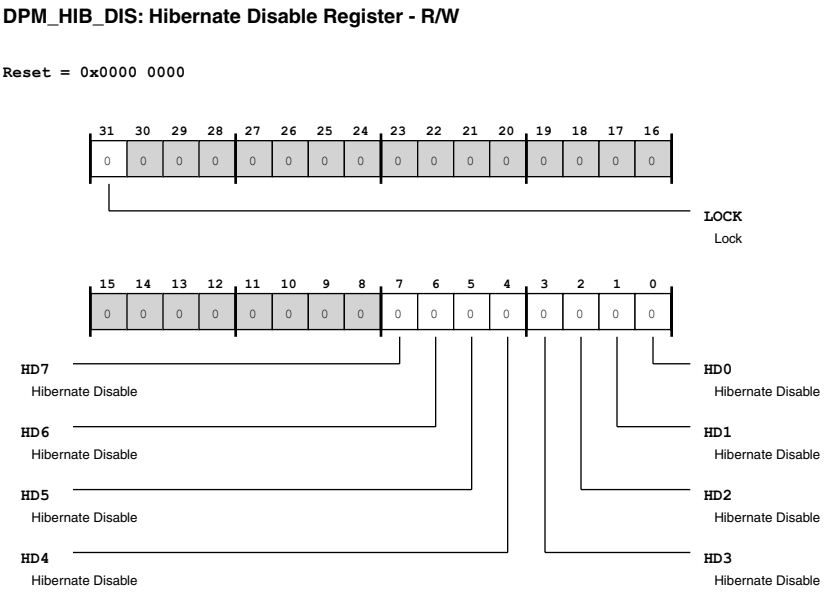


Figure 5-12: DPM_HIB_DIS Register Diagram

Table 5-19: DPM_HIB_DIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the DPM_HIB_DIS.LOCK bit is set, the DPM_HIB_DIS register is read only (locked).	
		0	Unlock
		1	Lock
7:0 (R/W)	HDn	Hibernate Disable. The DPM_HIB_DIS register controls which functional units on the V_{DD_EXT} domain are disabled during hibernate. For the bit assignments of this processor, see the DPM Hibernate Disable Bit Assignments topic.	
		0	Enable External Wakeup Source
		1	Disable Functional Unit During Hibernate

Power Good Counter Register

The DPM_PGCNTR register selects the count of CLKIN cycles the DPM waits for the V_{DDINT} power supply to reach the specified value. This register includes a write protection lock.

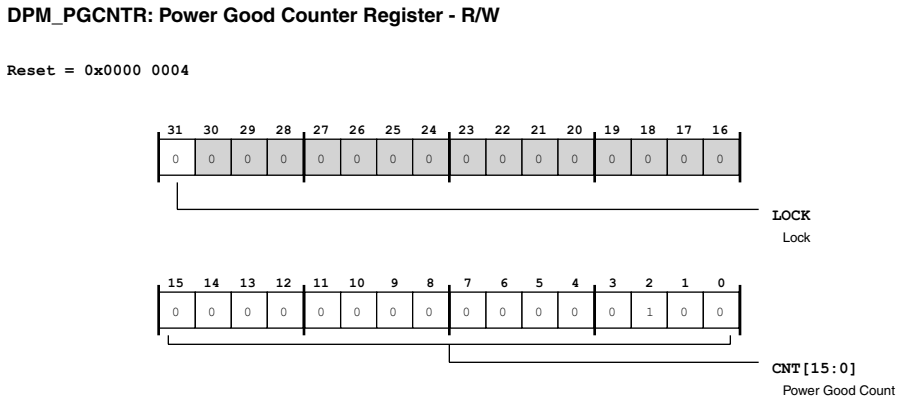


Figure 5-13: DPM_PGCNTR Register Diagram

Table 5-20: DPM_PGCNTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the DPM_PGCNTR.LOCK bit is set, the DPM_PGCNTR register is read only (locked).	
		0	Unlock
		1	Lock
15:0 (R/W)	CNT	Power Good Count. The DPM_PGCNTR.CNT bits select the count that the DPM uses to determine that the V_{DDINT} supply has reached the specified value. All values other than those shown are reserved.	
		4	128 cycles
		128	4K cycles
		256	8K cycles
		512	16K cycles
		1024	32K cycles
		2048	64K cycles
		4096	128K cycles
		8192	256K cycles
		16384	512K cycles
		32768	1M cycles

Restore Registers

The DPM_RESTOREn registers are general-purpose registers that the DPM preserves during hibernate mode and restores on exit from hibernate. The usage of these registers is application specific and does not affect DPM operations. For more information about using the DPM_RESTOREn registers, see the programming model.

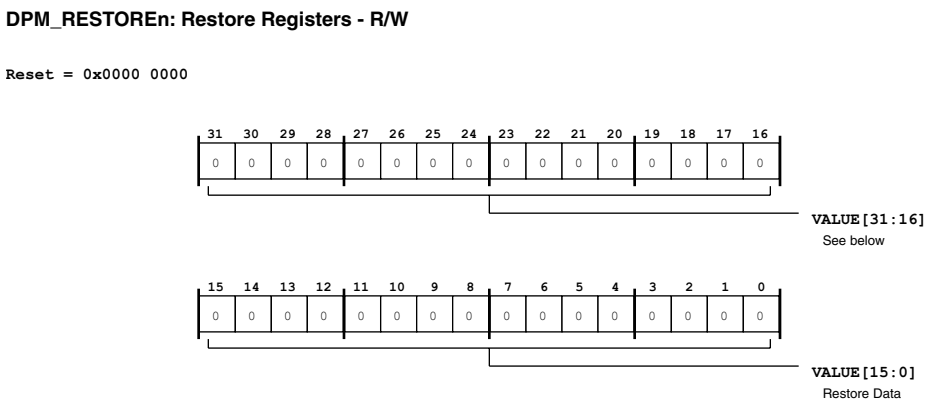


Figure 5-14: DPM_RESTOREn Register Diagram

Table 5-21: DPM_RESTOREn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Restore Data. The DPM_RESTOREn.VALUE data is stored during hibernate mode and restored by the DPM on exit from hibernate.

6 Core Timer (TMR)

Each processor core has its own dedicated timer. The core timer is clocked by the internal processor clock and is typically used as a system tick clock for generating periodic operating system interrupts.

NOTE: The core timer stops counting when there is an emulation event. The emulation event is controlled by the SDU (System Debug Unit). For more information, see the SDU chapter.

TMR Features

The core timer is a programmable 32-bit interval timer which can generate periodic interrupts. Unlike other peripherals, the core timer resides inside the Blackfin core and runs at the core clock (CCLK) rate. Core timer features include:

- 32-bit timer with 8-bit prescaler
- Operates at core clock (*CCLK*) rate
- Dedicated high-priority interrupt channel
- Single-shot or continuous operation

TMR Functional Description

The TMR (core timer) is a programmable 32-bit interval timer in each processor core. The following sections describe the TMR features:

- *ADSP-BF60x TMR Register List*
- *TMR Block Diagram*

ADSP-BF60x TMR Register List

The core timer (TMR) is a programmable 32-bit interval timer, which can generate periodic interrupts. Unlike other peripherals, the core timer resides inside the processor core and runs at the core clock (CCLK) rate. A set of registers govern TMR operations. For more information on TMR functionality, see the TMR register descriptions.

Table 6-1: ADSP-BF60x TMR Register List

Name	Description
TCNTL	Timer Control Register
TPERIOD	Timer Period Register
TSCALE	Timer Scale Register
TCOUNT	Timer Count Register

TMR Block Diagram

The following figure shows the core timer block diagram.

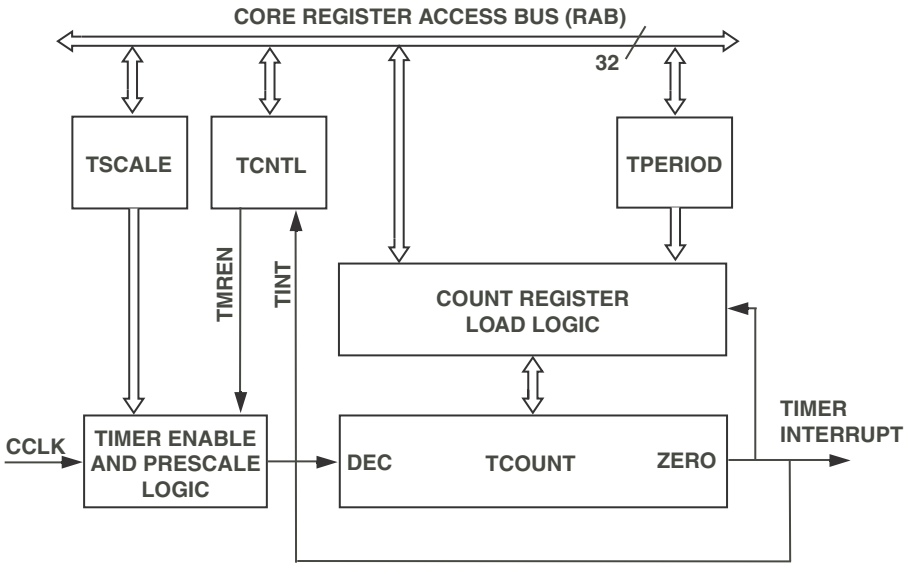


Figure 6-1: Core Timer Block Diagram

External Interfaces

The core timer does not directly interact with any pins of the chip.

Internal Interfaces

The core timer is accessed through the 32-bit register access bus. The module is clocked by the core clock *CCLK*. The timer's dedicated interrupt request is a higher priority than requests from all other peripherals.

TMR Operation

The software should initialize the TCOUNT register before the timer is enabled. The TCOUNT register can be written directly, but writes to the TPERIOD register are also passed through to TCOUNT.

When the timer is enabled by setting the TCNTL.EN bit, the TCOUNT register is decremented once every time the prescaler register (TSCALE) expires, that is, every TSCALE + 1 number of *CCLK* clock cycles. When the value of the TCOUNT register reaches 0, an interrupt is generated and the TCNTL.INT bit is set.

If the TCNTL.AUTORLD bit is set, then the TCOUNT register is reloaded with the contents of the TPERIOD register and the count begins again. If the TCNTL.AUTORLD bit is not set, the timer stops operation.

The core timer can be put into low power mode by clearing the TCNTL.PWR bit. Before using the timer, set the TCNTL.PWR bit. This restores clocks to the timer unit. When TCNTL.PWR is set, the core timer may then be enabled by setting the TCNTL.EN bit.

NOTE: Hardware behavior is undefined if TCNTL.EN is set when TCNTL.PWR=0.

Interrupt Processing

The timer's dedicated interrupt request is a higher priority than requests from all other peripherals. The request goes directly to the core event controller (CEC) and does not pass through the system event controller (SEC). Therefore, the interrupt processing is also completely in the *CCLK* domain.

NOTE: The core timer interrupt request is edge-sensitive and cleared by hardware automatically as soon as the interrupt is serviced.

The TCNTL.INT bit indicates that an interrupt has been generated and programs need to write a 0 (not W1C) to clear it (the write is optional). The core time module doesn't provide any further interrupt enable bit. When the timer is enabled, interrupts can be masked in the CEC controller.

ADSP-BF60x TMR Register Descriptions

Core Timer (TMR) contains the following registers.

Table 6-2: ADSP-BF60x TMR Register List

Name	Description
TCNTL	Timer Control Register
TPERIOD	Timer Period Register
TSCALE	Timer Scale Register

Table 6-2: ADSP-BF60x TMR Register List (Continued)

Name	Description
TCOUNT	Timer Count Register

Timer Control Register

The TCNTL register functions as the TMR control and status register.

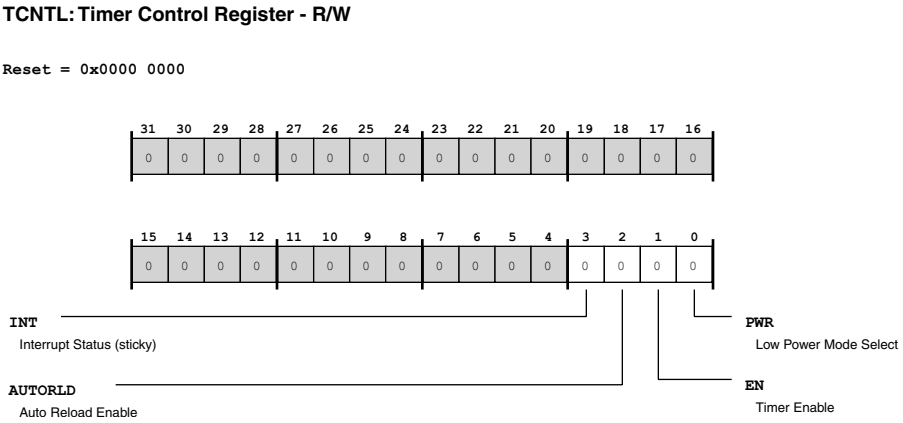


Figure 6-2: TCNTL Register Diagram

Table 6-3: TCNTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	INT	Interrupt Status (sticky). The TCNTL.INT bit indicates the status of the timer generated interrupt, which is generated when the TCOUNT value decrements to 0. This bit is sticky, remaining set until cleared.	
		0	No Interrupt Pending
		1	Pending Interrupt
2 (R/W)	AUTORLD	Auto Reload Enable. The TCNTL.AUTORLD bit enables the TMR to automatically reload the TCOUNT register from the TPERIOD register when the count expires. If disabled, the timer stops operation when the count expires.	
		0	Disable Auto Reload
		1	Enable Auto Reload

Table 6-3: TCNTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	EN	Timer Enable. The TCNTL.EN bit enables TMR operation, starting the count at the value in the TCOUNT register and decrementing this value each time the prescaler (TSCALE) time expires.	
		0	Disable Timer
		1	Enable Timer
0 (R/W)	PWR	Low Power Mode Select. The TCNTL.PWR bit selects active mode or low power mode for TMR operation. When in low power mode, clocks to the TMR are disabled, Before enabling the timer (TCNTL.EN =1), put the timer in active mode (restoring clocks to the TMR).	
		0	Low Power Mode
		1	Active Mode

Timer Period Register

The TPERIOD register holds the timer period. When the processor writes the TPERIOD register, the TMR automatically writes the value to the TCOUNT register also. When count auto reload is enabled (TCNTL.AUTORLD =1), the TMR automatically reloads the TCOUNT register with the value in TPERIOD when the counter expires. Note that writes to TPERIOD are ignored when the timer is running.

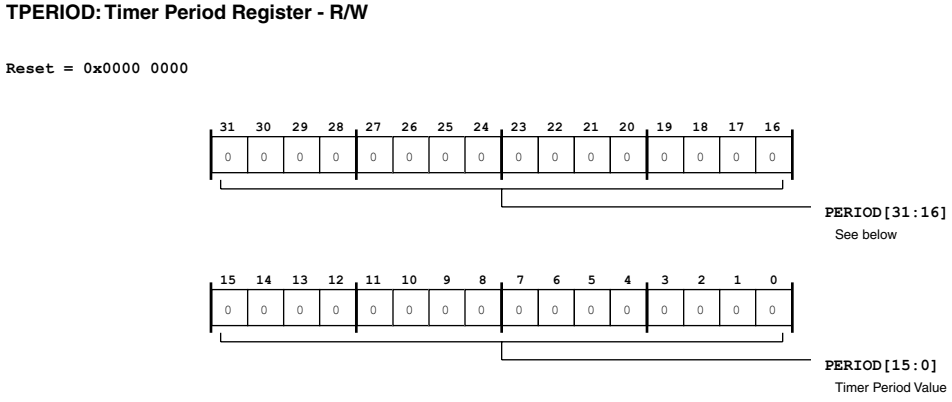


Figure 6-3: TPERIOD Register Diagram

Table 6-4: TPERIOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	PERIOD	Timer Period Value. The TPERIOD.PERIOD bits hold the timer period value.

Timer Scale Register

The TSCALE register stores the scaling value that is one core clock (CCLK) cycle less than the number of cycles between each decrement of the timer count (TCOUNT). For example, if the value in the TSCALE register is 0, the TCOUNT register decrements once for every CCLK cycle. If the TSCALE value is 1, the TCOUNT decrements once every two CCLK cycles.

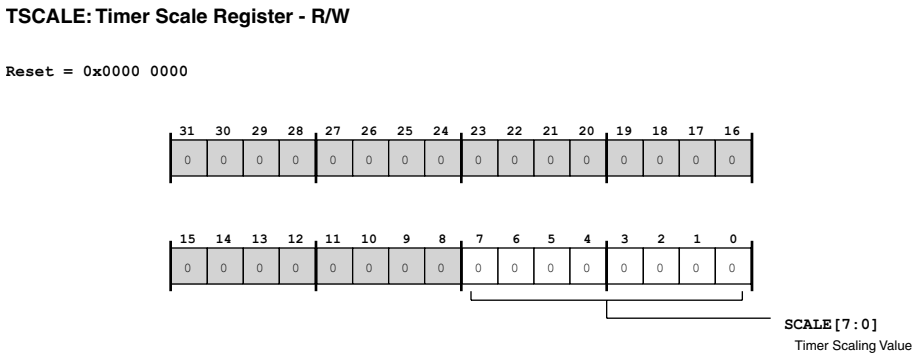


Figure 6-4: TSCALE Register Diagram

Table 6-5: TSCALE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	SCALE	Timer Scaling Value. The TSCALE.SCALE bits hold a scaling factor, relating CCLK cycles to the rate that the count decrements.

Timer Count Register

The TCOUNT register holds the current count for the TMR. Typically, this count is initially load with a write to the TPERIOD register, because the TMR automatically copies the value written to the TPERIOD register and writes the value to the TCOUNT register. The TCOUNT register can be written directly. When the timer is running, the TCOUNT value decrements once every TSCALE + 1 core clock (CCLK) cycles. When the value in TCOUNT reaches 0, the TMR generates an interrupt and sets the TCNTL.INT bit. Note that writes to TCOUNT are ignored when the timer is running.

TCOUNT: Timer Count Register - R/W

Reset = 0x0000 0000

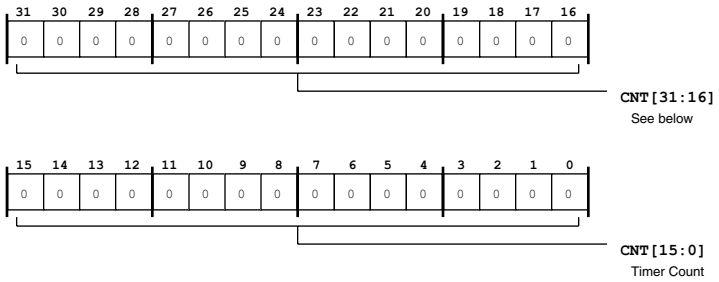


Figure 6-5: TCOUNT Register Diagram

Table 6-6: TCOUNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	CNT	Timer Count. The `TCOUNT.CNT` bits hold the current timer count value.

7 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 as well as the propagation of system events to all connected cores and the system fault interface.

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, time out, 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 completely independent of all others.
- Slave Control Port which provides access to all SEC registers for configuration, status, and interrupt/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.

ADSP-BF60x SEC Register List

The system event controller (SEC) manages the system interrupt and system fault sources. The SEC also provides all system interrupt and fault sources control features, such as enable/disable, prioritization, and active/pending source status. On multi-core processors, the SEC provides connected core(s) and fault management with source pending and active indication. For more information on SEC functionality, see the SEC register descriptions.

Table 7-1: ADSP-BF60x SEC Register List

Name	Description
SEC_CCTLn	SCI Control Register n
SEC_CSTATn	SCI Status Register n
SEC_CPNDn	Core Pending Register n
SEC_CACTn	SCI Active Register n
SEC_CPMSKn	SCI Priority Mask Register n
SEC_CGMSKn	SCI Group Mask Register n
SEC_CPLVLn	SCI Priority Level Register n
SEC_CSIDn	SCI Source ID Register n
SEC_FCTL	Fault Control Register
SEC_FSTAT	Fault Status Register
SEC_FSID	Fault Source ID Register
SEC_FEND	Fault End Register
SEC_FDLY	Fault Delay Register
SEC_FDLY_CUR	Fault Delay Current Register
SEC_FSRDLY	Fault System Reset Delay Register
SEC_FSRDLY_CUR	Fault System Reset Delay Current Register
SEC_FCOPP	Fault COP Period Register
SEC_FCOPP_CUR	Fault COP Period Current Register
SEC_GCTL	Global Control Register
SEC_GSTAT	Global Status Register
SEC_RAISE	Global Raise Register
SEC_END	Global End Register

Table 7-1: ADSP-BF60x SEC Register List (Continued)

Name	Description
SEC_SCTLn	Source Control Register n
SEC_SSTATn	Source Status Register n

ADSP-BF60x Interrupt List

Table 7-2: ADSP-BF60x Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
SEC0 Error	0		LEVEL
CGU0 Event	1		PULSE/EDGE
WDOG0 Expiration	2		LEVEL
WDOG1 Expiration	3		LEVEL
L2CTL0 ECC Error	4		LEVEL
Core 0 Double Fault	6		PULSE/EDGE
Core 1 Double Fault	7		PULSE/EDGE
Core 0 Hardware Error	8		PULSE/EDGE
Core 1 Hardware Error	9		PULSE/EDGE
Core 0 Unhandled NMI or L1 Memory Parity Error	10		PULSE/EDGE
Core 1 Unhandled NMI or L1 Memory Parity Error	11		PULSE/EDGE
TIMER0 Timer 0	12		LEVEL
TIMER0 Timer 1	13		LEVEL
TIMER0 Timer 2	14		LEVEL
TIMER0 Timer 3	15		LEVEL
TIMER0 Timer 4	16		LEVEL
TIMER0 Timer 5	17		LEVEL
TIMER0 Timer 6	18		LEVEL
TIMER0 Timer 7	19		LEVEL
TIMER0 Status	20		LEVEL
PINT0 Pin Interrupt Block	21		LEVEL
PINT1 Pin Interrupt Block	22		LEVEL

Table 7-2: ADSP-BF60x Interrupt List Interrupt List (Continued)

Description	Interrupt ID	DMA Channel	Sensitivity
PINT2 Pin Interrupt Block	23		LEVEL
PINT3 Pin Interrupt Block	24		LEVEL
PINT4 Pin Interrupt Block	25		LEVEL
PINT5 Pin Interrupt Block	26		LEVEL
CNT0 Status	27		LEVEL
PWM0 PWMTMR Group	28		LEVEL
PWM0 Trip	29		LEVEL
PWM1 PWMTMR Group	30		LEVEL
PWM1 Trip	31		LEVEL
TWI0 Data Interrupt	32		LEVEL
TWI1 Data Interrupt	33		LEVEL
Software-driven Interrupt 0	34		PULSE/EDGE
Software-driven Interrupt 1	35		PULSE/EDGE
Software-driven Interrupt 2	36		PULSE/EDGE
Software-driven Interrupt 3	37		PULSE/EDGE
ACM0 Event Miss	38		LEVEL
ACM0 Event Complete	39		LEVEL
CAN0 Receive	40		LEVEL
CAN0 Transmit	41		LEVEL
CAN0 Status	42		LEVEL
SPORT0 Channel A DMA	43	0	LEVEL
SPORT0 Channel A Status	44		LEVEL
SPORT0 Channel B DMA	45	1	LEVEL
SPORT0 Channel B Status	46		LEVEL
SPORT1 Channel A DMA	47	2	LEVEL
SPORT1 Channel A Status	48		LEVEL
SPORT1 Channel B DMA	49	3	LEVEL
SPORT1 Channel B Status	50		LEVEL

Table 7-2: ADSP-BF60x Interrupt List Interrupt List (Continued)

Description	Interrupt ID	DMA Channel	Sensitivity
SPORT2 Channel A DMA	51	4	LEVEL
SPORT2 Channel A Status	52		LEVEL
SPORT2 Channel B DMA	53	5	LEVEL
SPORT2 Channel B Status	54		LEVEL
SPI0 TX DMA Channel	55	6	LEVEL
SPI0 RX DMA Channel	56	7	LEVEL
SPI0 Status	57		LEVEL
SPI1 TX DMA Channel	58	8	LEVEL
SPI1 RX DMA Channel	59	9	LEVEL
SPI1 Status	60		LEVEL
RSI0 DMA Channel	61	10	LEVEL
RSI0 Interrupt 0	62		LEVEL
RSI0 Interrupt 1	63		LEVEL
SDU0 DMA	64	11	LEVEL
Reserved	65		
Reserved	66		
Reserved	67		
EMAC0 Status	68		LEVEL
Reserved	69		
EMAC1 Status	70		LEVEL
Reserved	71		
LP0 DMA Channel	72	13	LEVEL
LP0 Status	73		LEVEL
LP1 DMA Channel	74	14	LEVEL
LP1 Status	75		LEVEL
LP2 DMA Channel	76	15	LEVEL
LP2 Status	77		LEVEL
LP3 DMA Channel	78	16	LEVEL
LP3 Status	79		LEVEL
UART0 Transmit DMA	80	17	LEVEL
UART0 Receive DMA	81	18	LEVEL

Table 7-2: ADSP-BF60x Interrupt List Interrupt List (Continued)

Description	Interrupt ID	DMA Channel	Sensitivity
UART0 Status	82		LEVEL
UART1 Transmit DMA	83	19	LEVEL
UART1 Receive DMA	84	20	LEVEL
UART1 Status	85		LEVEL
Memory DMA Stream 0 Source / CRC0 Input Channel	86	21	LEVEL
Memory DMA Stream 0 Destination / CRC0 Output Channel	87	22	LEVEL
CRC0 Datacount expiration	88		LEVEL
CRC0 Error	89		LEVEL
Memory DMA Stream 1 Source / CRC1 Input Channel	90	23	LEVEL
Memory DMA Stream 1 Destination / CRC1 Output Channel	91	24	LEVEL
CRC1 Datacount expiration	92		LEVEL
CRC1 Error	93		LEVEL
Memory DMA Stream 2 Source Channel	94	25	LEVEL
Memory DMA Stream 2 Destination Channel	95	26	LEVEL
Memory DMA Stream 3 Source Channel	96	27	LEVEL
Memory DMA Stream 3 Destination Channel	97	28	LEVEL
EPPI0 Channel 0 DMA	98	29	LEVEL
EPPI0 Channel 1 DMA	99	30	LEVEL
EPPI0 Status	100		LEVEL
EPPI2 Channel 0 DMA	101	31	LEVEL
EPPI2 Channel 1 DMA	102	32	LEVEL

Table 7-2: ADSP-BF60x Interrupt List Interrupt List (Continued)

Description	Interrupt ID	DMA Channel	Sensitivity
EPPI2 Status	103		LEVEL
EPPI1 Channel 0 DMA	104	33	LEVEL
EPPI1 Channel 1 DMA	105	34	LEVEL
EPPI1 Status	106		LEVEL
PIXC0 Channel 0 DMA	107	35	LEVEL
PIXC0 Channel 1 DMA	108	36	LEVEL
PIXC0 Channel 2 DMA	109	37	LEVEL
PIXC0 Status	110		LEVEL
PVP0 Camera Pipe Data Out B DMA Channel	111	38	LEVEL
PVP0 Camera Pipe Data Out C DMA Channel	112	39	LEVEL
PVP0 Camera Pipe Status Out DMA Channel	113	40	LEVEL
PVP0 Camera Pipe Control In DMA Channel	114	41	LEVEL
PVP0 Status 0	115		LEVEL
PVP0 Memory Pipe Data Out DMA Channel	116	42	LEVEL
PVP0 Memory Pipe Data In DMA Channel	117	43	LEVEL
PVP0 Memory Pipe Status Out DMA Channel	118	44	LEVEL
PVP0 Memory Pipe Control In DMA Channel	119	45	LEVEL
PVP0 Camera Pipe Data Out A DMA Channel	120	46	LEVEL
PVP0 Status 1	121		LEVEL
USB0 Status/FIFO Data Ready	122		LEVEL
USB0 DMA Status/ Transfer Complete	123		LEVEL
TRU0 Interrupt 0	124		PULSE/EDGE
TRU0 Interrupt 1	125		PULSE/EDGE

Table 7-2: ADSP-BF60x Interrupt List Interrupt List (Continued)

Description	Interrupt ID	DMA Channel	Sensitivity
TRU0 Interrupt 2	126		PULSE/EDGE
TRU0 Interrupt 3	127		PULSE/EDGE
DMA Controller Error	128		LEVEL
CGU0 Error	129		LEVEL
Reserved	130		
DPM0 Event	131		LEVEL
Reserved	132		
SWU0 Event	133		LEVEL
SWU1 Event	134		LEVEL
SWU2 Event	135		LEVEL
SWU3 Event	136		LEVEL
SWU4 Event	137		LEVEL
SWU5 Event	138		LEVEL
SWU6 Event	139		LEVEL

ADSP-BF60x SEC Trigger List

Table 7-3: ADSP-BF60x SEC Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
SEC0 Fault	71	PULSE/EDGE

Table 7-4: ADSP-BF60x SEC Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
None		

SEC Definitions

The following definitions are used in describing the event controller.

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 sub-block of the SEC

SCI

SEC Core Interface, core interface sub-block of the SEC

SPR

SEC prioritizer determines the highest priority pending interrupt and the highest priority active interrupt and provides them in the appropriate registers of the SCI for the priority and nesting model of the SCI

SFI

SEC Fault Interface, fault management sub-block of the SEC

SEC Block Diagram

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.

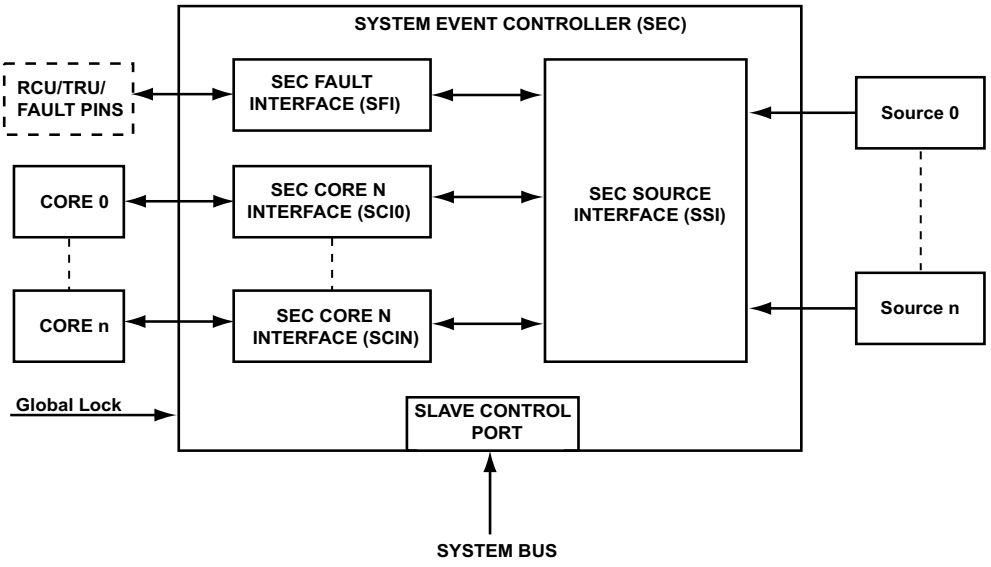


Figure 7-1: System Even Controller Block Diagram

SFI Block Diagram

The SFI manages fault events and associated actions. The fault management support provided in the SEC is intended to help 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, the prescribed action is taken.

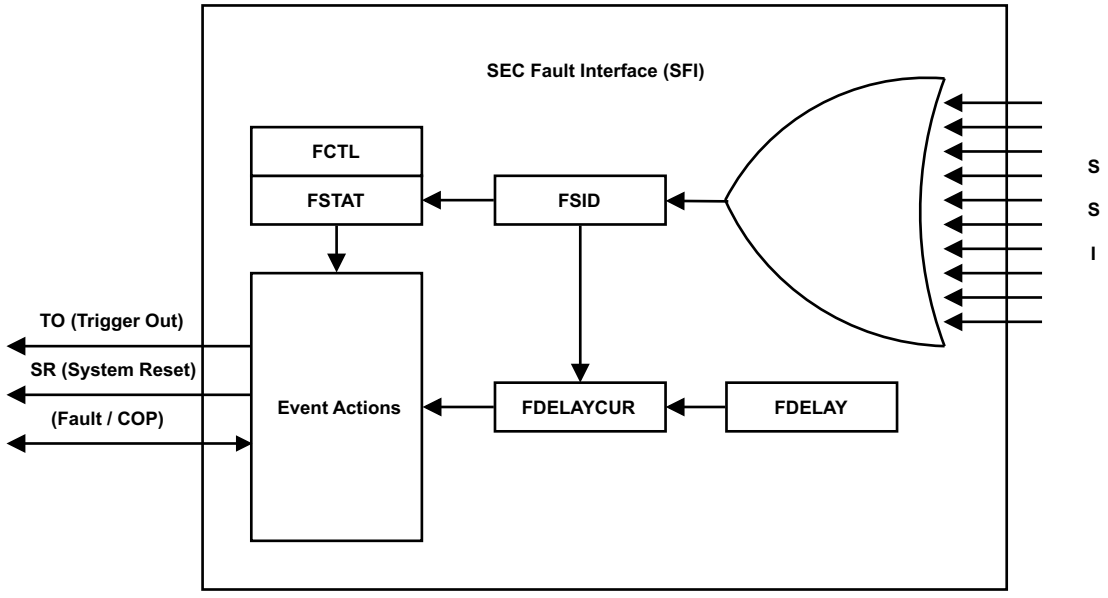


Figure 7-2: SFI Overview Block Diagram

SCI Block Diagram

The SCI manages communication between the corresponding core and the SEC. The 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 will propagate to the core. The SCI maintains the coherency for the system event service model implemented on the connected core.

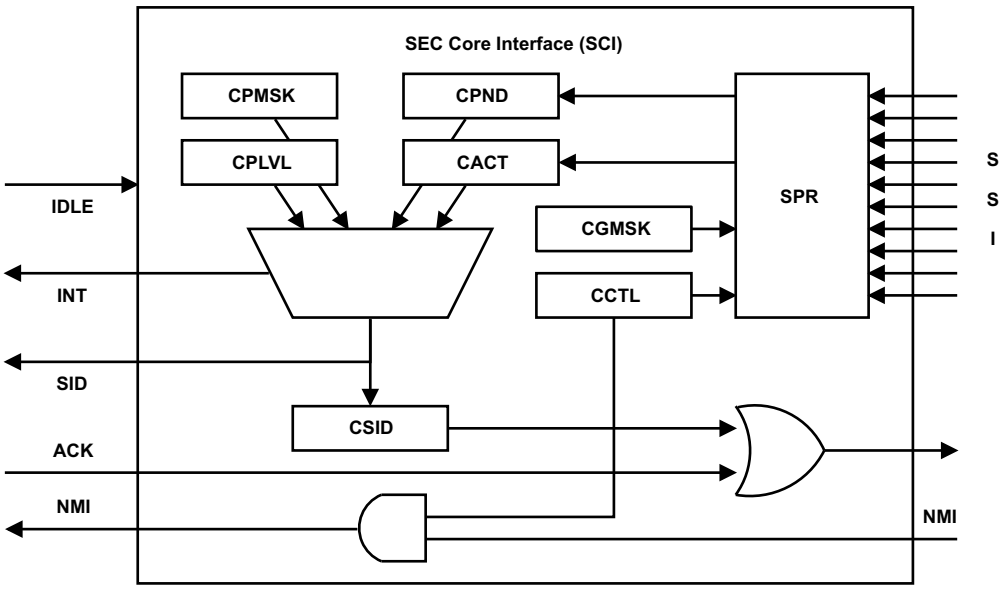


Figure 7-3: SCI Overview Block Diagram

SSI Block Diagram

The SSI manages all of the system event sources. The status of each source is maintained in the corresponding `SEC_SSTATn` register and the control of each source is managed by the corresponding `SEC_SCTLn` register. A pending and enabled event passes its indication and priority to the SCI to which it is assigned for further processing.

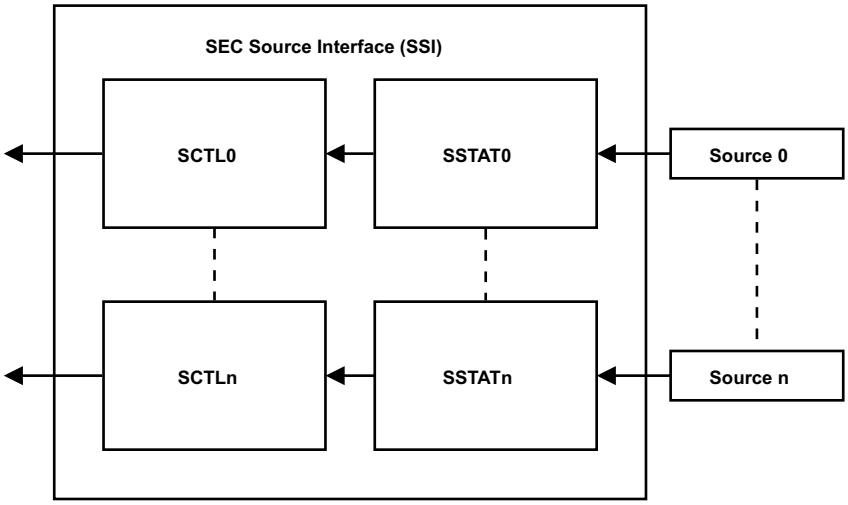


Figure 7-4: SSI Overview Block Diagram

SEC Architectural Concepts

The following sections describe SEC architectural features supporting interrupt acknowledge, priority levels, grouping, flow, and error management.

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 for this to be generated by either of the following.

- A slave port write to the `SEC_CSIDn` register.
- The assertion of an input acknowledge signal (generated by the connected core).

System Interrupt Groups

System sources can be assigned to groups using the `SEC_SCTLn.GRP` bit field. Source groups allow fast context switching for system interrupts at each SCI. The `SEC_CGMSKn` 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_SCTLn.CTG`) bit field setting. The SEC prioritizer determines the highest priority pending system interrupt and the SCI updates the `SEC_CPNDn.SID` and `SEC_CACTn.PRIO` bit field values. The SCI compares the `SEC_CPNDn` register value against the highest priority active source in the `SEC_CACTn` register).

The priority level register (SEC_CPLVLn) determines how many of the MSBs are used in the comparison while the priority mask register (SEC_CPMSKn) and the group mask register (SEC_CGMSKn) determines which pending interrupt sources participate. If the SEC_CPNDn register value is a higher priority (lower value) than that of the SEC_CACTn register from the comparison based on SEC_CPLVLn register, the system interrupt output is asserted and the source ID register (SEC_CSIDn) is updated with the SEC_CPNDn.SID bit field value and forwarded to the connected core.

After the core provides an interrupt acknowledgment (MMR write of the SEC_CSIDn register or the core version of the SEC_CSIDn register), the interrupt source is set active (SEC_SSTATn.ACT==1) until the service is complete by a write of the same value to the SEC_END.SID bit field.

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

1. The SEC compares the SEC_CPNDn register value to the SEC_CACTn register value, if the interrupt in the SEC_CPNDn register is higher priority, continue.
2. The SEC copies the SEC_CPNDn register value to the SEC_CSIDn register and asserts the interrupt signal.
3. The core reads the SEC_CSIDn register (or core version).
4. The core writes to the SEC_CSIDn register (or core version, asserts the acknowledge signal).
5. The SEC de-asserts the interrupt signal and clears the SEC_SSTATn.PND bit and sets the SEC_SSTATn.ACT bit of the source going active.
6. The core writes the SEC_CSIDn of the active interrupt to the SEC_END register.
7. The SEC clears theSEC_SSTATn.ACT bit of the source being ended.

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

1. The SEC compares the SEC_CPNDn (A) register value to the SEC_CACTn register and if the interrupt in the SEC_CPNDn register is a higher priority, continue.
2. The SEC copies SEC_CPNDn (A) register to the SEC_CSIDn register and asserts the interrupt signal.
3. The core reads the SEC_CSIDn (A) register (or core version).
4. The core writes to the SEC_CSIDn register (or core version, asserts the acknowledge signal).
5. The SEC de-asserts the INT signal and clears the SEC_SSTATn.PND bit and sets the SEC_SSTATn.ACT bit of the source (A) going active.
6. The SEC compares the SEC_CPNDn (B) register value to the SEC_CACTn (A) register value. If the SEC_CACTn (A) register value is a higher priority, continue.
7. The SEC copies the SEC_CPNDn (B) register value to SEC_CSIDn register and asserts the interrupt signal.
8. The core reads the SEC_CSIDn (B) register (or core version).
9. The core writes to the SEC_CSIDn register (or core version, asserts the acknowledge signal).

10. The SEC de-asserts the INT signal and clears the SEC_SSTATn.PND bit and sets the SEC_SSTATn.ACT bit of the source (B) going active.
11. The core writes the SEC_CSIDn of the active interrupt (B) to the SEC_END register.
12. The SEC clears the SEC_SSTATn.ACT bit of the source (B) being ended.
13. The core writes the SEC_CSIDn of the active interrupt (A) to the SEC_END register.
14. The SEC clears the SEC_SSTATn.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_SCTLn.PRIO bit field. The SCI evaluates the priority of all pending sources to determine the highest priority pending system interrupt source for forwarding to the attached core. If more than one pending system interrupt source 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 pending system interrupt source.

SEC Error

A SEC error (SEC_GSTAT.ERR) is included as a system 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), core interrupt/event service model, and the SEC. 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, software triggering may be used 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 system event controller.

- Configuring an SSI as a system interrupt for a specific core.
- Configuring an SCI to provide system interrupts to the connected core.

- Configuring an SSI as a system fault.
- Configuring the SFI to manage system faults.

Programming Examples

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

Configuring a System Source to Interrupt a Core

1. Write to the SEC_GCTL register to enable the SEC.
2. Write to the SEC_CCTLn register of the specific SCI to enable interrupts to be sent to core.
3. Write to the SEC_SCTLn register of specific source to enable the source as an interrupt and set the core target field to map the source to the appropriate SCI.

Configuring 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. Optionally 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.

ADSP-BF60x SEC Register Descriptions

System Event Controller (SEC) contains the following registers.

Table 7-5: ADSP-BF60x SEC Register List

Name	Description
SEC_CCTLn	SCI Control Register n
SEC_CSTATn	SCI Status Register n
SEC_CPNDn	Core Pending Register n
SEC_CACTn	SCI Active Register n
SEC_CPMSKn	SCI Priority Mask Register n
SEC_CGMSKn	SCI Group Mask Register n

Table 7-5: ADSP-BF60x SEC Register List (Continued)

Name	Description
SEC_CPLVLn	SCI Priority Level Register n
SEC_CSIDn	SCI Source ID Register n
SEC_FCTL	Fault Control Register
SEC_FSTAT	Fault Status Register
SEC_FSID	Fault Source ID Register
SEC_FEND	Fault End Register
SEC_FDLY	Fault Delay Register
SEC_FDLY_CUR	Fault Delay Current Register
SEC_FSRDLY	Fault System Reset Delay Register
SEC_FSRDLY_CUR	Fault System Reset Delay Current Register
SEC_FCOPP	Fault COP Period Register
SEC_FCOPP_CUR	Fault COP Period Current Register
SEC_GCTL	Global Control Register
SEC_GSTAT	Global Status Register
SEC_RAISE	Global Raise Register
SEC_END	Global End Register
SEC_SCTLn	Source Control Register n
SEC_SSTATn	Source Status Register n

SCI Control Register n

The SEC control register (SEC_CCTLn) contains SCI control bits for all system sources.

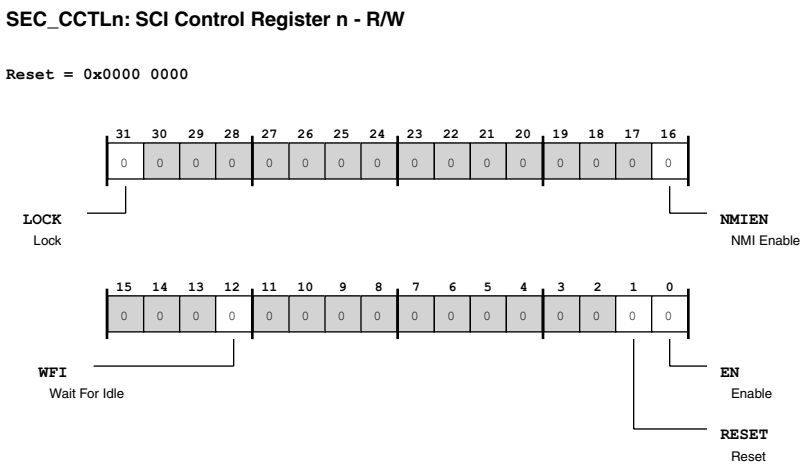


Figure 7-5: SEC_CCTLn Register Diagram

Table 7-6: SEC_CCTLn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (SPU_CTL_GLCK bit =1) and the SEC_CCTLn.LOCK bit is enabled, the SEC_CCTLn register is read only.	
		0	Unlock
		1	Lock
16 (R/W)	NMIEN	NMI Enable. The SEC_CCTLn.NMIEN bit controls NMI propagation to the core. When the SEC_CCTLn.NMIEN bit is enabled, the SCI allows NMIs to propagate to the core for servicing.	
		0	Disable
		1	Enable
12 (R0/W1A)	WFI	Wait For Idle. When set, the SEC_CCTLn.WFI bit forces the SCI to wait for indication of core idle before the SCI resumes activity.	
		0	No Action
		1	Wait for Idle
1 (R0/W1A)	RESET	Reset. When set, the SEC_CCTLn.RESET bit resets all SCI registers to their default values.	
		0	No Action
		1	Reset

Table 7-6: SEC_CCTLn Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable. The SEC_CCTLn.EN bit controls operation of the SCI. Clearing the SEC_CCTLn.EN bit halts the execution of the SCI without resetting status registers. (The INT signal to a core is not affected.) Setting the SEC_CCTLn.EN bit enables the SCI to begin or resume operation with the current configuration and status.	
		0	Disable
		1	Enable

SCI Status Register n

The SCI status register (SEC_CSTATn) contains status bits, indicating the operational status of the SCI.

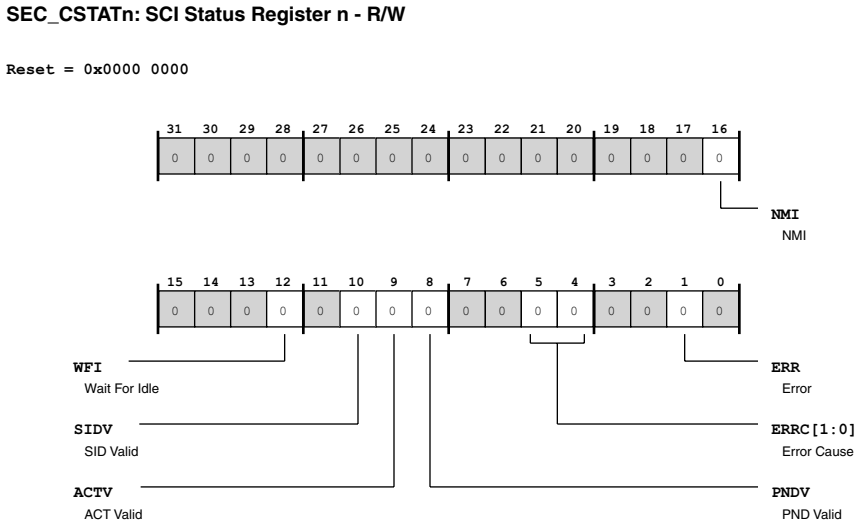


Figure 7-6: SEC_CSTATn Register Diagram

Table 7-7: SEC_CSTATn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W1C)	NMI	NMI. The SEC_CSTATn.NMI bit indicates whether an NMI has occurred since the bit was last cleared.	
		0	No NMI Occured
		1	NMI Occurred

Table 7-7: SEC_CSTATn Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1C)	WFI	Wait For Idle. The SEC_CSTATn.WFI bit indicates (if set) that the SCI is temporarily disabled, pending a core idle indication. This bit is set when SEC_CCTLn.WFI is set.	
		0	Not Waiting
		1	Waiting
10 (R/NW)	SIDV	SID Valid. The SEC_CSTATn.SIDV bit indicates (if set) that the current value in the SEC_CSIDn register is valid. The SCI sets the SEC_CSTATn.SIDV bit when the updating the SEC_CSIDn register with a new value. The SEC_CSTATn.SIDV bit is cleared when the SEC_CSIDn register is written. This status indication may be used to extract all pending interrupts in a single interrupt service routine.	
		0	Invalid
		1	Valid
9 (R/NW)	ACTV	ACT Valid. The SEC_CSTATn.ACTV bit indicates (if set) that the current value in the SEC_CACTn register is valid. The SCI sets the SEC_CSTATn.ACTV bit when updating the SEC_CACTn registers with a new value. The SEC_CSTATn.ACTV bit is cleared when the SEC_CSIDn register is written.	
		0	Invalid
		1	Valid
8 (R/NW)	PNDV	PND Valid. The SEC_CSTATn.PNDV bit indicates (if set) that the current value in the SEC_CPNDn register is valid. The SCI sets the SEC_CSTATn.PNDV bit when updating the SEC_CPNDn register with a new value. The SEC_CSTATn.PNDV bit is cleared when the SEC_CSIDn register is written.	
		0	Invalid
		1	Valid

Table 7-7: SEC_CSTATn Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/NW)	ERRC	Error Cause. The SEC_CSTATn.ERRC bits are updated on assertion of the SEC_CSTATn.ERR bit to indicate the SCI error type. SEC_CSTATn.ERRC is only updated on the assertion of SEC_CSTATn.ERR. Subsequent errors while SEC_CSTATn.ERR is asserted do not update SEC_CSTATn.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_CSTATn.ERR bit indicates that an error has occurred in the SCI. When SEC_CSTATn.ERR is set, the SCI updates the SEC_CSTATn.ERRC field to the value of the corresponding error cause.	
		0	No Error
		1	Error Occurred

Core Pending Register n

The SCI pending interrupt register (SEC_CPNDn) contains the source ID and priority of the highest priority pending interrupt detected by the SEC prioritizer.

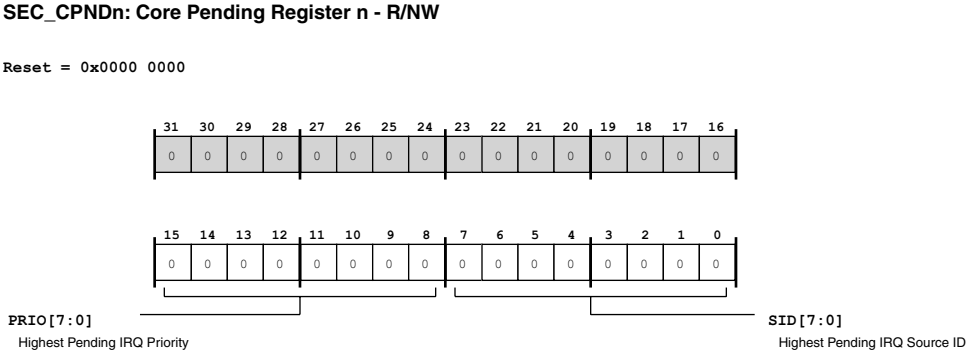


Figure 7-7: SEC_CPNDn Register Diagram

Table 7-8: SEC_CPNDn Register Fields

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

SCI Active Register n

The SEC SCI active interrupt register (SEC_CACTn) contains the source ID and priority of the highest priority active interrupt detected by the SEC prioritizer.

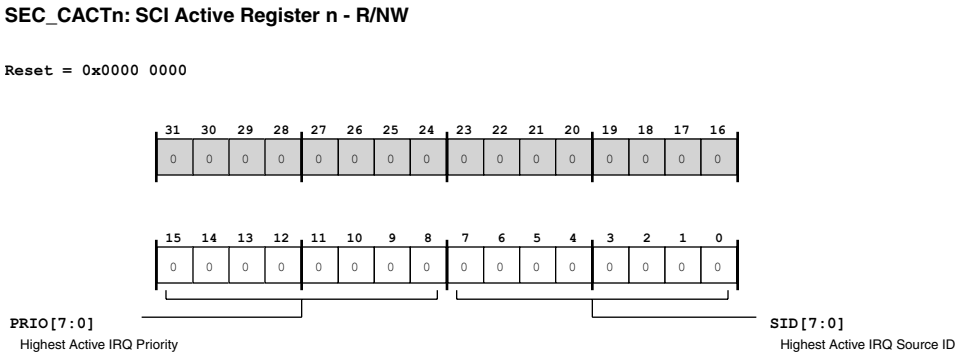


Figure 7-8: SEC_CACTn Register Diagram

Table 7-9: SEC_CACTn Register Fields

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

SCI Priority Mask Register n

The SEC SCI priority mask register (SEC_CPMSKn) contains the SCI priority mask for core n and includes a register lock.

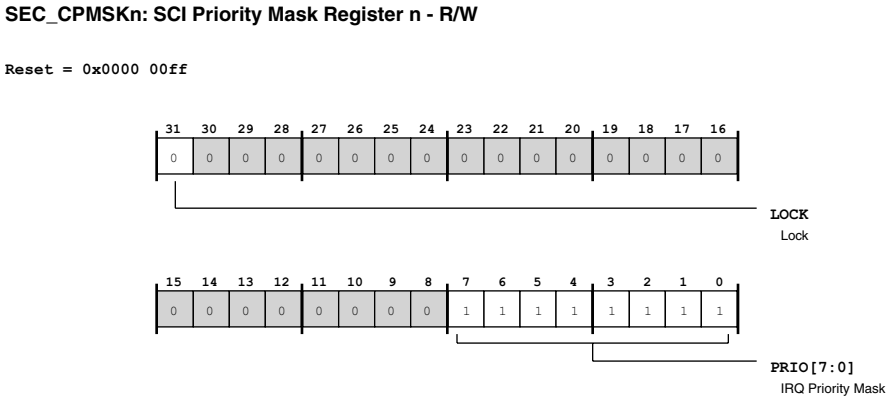


Figure 7-9: SEC_CPMSKn Register Diagram

Table 7-10: SEC_CPMSKn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (SPU_CTL_GLCK bit =1) and the SEC_CPMSKn.LOCK bit is enabled, the SEC_CPMSKn register is read only.	
		0	Unlock
		1	Lock
7:0 (R/W)	PRIO	IRQ Priority Mask. The SEC_CPMSKn.PRIO contains the system interrupt priority mask for core n. The core uses the SEC_CPMSKn.PRIO field to mask (block) interrupts below the specified level. 0b11111111 = Priority Level 255 (lowest) ... 0b00000000 = Priority Level 0 (highest)	

SCI Group Mask Register n

The SEC SCI group mask register (SEC_CGMSKn) 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_CGMSKn.UGRP and SEC_CGMSKn.GRP fields to mask (disable) interrupts from the specified groups.

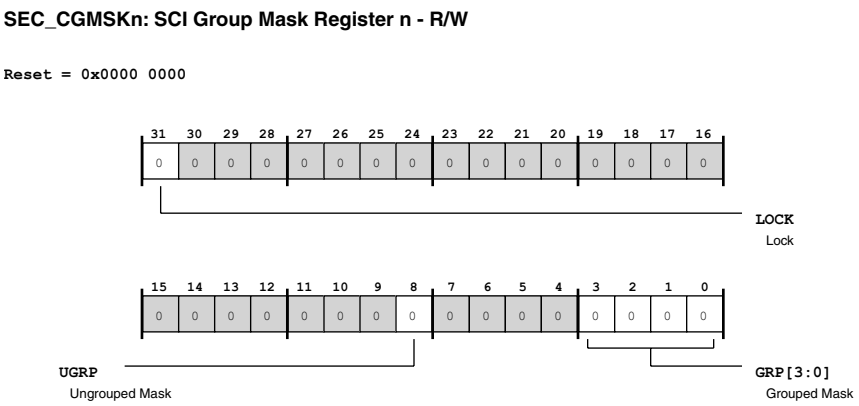


Figure 7-10: SEC_CGMSKn Register Diagram

Table 7-11: SEC_CGMSKn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (SPU_CTL_GLCK bit =1) and the SEC_CGMSKn.LOCK bit is enabled, the SEC_CGMSKn register is read only.	
		0	Unlock
		1	Lock
8 (R/W)	UGRP	Ungrouped Mask. The SEC_CGMSKn.UGRP 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 SEC_CGMSKn.GRP field selects a group of interrupt sources to mask for core n. (For more information about interrupt source groups, see the SEC_SCTLn register description.) 1111 = Mask group 0, 1, 2, 3 ... 0001 = Mask group 0 0000 = No groups masked	

SCI Priority Level Register n

The SEC SCI priority level register (SEC_CPLVLn) 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.

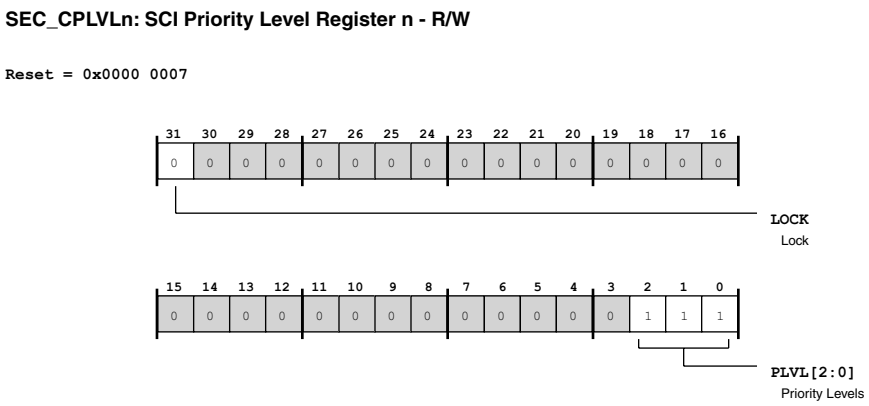


Figure 7-11: SEC_CPLVLn Register Diagram

Table 7-12: SEC_CPLVLn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (SPU_CTL_GLCK bit =1) and the SEC_CPLVLn.LOCK bit is enabled, the SEC_CPLVLn register is read only.	
		0	Unlock
		1	Lock
2:0 (R/W)	PLVL	Priority Levels. The SEC_CPLVLn.PLVL 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 intrrupt 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 SEC_CPLVLn.PLVL 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. 000 = 1 MSB (2 priority levels) ... 111 = 8 MSBs (256 priority levels)	

SCI Source ID Register n

The SCI source ID register (SEC_CSIDn) contains the source ID of the interrupt last issued to core n. The SEC_CSIDn register value is loaded by the SCI when a system interrupt indication is sent to core n. The SCI does not change the SEC_CSIDn until after the interface receives an interrupt acknowledge from core n.

Writing to the SEC_CSIDn register generates an interrupt acknowledge, but does not update the value in the register.

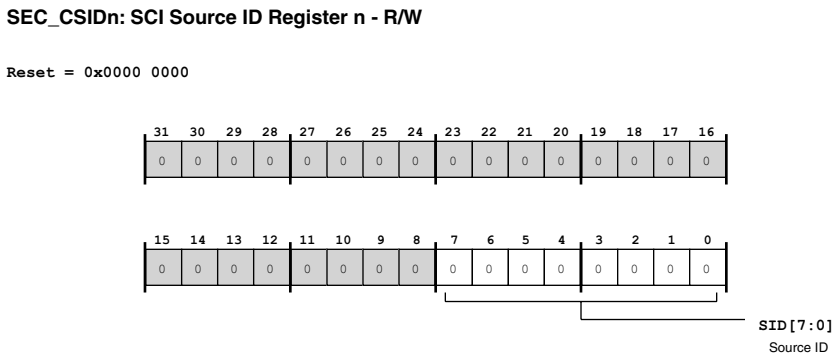


Figure 7-12: SEC_CSIDn Register Diagram

Table 7-13: SEC_CSIDn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	SID	Source ID.

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).

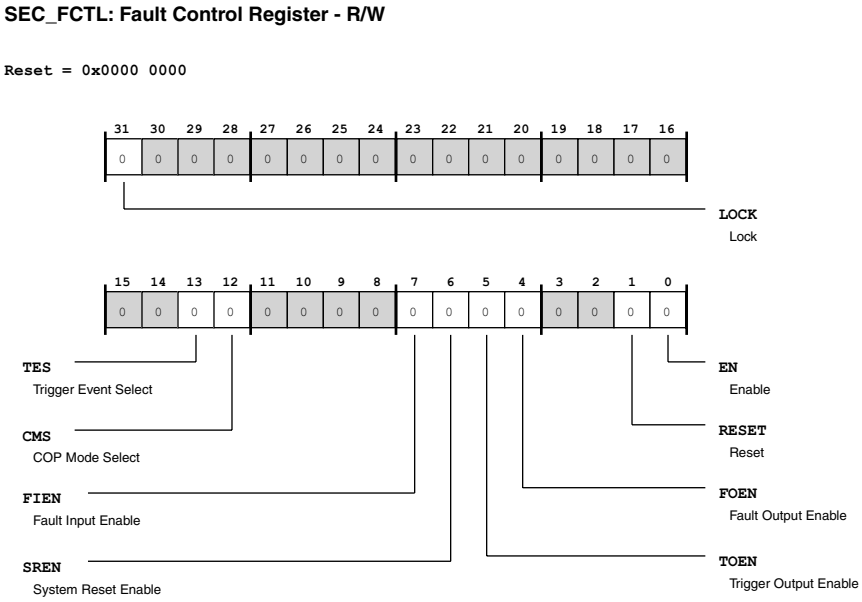


Figure 7-13: SEC_FCTL Register Diagram

Table 7-14: SEC_FCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (SPU_CTL_GLCK bit =1) and the SEC_FCTL.LOCK bit is enabled, the SEC_FCTL register is read only.	
		0	UnLock
		1	Lock
13 (R/W)	TES	Trigger Event Select. The SEC_FCTL.TES 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
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.	
		0	Fault Mode
		1	COP Mode
7 (R/W)	FIEN	Fault Input Enable. The SEC_FCTL.FIEN bit enables the SEC the 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

Table 7-14: SEC_FCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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/W1A)	RESET	Reset. Setting the SEC_FCTL.RESET bit resets ALL SEC registers to their default values.	
		0	No Action
		1	Reset
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 Status Register

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

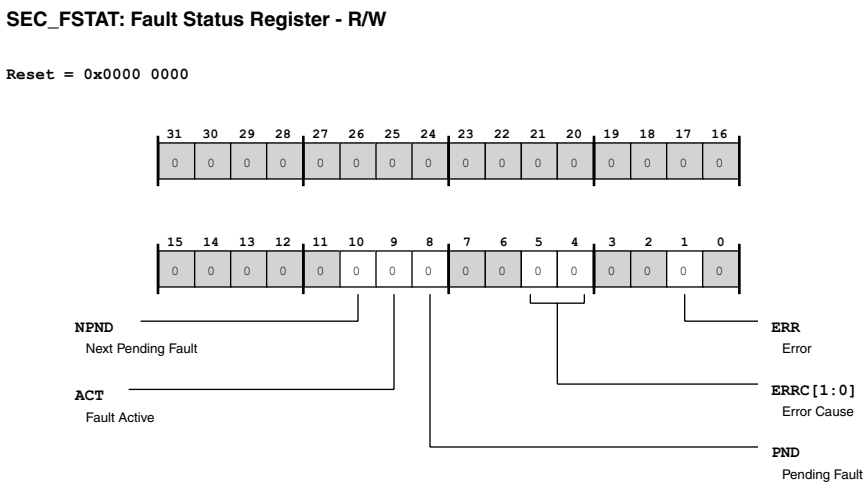


Figure 7-14: SEC_FSTAT Register Diagram

Table 7-15: SEC_FSTAT Register Fields

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

Table 7-15: SEC_FSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	PND	Pending Fault. The SEC_FSTAT.PND bit indicates a fault source has signalled 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 *either* by the SEC fault interface when the current delay count in the SEC_FDLY_CUR register expires *or* by writing the SEC_FSID.SID field value (which indicates the ID of the asserted fault) to the SEC_FEND register.	
		0	Not Pending
		1	Pending
5:4 (R/NW)	ERRC	Error Cause. When the SEC updates the SEC_FSTAT.ERR bit, the SEC updates the SEC_FSTAT.ERRC bits to indicate the error type. When the error status is End Error, the status indicates two possible error scenarios. Either, the SEC received a write to SEC_FEND while neither the pending fault bit (SEC_FSTAT.PND) nor fault active bit (SEC_FSTAT.ACT) were set, or the SEC detected that the SID written to SEC_FEND.SID does not match the fault source indicated in the SEC_FSID.SID field.	
		0	Reserved
		1	Reserved
		2	End Error
		3	Reserved
1 (R/W1C)	ERR	Error. 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.	
		0	No Error
		1	Error Occurred

Fault Source ID Register

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

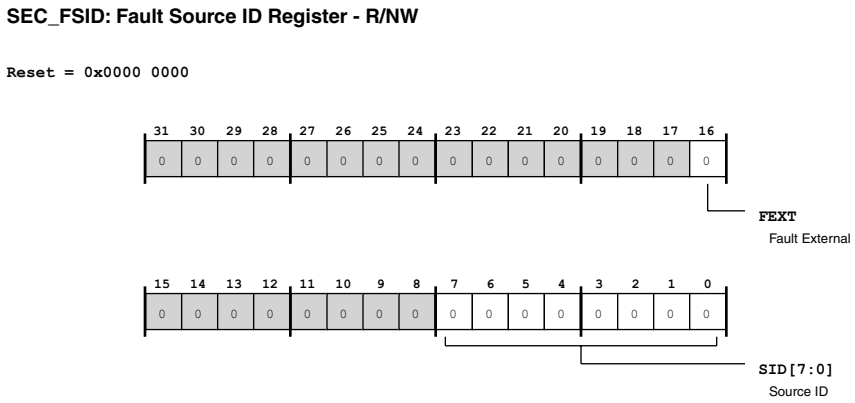


Figure 7-15: SEC_FSID Register Diagram

Table 7-16: SEC_FSID Register Fields

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

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.

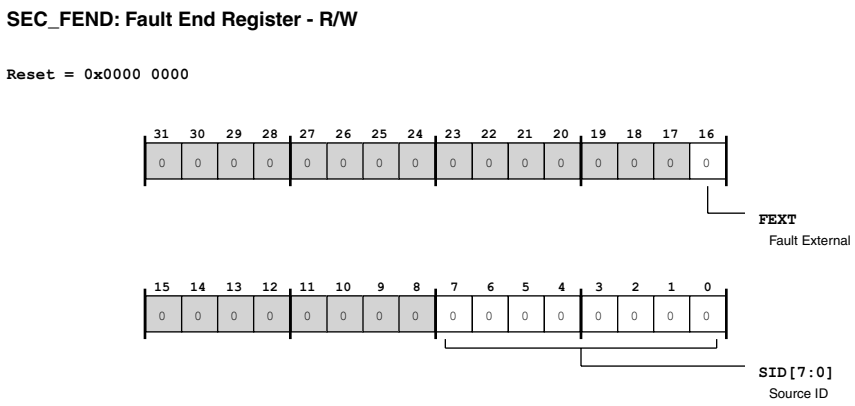


Figure 7-16: SEC_FEND Register Diagram

Table 7-17: SEC_FEND Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	FEXT	Fault External. Setting the SEC_FEND.FEXT bit, when the SEC_FEND.SID 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 SEC_FEND.SID identifies a fault to be ended as indicated to the SEC by the core. The core loads the SEC_FEND.SID field value. If the SEC_FEND.SID value matches the SEC_FSID.SID value, the SEC_FSTAT.PND bit and SEC_FSTAT.ACT bit are cleared.	

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.

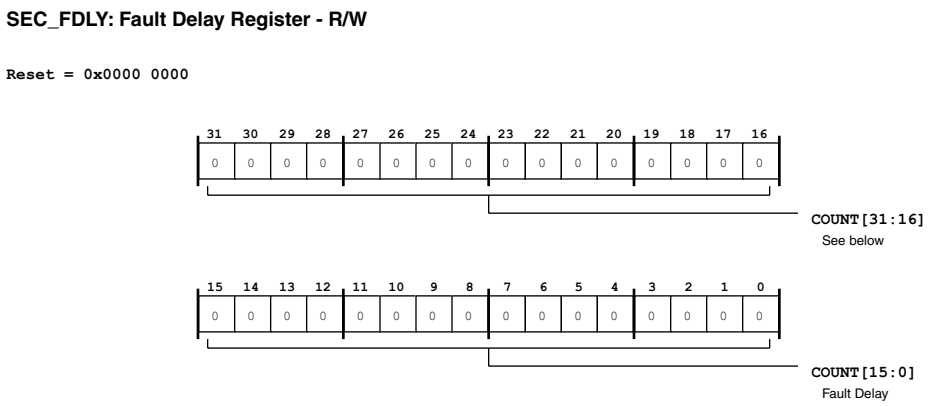


Figure 7-17: SEC_FDLY Register Diagram

Table 7-18: SEC_FDLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	COUNT	Fault Delay.

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.

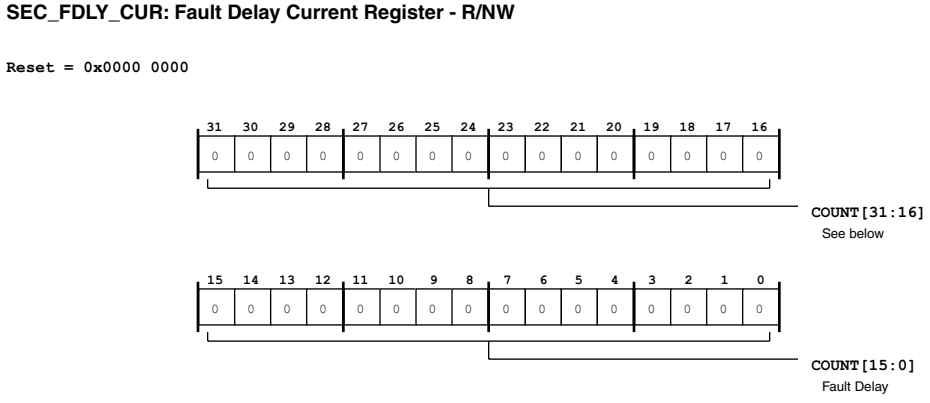


Figure 7-18: SEC_FDLY_CUR Register Diagram

Table 7-19: SEC_FDLY_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	COUNT	Fault Delay.

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.

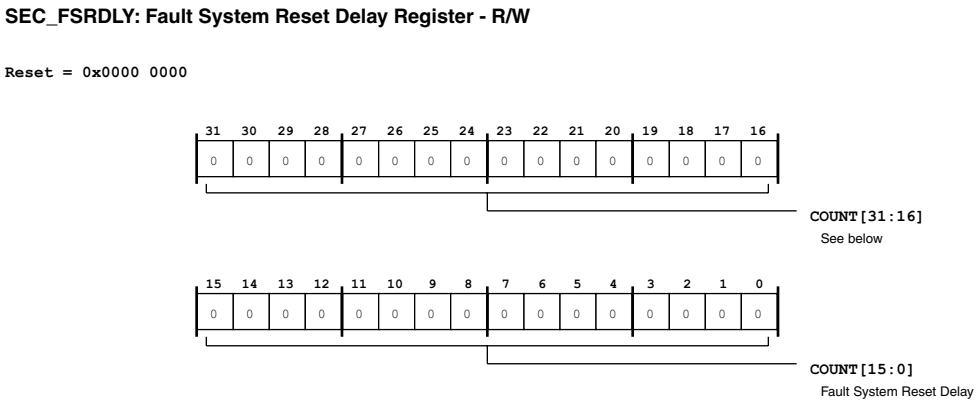


Figure 7-19: SEC_FSRDLY Register Diagram

Table 7-20: SEC_FSRDLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	COUNT	Fault System Reset Delay.

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.

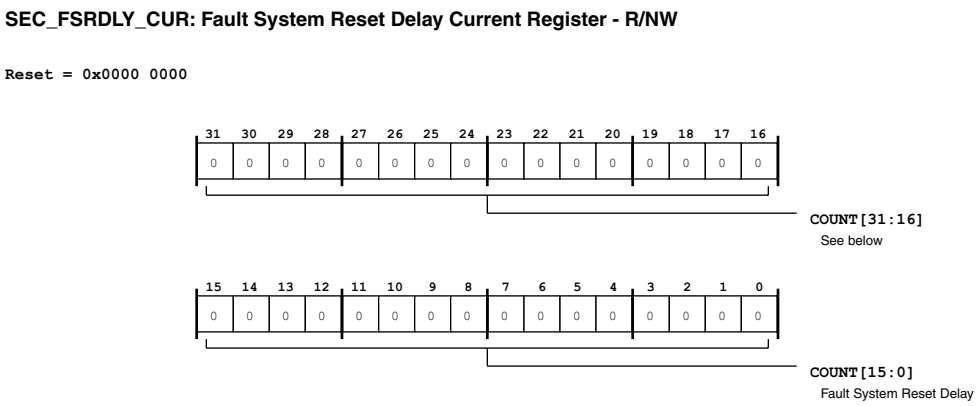


Figure 7-20: SEC_FSRDLY_CUR Register Diagram

Table 7-21: SEC_FSRDLY_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	COUNT	Fault System Reset Delay.

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 is value is the SEC_FCOPP.COUNT programmed value plus 1.

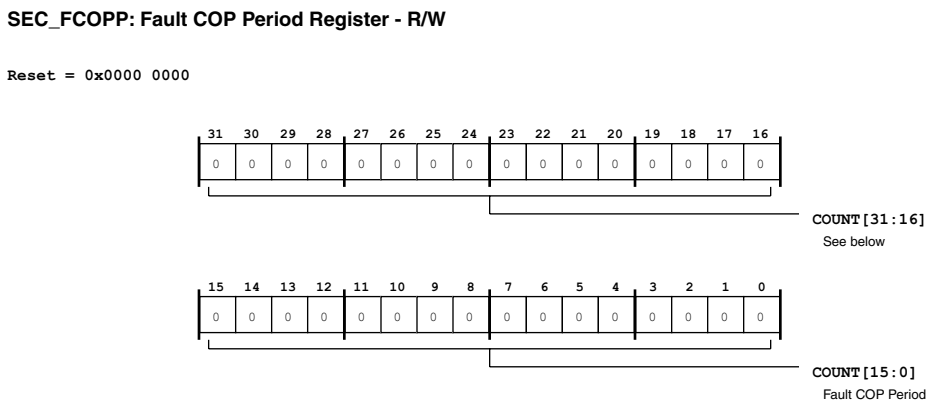


Figure 7-21: SEC_FCOPP Register Diagram

Table 7-22: SEC_FCOPP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	COUNT	Fault COP Period.

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.

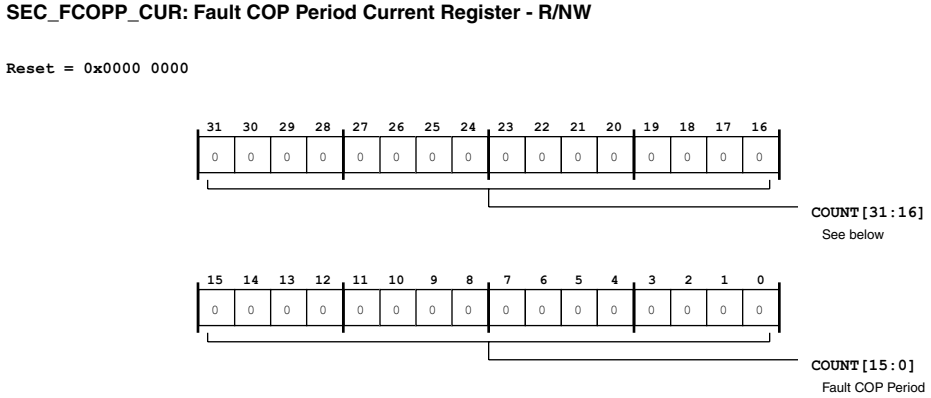


Figure 7-22: SEC_FCOPP_CUR Register Diagram

Table 7-23: SEC_FCOPP_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	COUNT	Fault COP Period.

Global Control Register

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

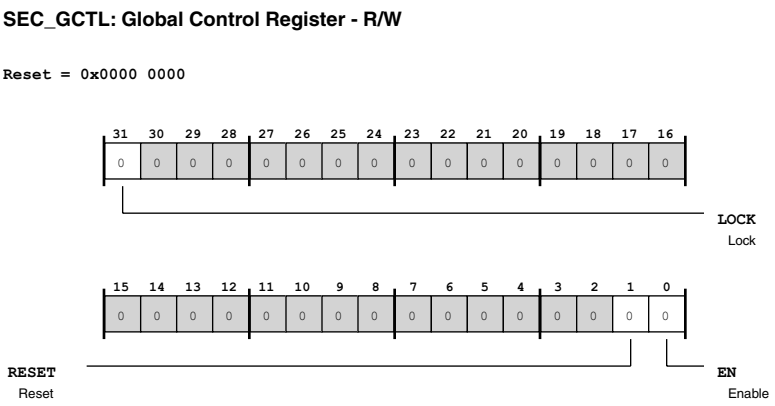


Figure 7-23: SEC_GCTL Register Diagram

Table 7-24: SEC_GCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (SPU_CTL_GLCK bit =1) and the SEC_GCTL.LOCK bit is enabled, the SEC_GCTL register is read only.	
		0	Unlock
		1	Lock
1 (R0/W1A)	RESET	Reset. The SEC_GCTL.RESET 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 SEC_GCTL.EN bit is read/write and must be set for the SEC to begin/resume SEC operation with the current configuration and status. Clearing SEC_GCTL.EN bit halts the execution of the SEC core interface (SCI). All SEC fault interfaces (SFI) and SEC source interfaces (SSI) 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.

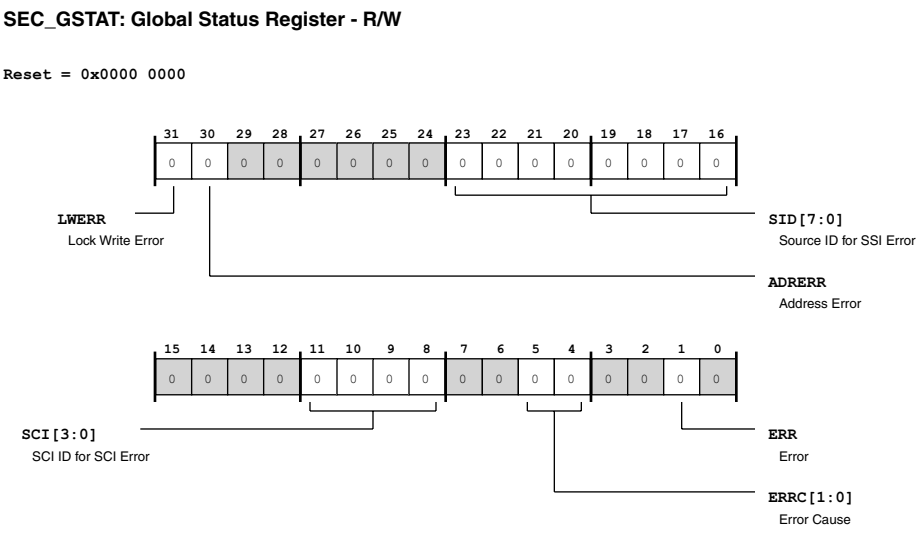


Figure 7-24: SEC_GSTAT Register Diagram

Table 7-25: SEC_GSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W1C)	LWERR	Lock Write Error. The SEC_GSTAT.LWERR bit indicates (when set) there was an attempted write to an SEC register while the SEC_GCTL.LOCK bit was set and while the global lock bit was enabled (SPU_CTL_GLCK 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 SEC_GSTAT.ADRERR bit indicates that the SEC generated and 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 SEC_GSTAT.SID bits indicate the source ID that generated the last SSI Error conveyed in the SEC_GSTAT.ERRC field.	
11:8 (R/NW)	SCI	SCI ID for SCI Error. The SEC_GSTAT.SCI bits indicate the number for the specific SCI that generated the last SCI error conveyed in the SEC_GSTAT.ERRC field.	

Table 7-25: 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 interrupt set to pending field (SEC_RAISE.SID). When a source ID value is written to this field, the SEC raises the source's interrupt to pending.

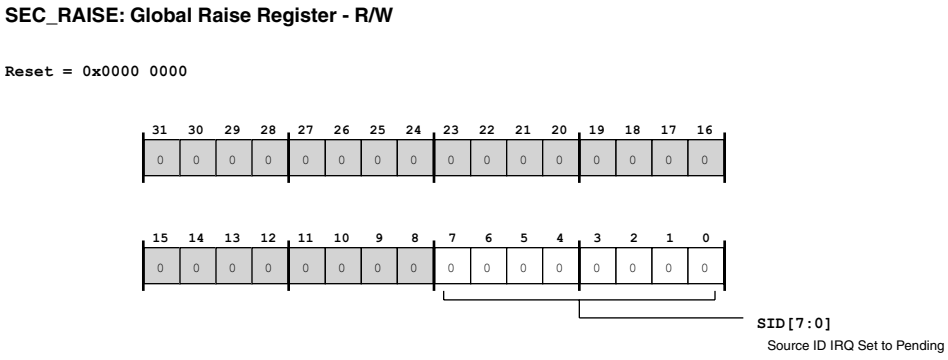


Figure 7-25: SEC_RAISE Register Diagram

Table 7-26: SEC_RAISE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	SID	Source ID IRQ Set to Pending.

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_SSTATn.ACT bit in the SEC_SSTATn register of the corresponding interrupt.

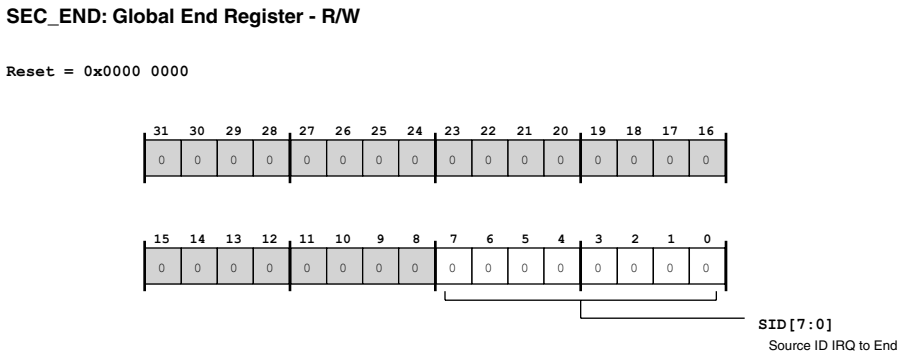


Figure 7-26: SEC_END Register Diagram

Table 7-27: SEC_END Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	SID	Source ID IRQ to End.

Source Control Register n

The SEC source control register (SEC_SCTLn) contains control bits to configure the SEC interrupt sources. This register controls the configuration of the corresponding SEC interrupt source.

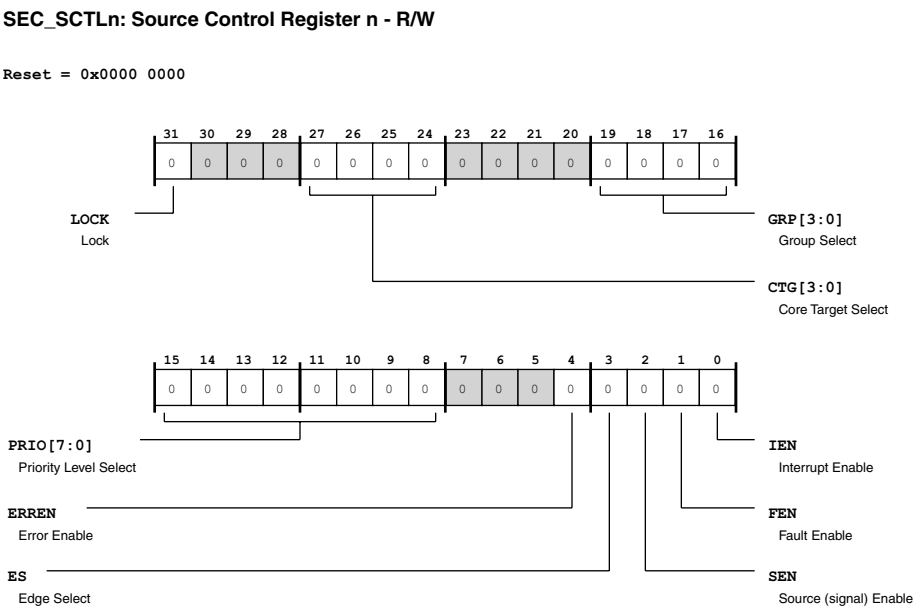


Figure 7-27: SEC_SCTLn Register Diagram

Table 7-28: SEC_SCTLn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (SPU_CTL_GLCK bit =1) and the SEC_SCTLn.LOCK bit is enabled, the SEC_SCTLn register is read only.	
		0	Unlock
		1	Lock
27:24 (R/W)	CTG	Core Target Select. The SEC_SCTLn.CTG 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 7-28: SEC_SCTLn Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	GRP	Group Select. The SEC_SCTLn.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_SCTLn.GRP field. For example, consider the situation where SEC_SCTLn.GRP0 represents interrupt group 0, SEC_SCTLn.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. 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_SCTLn.GRP bits = 0) places the interrupt source in the category of "ungrouped".	
15:8 (R/W)	PRIO	Priority Level Select. The SEC_SCTLn.PRIO bits sets the relative priority for an interrupt request. A pending interrupt request forwards its SEC_SCTLn.PRIO value to the SEC core interface.	
4 (R/W)	ERREN	Error Enable. The SEC_SCTLn.ERREN bit permits the SEC_SSTATn.ERR status bit to be set on error detection. If SEC_SCTLn.ERREN is cleared, no errors are detected.	
		0	Disable
		1	Enable
3 (R/W)	ES	Edge Select. The SEC_SCTLn.ES bit selects the operational and sensitivity mode of the SEC source interface input.	
		0	Level Sensitive
		1	Edge Sensitive
2 (R/W)	SEN	Source (signal) Enable. The SEC_SCTLn.SEN bit controls whether the system interrupt source input signal may affect the pending status of the source. Clearing SEC_SCTLn.SEN disables the source input signal from affecting pending. Setting SEC_SCTLn.SEN enables the source input signal to affect pending.	
		0	Disable
		1	Enable

Table 7-28: SEC_SCTLn Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	FEN	Fault Enable. The SEC_SCTLn.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_SCTLn.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_SCTLn.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 interrupt source status register (SEC_SSTATn) contains bits indicating the status of the corresponding interrupt source n. An interrupt source may be: pending, active, active and pending, or neither pending nor active.

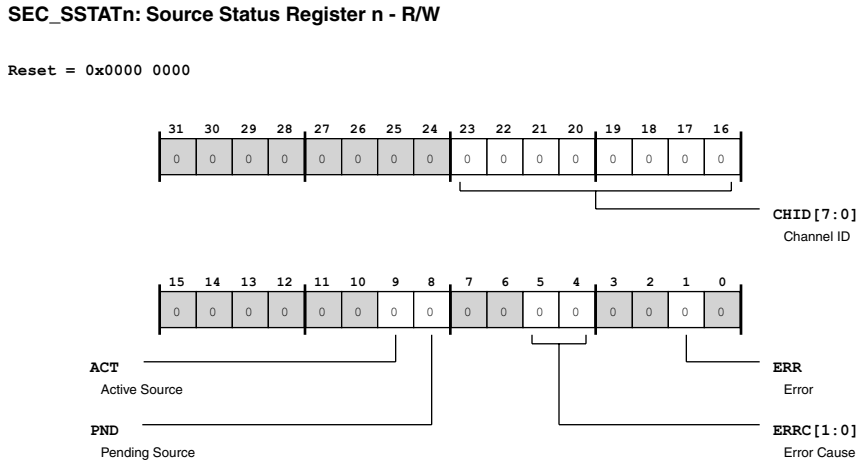


Figure 7-28: SEC_SSTATn Register Diagram

Table 7-29: SEC_SSTATn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:16 (R/NW)	CHID	Channel ID. The SEC_SSTATn.CHID 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 SEC_SSTATn.CHID 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 SEC_SSTATn.CHID is captured when the SEC source interface input is acknowledged.	
9 (R/W1C)	ACT	Active Source. The SEC_SSTATn.ACT bit indicates the source has been accepted by a core for servicing, but the service is not yet complete. An SEC_SSTATn.ACT bit is set by the SEC when the specific system interrupt is acknowledged by the core through the SEC core interface. An SEC_SSTATn.ACT 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	No Source
		1	Active Source
8 (R/W1C)	PND	Pending Source. The SEC_SSTATn.PND bit indicates the source has signalled an interrupt request assertion, but the request has not yet been accepted by a core for servicing. A SEC_SSTATn.PND bit is set by the SEC on detection of an assertion of the corresponding system interrupt input. A SEC_SSTATn.PND bit is cleared by the SEC when the specific system interrupt is acknowledged by the core through the SEC core interface.	
		0	Not Pending
		1	Pending

Table 7-29: SEC_SSTATn Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/NW)	ERRC	Error Cause. When the SEC_SSTATn.ERR bit is asserted, the SEC updates SEC_SSTATn.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_SSTATn.ACT bit was not set.	
		0	Source Overflow Error
		1	Reserved
		2	End Error
		3	Reserved
1 (R/W1C)	ERR	Error. The SEC_SSTATn.ERR bit indicates an error for a specific system interrupt source. When the SEC_SSTATn.ERR bit is set, the SEC updates the SEC_SSTATn.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_SSTATn.ERR bit.	
		0	No Error
		1	Error Occurred

8 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. 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.

- Trigger routing of any trigger master to any trigger slave.
- Software generation of any trigger master ID.
- Configuration protection through register level lock bits and global lock indication.

TRU Functional Description

The following sections provide a function description of the TRU.

ADSP-BF60x 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 resides in the SYSCLK domain and 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 govern TRU operations. For more information on TRU functionality, see the TRU register descriptions.

Table 8-1: ADSP-BF60x TRU Register List

Name	Description
TRU_SSRn	Slave Select Register
TRU_MTR	Master Trigger Register
TRU_ERRADDR	Error Address Register
TRU_STAT	Status Information Register
TRU_GCTL	Global Control Register

ADSP-BF60x TRU Interrupt List

Table 8-2: ADSP-BF60x TRU Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
TRU0 Interrupt 0	124		PULSE/EDGE
TRU0 Interrupt 1	125		PULSE/EDGE
TRU0 Interrupt 2	126		PULSE/EDGE
TRU0 Interrupt 3	127		PULSE/EDGE

ADSP-BF60x Trigger List

Table 8-3: ADSP-BF60x Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
Reserved	0	
CGU0 Event	1	PULSE/EDGE
TIMER0 Timer 0	2	PULSE/EDGE
TIMER0 Timer 1	3	PULSE/EDGE
TIMER0 Timer 2	4	PULSE/EDGE
TIMER0 Timer 3	5	PULSE/EDGE
TIMER0 Timer 4	6	PULSE/EDGE
TIMER0 Timer 5	7	PULSE/EDGE
TIMER0 Timer 6	8	PULSE/EDGE

Table 8-3: ADSP-BF60x Trigger List Trigger Masters (Continued)

Description	Trigger ID	Sensitivity
TIMER0 Timer 7	9	PULSE/EDGE
PINT0 Pin Interrupt Block	10	LEVEL
PINT1 Pin Interrupt Block	11	LEVEL
PINT2 Pin Interrupt Block	12	LEVEL
PINT3 Pin Interrupt Block	13	LEVEL
PINT4 Pin Interrupt Block	14	LEVEL
PINT5 Pin Interrupt Block	15	LEVEL
CNT0 Status	16	LEVEL
PWM0 PWMTMR Group	17	LEVEL
PWM1 PWMTMR Group	18	LEVEL
ACM0 Event Complete	19	LEVEL
SPORT0 Channel A DMA	20	PULSE/EDGE
SPORT0 Channel B DMA	21	PULSE/EDGE
SPORT1 Channel A DMA	22	PULSE/EDGE
SPORT1 Channel B DMA	23	PULSE/EDGE
SPORT2 Channel A DMA	24	PULSE/EDGE
SPORT2 Channel B DMA	25	PULSE/EDGE
SPI0 TX DMA Channel	26	PULSE/EDGE
SPI0 RX DMA Channel	27	PULSE/EDGE
SPI1 TX DMA Channel	28	PULSE/EDGE
SPI1 RX DMA Channel	29	PULSE/EDGE
RSI0 DMA Channel	30	PULSE/EDGE
SDU0 DMA	31	PULSE/EDGE
Reserved	32	
EMAC0 Status	33	LEVEL
EMAC1 Status	34	LEVEL
LP0 DMA Channel	35	PULSE/EDGE
LP1 DMA Channel	36	PULSE/EDGE
LP2 DMA Channel	37	PULSE/EDGE
LP3 DMA Channel	38	PULSE/EDGE
UART0 Transmit DMA	39	PULSE/EDGE

Table 8-3: ADSP-BF60x Trigger List Trigger Masters (Continued)

Description	Trigger ID	Sensitivity
UART0 Receive DMA	40	PULSE/EDGE
UART1 Transmit DMA	41	PULSE/EDGE
UART1 Receive DMA	42	PULSE/EDGE
Memory DMA Stream 0 Source / CRC0 Input Channel	43	PULSE/EDGE
Memory DMA Stream 0 Destination / CRC0 Output Channel	44	PULSE/EDGE
Memory DMA Stream 1 Source / CRC1 Input Channel	45	PULSE/EDGE
Memory DMA Stream 1 Destination / CRC1 Output Channel	46	PULSE/EDGE
Memory DMA Stream 2 Source Channel	47	PULSE/EDGE
Memory DMA Stream 2 Destination Channel	48	PULSE/EDGE
Memory DMA Stream 3 Source Channel	49	PULSE/EDGE
Memory DMA Stream 3 Destination Channel	50	PULSE/EDGE
EPPI0 Channel 0 DMA	51	PULSE/EDGE
EPPI0 Channel 1 DMA	52	PULSE/EDGE
EPPI2 Channel 0 DMA	53	PULSE/EDGE
EPPI2 Channel 1 DMA	54	PULSE/EDGE
EPPI1 Channel 0 DMA	55	PULSE/EDGE
EPPI1 Channel 1 DMA	56	PULSE/EDGE
PIXC0 Channel 0 DMA	57	PULSE/EDGE
PIXC0 Channel 1 DMA	58	PULSE/EDGE
PIXC0 Channel 2 DMA	59	PULSE/EDGE
PVP0 Camera Pipe Data Out B DMA Channel	60	PULSE/EDGE
PVP0 Camera Pipe Data Out C DMA Channel	61	PULSE/EDGE

Table 8-3: ADSP-BF60x Trigger List Trigger Masters (Continued)

Description	Trigger ID	Sensitivity
PVP0 Camera Pipe Status Out DMA Channel	62	PULSE/EDGE
PVP0 Camera Pipe Control In DMA Channel	63	PULSE/EDGE
PVP0 Memory Pipe Data Out DMA Channel	64	PULSE/EDGE
PVP0 Memory Pipe Data In DMA Channel	65	PULSE/EDGE
PVP0 Memory Pipe Status Out DMA Channel	66	PULSE/EDGE
PVP0 Memory Pipe Control In DMA Channel	67	PULSE/EDGE
PVP0 Camera Pipe Data Out A DMA Channel	68	PULSE/EDGE
USB0 DMA Status/Transfer Complete	69	LEVEL
Reserved	70	
SEC0 Fault	71	PULSE/EDGE
Software-driven Trigger 0	72	PULSE/EDGE
Software-driven Trigger 1	73	PULSE/EDGE
Software-driven Trigger 2	74	PULSE/EDGE
Software-driven Trigger 3	75	PULSE/EDGE
Software-driven Trigger 4	76	PULSE/EDGE
Software-driven Trigger 5	77	PULSE/EDGE
PVP0 Status 0	78	LEVEL
PVP0 Status 1	79	LEVEL
SWU0 Event	80	PULSE/EDGE
SWU1 Event	81	PULSE/EDGE
SWU2 Event	82	PULSE/EDGE
SWU3 Event	83	PULSE/EDGE
SWU4 Event	84	PULSE/EDGE
SWU5 Event	85	PULSE/EDGE
SWU6 Event	86	PULSE/EDGE

Table 8-4: ADSP-BF60x Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
RCU0 System Reset 0	0	
RCU0 System Reset 1	1	
TIMER0 Timer 0	2	
TIMER0 Timer 1	3	
TIMER0 Timer 2	4	
TIMER0 Timer 3	5	
TIMER0 Timer 4	6	
TIMER0 Timer 5	7	
TIMER0 Timer 6	8	
TIMER0 Timer 7	9	
Reserved	10	
Reserved	11	
NMI (Core 0) Slave 0	12	
NMI (Core 0) Slave 1	13	
NMI (Core 1) Slave 0	14	
NMI (Core 1) Slave 1	15	
TRU0 Interrupt Request 0	16	
TRU0 Interrupt Request 1	17	
TRU0 Interrupt Request 2	18	
TRU0 Interrupt Request 3	19	
SPORT0 Channel A DMA	20	
SPORT0 Channel B DMA	21	
SPORT1 Channel A DMA	22	
SPORT1 Channel B DMA	23	
SPORT2 Channel A DMA	24	
SPORT2 Channel B DMA	25	
SPI0 TX DMA Channel	26	
SPI0 RX DMA Channel	27	
SPI1 TX DMA Channel	28	
SPI1 RX DMA Channel	29	

Table 8-4: ADSP-BF60x Trigger List Trigger Slaves (Continued)

Description	Trigger ID	Sensitivity
RSI0 DMA Channel	30	
SDU0 DMA	31	
Reserved	32	
ACM0 Trigger Input 2	33	
ACM0 Trigger Input 3	34	
LP0 DMA Channel	35	
LP1 DMA Channel	36	
LP2 DMA Channel	37	
LP3 DMA Channel	38	
UART0 Transmit DMA	39	
UART0 Receive DMA	40	
UART1 Transmit DMA	41	
UART1 Receive DMA	42	
Memory DMA Stream 0 Source / CRC0 Input Channel	43	
Memory DMA Stream 0 Destination / CRC0 Output Channel	44	
Memory DMA Stream 1 Source / CRC1 Input Channel	45	
Memory DMA Stream 1 Destination / CRC1 Output Channel	46	
Memory DMA Stream 2 Source Channel	47	
Memory DMA Stream 2 Destination Channel	48	
Memory DMA Stream 3 Source Channel	49	
Memory DMA Stream 3 Destination Channel	50	
EPPI0 Channel 0 DMA	51	
EPPI0 Channel 1 DMA	52	
EPPI2 Channel 0 DMA	53	

Table 8-4: ADSP-BF60x Trigger List Trigger Slaves (Continued)

Description	Trigger ID	Sensitivity
EPPI2 Channel 1 DMA	54	
EPPI1 Channel 0 DMA	55	
EPPI1 Channel 1 DMA	56	
PIXC0 Channel 0 DMA	57	
PIXC0 Channel 1 DMA	58	
PIXC0 Channel 2 DMA	59	
PVP0 Camera Pipe Data Out B DMA Channel	60	
PVP0 Camera Pipe Data Out C DMA Channel	61	
PVP0 Camera Pipe Status Out DMA Channel	62	
PVP0 Camera Pipe Control In DMA Channel	63	
PVP0 Memory Pipe Data Out DMA Channel	64	
PVP0 Memory Pipe Data In DMA Channel	65	
PVP0 Memory Pipe Status Out DMA Channel	66	
PVP0 Memory Pipe Control In DMA Channel	67	
PVP0 Camera Pipe Data Out A DMA Channel	68	
SDU0 Slave Trigger	69	
Reserved	70	
Core 0 Wakeup Input 0	71	
Core 0 Wakeup Input 1	72	
Core 0 Wakeup Input 2	73	
Core 0 Wakeup Input 3	74	
Core 1 Wakeup Input 0	75	
Core 1 Wakeup Input 1	76	
Core 1 Wakeup Input 2	77	
Core 1 Wakeup Input 3	78	

Table 8-4: ADSP-BF60x Trigger List Trigger Slaves (Continued)

Description	Trigger ID	Sensitivity
Reserved	79	
SWU0 Event	80	
SWU1 Event	81	
SWU2 Event	82	
SWU3 Event	83	
SWU4 Event	84	
SWU5 Event	85	
SWU6 Event	86	

TRU Definitions

Trigger Master

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

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. A trigger event response is defined by the trigger slave modules.

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.

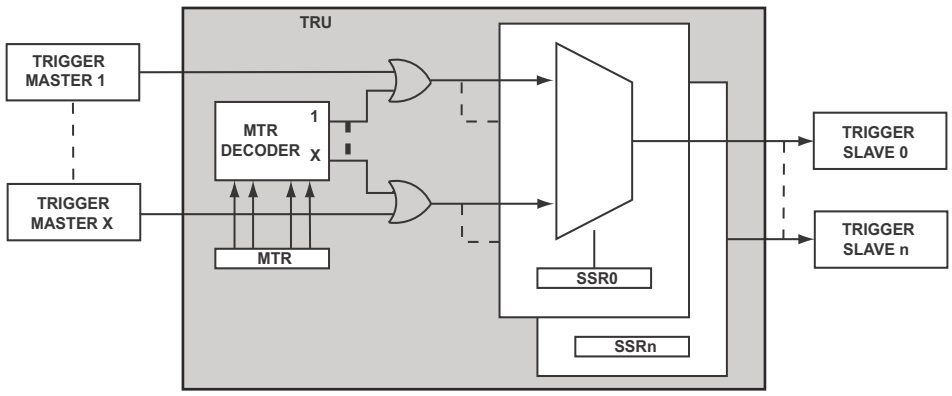


Figure 8-1: TRU Block Diagram

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_SSRn`).

System modules may be 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.

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 should be configured and enable as the last step.

The other steps that must be completed are:

- The trigger slave must be configured for response to triggers.
- The TRU must be configured to map the trigger master to the trigger slave through the `TRU_SSRn` registers.
- The trigger master must be configured to generate trigger assertions.
- Alternatively, software triggering may be used for trigger assertion. Software triggers are generated by writing the trigger master ID to the MTR register.

Programming Concepts

The following concepts will aid in programming the TRU.

- **Trigger Sequence Configuration.** A simple sequence may consist of one trigger master and one trigger slave. More complex trigger sequences may consist of several trigger slaves functioning as trigger slave and trigger master. Additionally, trigger sequences may loop back 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 and/or by generating multiple trigger master assertions simultaneously through the MTR.
- **Configuration Protection.** The `TRU_SSRn.LOCK` bit and the `TRU_GCTL.LOCK` bit enable register level write protection when global lock is asserted in the SPU.

Programming Example

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

1. Write to the `TRU_GCTL` register to enable the TRU.
2. Write to the `TRU_SSRn` register of a specific trigger slave to assign it to a specific trigger master.
3. Enable the trigger slave to wait for and accept a trigger.
4. Enable the trigger master to generate a trigger.

TRU Event Control

The TRU is a major part of event control solutions. It is the center of the trigger functional model and may be extended 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 bus protocol.

ADSP-BF60x TRU Register Descriptions

Trigger Routing Unit (TRU) contains the following registers.

Table 8-5: ADSP-BF60x TRU Register List

Name	Description
TRU_SSRn	Slave Select Register
TRU_MTR	Master Trigger Register
TRU_ERRADDR	Error Address Register
TRU_STAT	Status Information Register
TRU_GCTL	Global Control Register

Slave Select Register

The TRU slave select registers (TRU_SSRn) each provide slave selection and register locking.

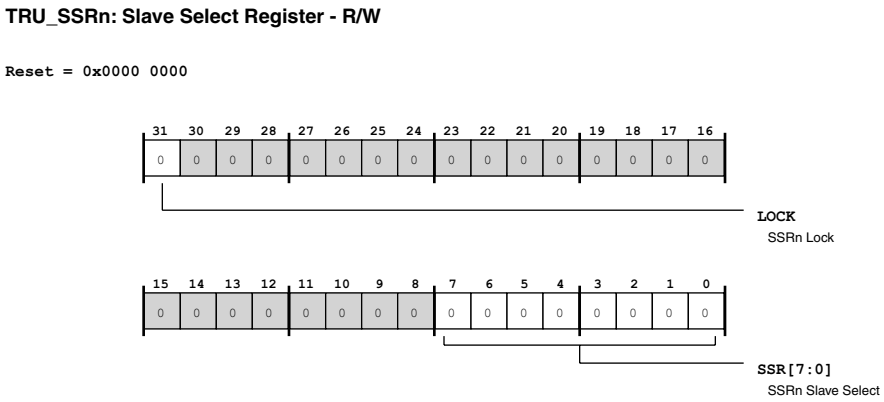


Figure 8-2: TRU_SSRn Register Diagram

Table 8-6: TRU_SSRn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	SSRn Lock. If the global lock is enabled (SPU_CTL_GLCK bit =1) and the TRU_SSRn.LOCK bit is enabled, the TRU_SSRn register is read only.
7:0 (R/W)	SSR	SSRn Slave Select. The TRU_SSRn register selects the trigger master ID to which the trigger slave responds. For example, when a TRU_SSRn 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.

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.

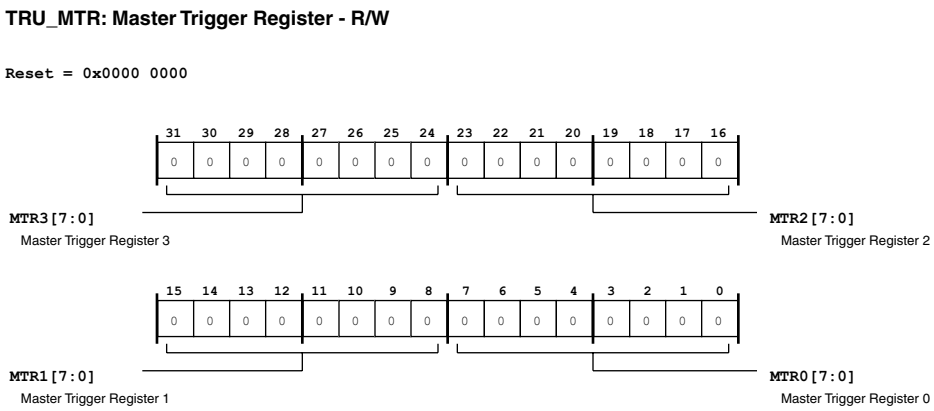


Figure 8-3: TRU_MTR Register Diagram

Table 8-7: TRU_MTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	MTR3	Master Trigger Register 3.
23:16 (R/W)	MTR2	Master Trigger Register 2.
15:8 (R/W)	MTR1	Master Trigger Register 1.
7:0 (R/W)	MTR0	Master Trigger Register 0.

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.

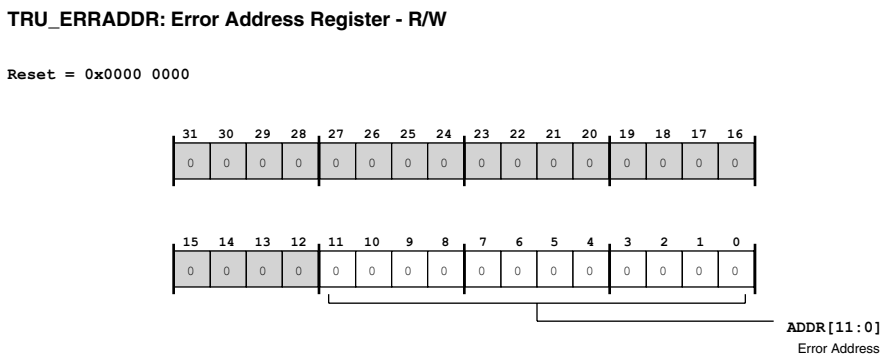


Figure 8-4: TRU_ERRADDR Register Diagram

Table 8-8: TRU_ERRADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/W)	ADDR	Error Address. The TRU_ERRADDR.ADDR holds the address from the memory mapped register access generating an access error of TRU registers. These errors occur on access to the TRU_SSRn or TRU_MTR registers when these registers are locked or on access to an invalid address. See the TRU_SSRn and TRU_MTR register descriptions for more information about locking. 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 (TRU_STAT.LWERR and TRU_STAT.ADDRERR) in the TRU_STAT register must be cleared.

Status Information Register

The TRU status register (TRU_STAT) contains the status of TRU_MTR and TRU_SSRn register writes and status of bus read/write errors.

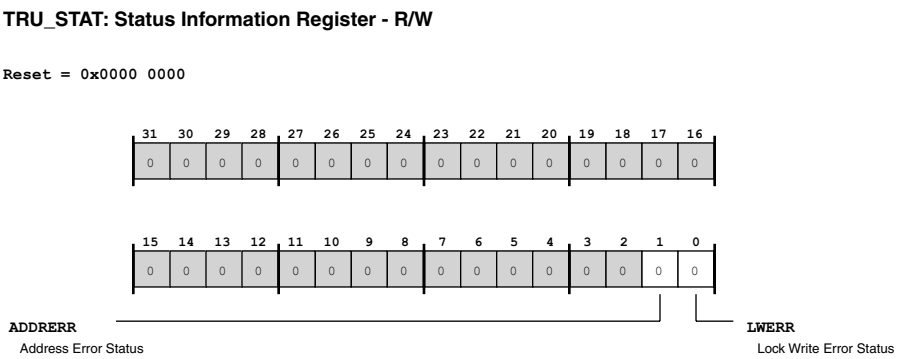


Figure 8-5: TRU_STAT Register Diagram

Table 8-9: TRU_STAT Register Fields

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

Global Control Register

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

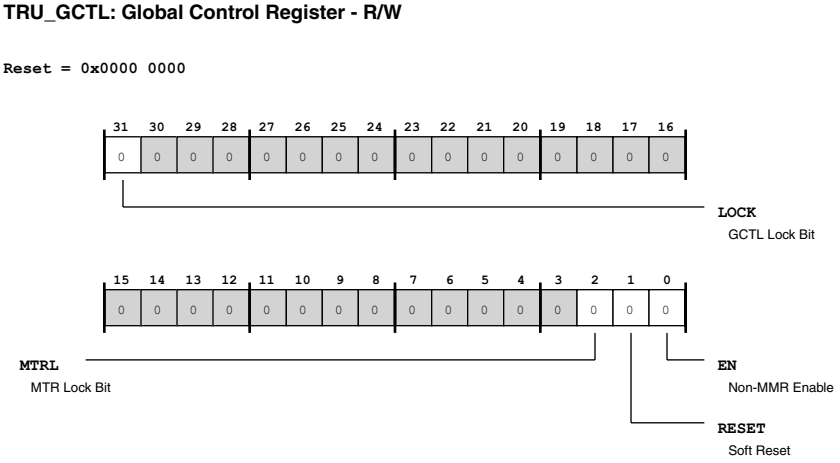


Figure 8-6: TRU_GCTL Register Diagram

Table 8-10: TRU_GCTL Register Fields

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

9 Static Memory Controller (SMC)

The static memory controller is a protocol converter and data transfer interface between the internal processor bus and the external L3 memory. It provides a glueless interface to a variety of external memories and peripheral devices, including SRAM, ROM, EPROM, NOR flash memory, and FPGA/ASIC devices.

The SMC acts as an SCB slave, and accesses to the SMC are arbitrated by the processor SCB interconnect fabric. On the chip boundary, the SMC is connected to an address bus, a data bus, and memory control signal pins (such as read, write, output enable, and memory select lines).

SMC Features

SMC features include:

- 16-bit I/O width
- Provides flexible timing control through extended timing parameters
- Supports asynchronous access extension (SMC_ARDY pin)
- Supports 8-bit data masking writes
- Supports bus request/grant arbitration
- Supports burst read with programmable burst length of 4, 8 and 16

SMC Functional Description

The SMC contains memory-mapped registers that control the access characteristics for each asynchronous memory bank. Different banks can be programmed in different modes, independently controlled using the functional and cycle time bit settings for each bank.

The SMC_GCTL register controls the bus grant feature of the controller. There is only one programmable bit in this register which if set (=1) disables the bus grant feature of SMC. It is cleared at reset which means that bus grant is enabled by default.

The SMC_GSTAT register indicates the status of the $\overline{\text{SMC\_BG}}$, $\overline{\text{SMC\_BR}}$, and $\overline{\text{SMC\_BGH}}$ pins. There are three bits in this register to reflect the status of these pins.

The SMC provides separate sets of registers, SMC_B0CTL – SMC_B3CTL(control), SMC_B0TIM – SMC_B3TIM(timing) and SMC_B0ETIM – SMC_B3ETIM(extended timing) to control the mode and timing charac-

teristic of each bank independently. The control registers contain bits for enabling the bank and bits for selecting mode of operation.

The control registers also include bits to configure use of SMC_ARDY feature and bits for flash page size.

The control registers also contain bits to control the type of bank select control signal. External FIFO devices often do not have a separate chip select pin. As a result, for a read, the FIFO's output enable ($\overline{\text{SMC\_AOE}}$) pin must be connected to the OR (negative AND) of the $\overline{\text{SMC\_AMS0}}$ and the $\overline{\text{SMC\_ARE}}$. Similarly, the write case requires an OR between $\overline{\text{SMC\_AMS0}}$ and $\overline{\text{SMC\_AWE}}$. The SMC provides this function so that an external OR gate is not required. The appropriate AMS function can be selected for each memory bank region using the SMC_BOCTL.SELCTRL bits.

Finally, the control registers also contain bits for synchronous burst and selecting NOR clock frequency.

The following sections provide additional functional descriptions of the SMC.

- *SMC Definitions*
- *SMC Architectural Concepts*

ADSP-BF60x SMC Register List

The static memory controller SMC is a protocol converter and data transfer interface between the internal processor bus and the external L3 memory. The SMC acts as a bus slave and accesses to SMC are arbitrated by the module's system crossbar. On the chip boundary, the SMC is connected to a 25 bit external memory address bus, a 16-bit data bus and memory control signal pins (read, write) including 4 chip selects. This memory interface can support 64MB of external memory connected to 4 different banks, each bank being controlled by the chip select signal. A set of registers govern SMC operations. For more information on SMC functionality, see the SMC register descriptions.

Table 9-1: ADSP-BF60x SMC Register List

Name	Description
SMC_GCTL	Grant Control Register
SMC_GSTAT	Grant Status Register
SMC_B0CTL	Bank 0 Control Register
SMC_B0TIM	Bank 0 Timing Register
SMC_B0ETIM	Bank 0 Extended Timing Register
SMC_B1CTL	Bank 1 Control Register
SMC_B1TIM	Bank 1 Timing Register

Table 9-1: ADSP-BF60x SMC Register List (Continued)

Name	Description
SMC_B1ETIM	Bank 1 Extended Timing Register
SMC_B2CTL	Bank 2 Control Register
SMC_B2TIM	Bank 2 Timing Register
SMC_B2ETIM	Bank 2 Extended Timing Register
SMC_B3CTL	Bank 3 Control Register
SMC_B3TIM	Bank 3 Timing Register
SMC_B3ETIM	Bank 3 Extended Timing Register

SMC Definitions

The timing registers contain bits to program the setup time, hold time and access time for read and write access to each bank separately. The SMC allows for totally different setup/hold/access times for reads and writes. The SMC_B0TIM – SMC_B3TIMregisters control the timing characteristics of the asynchronous memory interface using the following parameter definitions. Each of these parameters can be programmed in terms of *SCLK* clock cycles.

Read setup time

The time between the beginning of a memory cycle ($\overline{\text{SMC_AMSO}}$low) and the read-enable assertion ($\overline{\text{SMC_ARE}}$low).

Read hold time

The time between read-enable deassertion ($\overline{\text{SMC_ARE}}$high) and the end of the memory cycle ($\overline{\text{SMC_AMSO}}$high).

Read access

The time between read-enable assertion ($\overline{\text{SMC_ARE}}$low) and deassertion ($\overline{\text{SMC_ARE}}$high).

Write setup time

The time between the beginning of a memory cycle ($\overline{\text{SMC_AMSO}}$low) and the write-enable assertion ($\overline{\text{SMC_AWE}}$low).

Write hold time

The time between write-enable deassertion ($\overline{\text{SMC_AWE}}$high) and the end of the memory cycle ($\overline{\text{SMC_AMSO}}$high).

Write access

The time between write-enable assertion ($\overline{\text{SMC_AWE}}$low) and deassertion ($\overline{\text{SMC_AWE}}$high).

The SMC provides another register for defining additional timing characteristics of control signals by programming the extended timing registers SMC_B0TIM – SMC_B3TIM. These registers contain bits to program following timing characteristics.

Pre-setup time

The number of cycles $\overline{\text{SMC\_AMSO}}$ is asserted before $\overline{\text{SMC\_AOE}}$ is asserted.

Pre-access time

The number of cycles inserted after $\overline{\text{SMC\_AOE}}$/SMC_NORDVis de-asserted, before $\overline{\text{SMC\_ARE}}$ is asserted for the next access.

Memory idle time

The number of bus idle cycles between $\overline{\text{SMC\_AMSO}}$ de-asserting edge and next asserting edge.

Memory transition time

The number of bus idle cycles extending the Idle time cycles in case of the subsequent access has a different data direction or is to different bank.

Additional useful definitions are provided below.

Bus contention

State of the bus in which more than one device on the bus attempts to place values on the bus at the same time. For more information see *Avoiding Bus Contention*.

ARDY signal

The SMC_ARDY signal is used to insert wait states for slower asynchronous memories. There is no upper limit to how many wait states the ARDY signal can enter. As long as its held, the processor waits for the access to the asynchronous memory. Once asserted, the processor accesses the memory according to the timing diagrams. For more information see *ARDY Input Control*.

SMC Architectural Concepts

The SMC can support connection to multiple different external banks, with each bank controlled by an $\overline{\text{SMC\_AMSn}}$ chip select signal. Check the processor data sheet for details on the bank address ranges and configurations.

NOTE: The address range allocated to each bank is shown in the processor data sheet. Not all of an enabled memory bank need to be populated.

The processor does not directly support 8-bit accesses to the external memories. So, the SMC address lines start from SMC_A01; there is no SMC_A0pin.

The SMC does indirectly support 8-bit accesses through the additional byte enable signals $\overline{\text{SMC\_ABE0}}$ and $\overline{\text{SMC\_ABE1}}$. Some 16-bit memory systems allow the processor to perform 8-bit reads and writes, which are selected through the $\overline{\text{SMC\_ABE0}}$ and $\overline{\text{SMC\_ABE1}}$ signals.

The byte enable pins are both low during all asynchronous reads and 16-bit asynchronous writes. When an asynchronous write is made to the upper byte of a 16-bit memory, $\overline{\text{SMC\_ABE1}}$=0 and $\overline{\text{SMC\_ABE0}}$=1. When an asynchronous write is made to the lower byte of a 16-bit memory, $\overline{\text{SMC\_ABE1}}$=1 and $\overline{\text{SMC\_ABE0}}$=0.

Avoiding Bus Contention

Bus contention occurs during the time one device is getting off the bus and another is getting on. If the first device is slow to three-state and the second device is quick to drive, the devices contend. Bus contention causes excessive power dissipation and can lead to device failure.

There are two cases where contention can occur.

- In reads followed by writes to the same memory space, the data bus drivers can potentially contend with those of the memory device addressed by the read.
- In back-to-back reads from two different memory spaces, the two memory devices addressed by the two reads can contend at the transition between the two read operations.

To avoid contention, program the turnaround time appropriately in the extended time registers (SMC_B0ETIM – SMC_B3ETIM), setting the number of clock cycles between these types of accesses on a bank-by-bank basis.

The idle time bit (SMC_BOETIM.IT) applies to similar back to back access types on the same bank. The transition time bit (SMC_BOETIM.TT) applies to the SMC_BOETIM.ITbit. For actual turnaround situations, idle time and transition time function in an accumulated fashion. The sequence of access types and times are shown below.

- A write followed by write to same bank – SMC_BOETIM.IT
- A read followed by read to same bank – SMC_BOETIM.IT
- A write followed by read to same bank – SMC_BOETIM.IT + SMC_BOETIM.TT
- A read followed by write to same bank – SMC_BOETIM.IT + SMC_BOETIM.TT
- Any access to a given bank followed by any access to a different bank – SMC_BOETIM.IT + SMC_BOETIM.TT

The reset value of turnaround transition time is 2 cycles. Program the SMC_BOETIM.TTbit to a value either greater than or equal to 2 cycles, depending on memory AC-timing specifications. It is important to be aware that the SMC_BOETIM.TTbit may be programmed to 0 *only* when:

- There are *no* SMC banks programmed to operate in synchronous burst mode.
- There are *either* only read accesses *or* only write accesses possible to external memory devices for the current device configuration/processor operation situation.

ARDY Input Control

Each bank can be programmed to sample the SMC_ARDY input after the read or write access timer has counted down or to ignore this input's signal. If enabled and disabled at the sample window, SMC_ARDY can be used to extend the access time as required.

The SMC_ARDY input is treated as an asynchronous input, however it must reach the desired value (either asserted or deasserted) more than two *SCLK* cycles before the completion of access time (scheduled rising edge of $\overline{\text{SMC\_AWE}}$ or $\overline{\text{SMC\_ARE}}$). This determines whether the access is extended by the assertion of SMC_ARDYor not. Once SMC_ARDY(asserted by the memory device), is sampled high the total delay between SMC_ARDY going high at the pads and $\overline{\text{SMC\_ARE}}$ being de-asserted at the pads can be a maximum of 5 *SCLK*cycles.

Asynchronous SRAM writes are also possible with the SMC_ARDY signal enabled. In asynchronous SRAM writes, the write access is extended beyond the programmed write access cycles depending on the SMC_ARDY signal state. Once SMC_ARDY is sampled asserted, the $\overline{\text{SMC\_AWE}}$ signal is deasserted after 2 *CLKOUT* cycles and the write access ends.

The polarity of SMC_ARDY is programmable on a per-bank basis. Since SMC_ARDY is not sampled until an access is in progress to a bank in which the SMC_ARDY enable is asserted, it does not need to be driven by default. When using flash memory, the WAIT input should be connected to SMC_ARDY.

To avoid stalls in case of erroneous SMC_ARDYbehavior, set the SMC_BOCTL.RDYABTEN bit to enable the SMC_ARDY abort counter. When the abort counter is enabled, it starts counting down as soon as the programmed read/write access cycles expire, and times out (generating an error) if the SMC_ARDY signal is

not sampled as asserted within 64 cycles. This ensures that the processor does not hang if SMC_ARDY is not sampled correctly.

Bus Request and Bus Grant

The SMC can relinquish control of the data and address buses to an external device, using the bus request and bus grant protocol. SMC three states it's memory interface to allow an external controller to access either external asynchronous or synchronous memory parts.

When an external controller requires access to the static memory bus, it asserts the bus request ($\overline{\text{SMC\_BR}}$) signal. If no internal SMC request is pending, and if the SMC is in the IDLE state, the SMC grants the external bus request. A bus grant is initiated by the following.

- Three-stating the data and address buses and the memory control signals.
- Asserting the bus grant ($\overline{\text{SMC\_BG}}$) signal.

When the static memory bus is granted to an external controller, the SMC stalls all accesses, including instruction fetches and data reads/writes, that address the SMC memory space. Bus grant status may be checked through examination of the SMC_GSTAT.BGSTAT bit. This bit can be used by software to check the bus grant status prior to initiating transactions that would be delayed by the external bus grant. When the SMC is ready to perform an access, but is held off because the bus is granted, it asserts the bus grant hang ($\overline{\text{SMC\_BGH}}$) pin. When the external controller releases ($\overline{\text{SMC\_BR}}$), the SMC de-asserts ($\overline{\text{SMC\_BG}}$) and continues servicing access requests.

Bank-Off Bus Grant

The SMC bus is granted during system reset and after system reset before any one of the SMC banks is enabled. Since all SMC banks are disabled at reset, including bank 0, the SMC bus is granted by default. This bank-off bus grant state does not depend on the status of the external bus request signal. During this bank-off bus grant state, the $\overline{\text{SMC\_BG}}$ signal is asserted low and the $\overline{\text{SMC\_BGH}}$ signal is unchanged since internal requests are not possible when all SMC banks are disabled. The state of SMC_GCTL.BGDISbit has no effect during the bank-off bus grant state.

When an SMC bank is enabled, it transitions from the bank-off bus grant state to the active state, during which bus grant depends on external bus request ($\overline{\text{SMC\_BR}}$), on internal access requests, and on SMC settings (for example the SMC_GCTL.BGDISbit). If after SMC operation all SMC banks are disabled, the SMC transitions to the bank-off bus grant state, during which the SMC bus is granted irrespective of external bus request status.

Bus Request and Bus Grant Protocol Timing

Fig. shows the bus cycles for the bus request and bus grant protocol. The external device bus request signal ($\overline{\text{SMC\_BR}}$) is latched on the rising edge of *SCLK*. If the SMC is in the idle state (no access requests pending), the bus grant ($\overline{\text{SMC\_BG}}$) signal is asserted 3.5 cycles later on the falling edge of *CLKOUT*. The SMC transitions to the bus grant state, three-stating all SMC buses (address, data, memory select).

If an internal access request is issued while the SMC has the bus granted, the bus grant hang (SMC_BGH) signal is asserted 1.5 cycles later on the falling edge of *CLKOUT*.

When the (SMC_BR) signal is de-asserted by the external memory controller, signaling that the external device is relinquishing control of the bus, the SMC latches the state of (SMC_BR) on the rising edge of *SCLK*, and 3.5 cycles later, de-asserts (SMC_BG) and (SMC_BGH) on the falling edge of *CLKOUT*, transitions from the bus grant state into the active state and resumes processing pending access requests.

NOTE: When asserting SMC_BR low the external host must wait for 3.5 more *SYSCLK*cycles before sampling SMC_BG, where a single SMC_BGsample is used to initiate multi-cycle bus use.

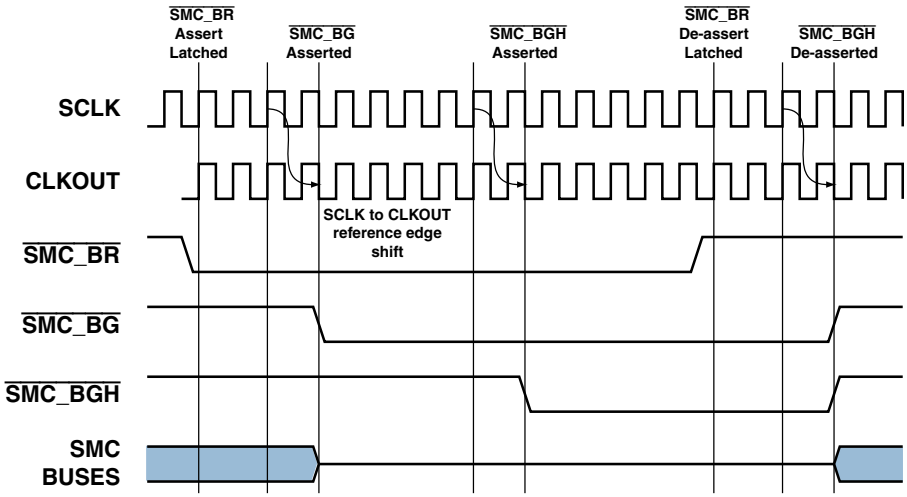


Figure 9-1: SMC Bus Request Bus Grant Protocol

Disabling Bus Grant to External Memory Controllers

By default, the bus grant feature of the SMC is enabled. To configure the SMC to ignore bus requests and the status of the SMC_BR pin, set the SMC_GCTL.BGDISbit.

If the SMC_GCTL.BGDISbit is set while the bus is granted to an external memory controller, the current bus grant status is not affected. The bus continues to be granted until the external memory controller relinquishes control of the bus. At that point, subsequent bus requests are ignored.

SMC Operating Modes

The SMC supports the following operating modes.

- *Asynchronous Flash Mode*
- *Synchronous Burst Mode*
- *Asynchronous Page Mode*

Asynchronous Flash Mode

When the access selected mode is asynchronous flash (SMC_BOCTL.MODE=01), external bank accesses operate exactly the same as in standard asynchronous mode, except for the pin configuration. This mode should be used when accessing burst devices in non-read array modes.

Synchronous Burst Mode

When synchronous mode access is selected (SMC_BOCTL.MODE=11), synchronous reads are enabled. The burst clock frequency can be configured for *SCLK*, *SCLK*/2, *SCLK*/3 or *SCLK*/4. This is the frequency of the clock output and determines the frequency of latching data for subsequent beats of a burst. It does not affect any of the other timing parameters which are still determined by the SMC_BOTIM and SMC_BOETIMtiming registers. During the entire setup time of an access, the $\overline{\text{SMC\_AOE}}$/SMC_NORDVsignal is asserted and the burst clock begins running. The first rising edge of the burst clock signal (SMC_NORCLK) coincides with the fall of *CLKOUT* at the assertion of the $\overline{\text{SMC\_AOE}}$/SMC_NORDVsignal.

Once the address is latched, the initial burst access occurs based on the read access timing for that bank. The strobe time is then extended by a burst clock duration for each subsequent beat of the burst. Any access in the burst may be extended by connecting the flash WAIT signal to the SMC_ARDY signal and the flash device must be configured to deassert the SMC_ARDY signal at the same time that data is valid.

There are cases when the SMC_NORWTsignal is asserted by the flash device in the middle of a read burst (for page crosses). In such cases, read data latching halts until the SMC_NORWTsignal is again deasserted by the flash device, after which data is latched on every rising edge of SMC_NORCLK.

The synchronous read may be burst or single mode, depending on the type of transfer requested. Burst access is only supported for back-to-back reads, such as cache line fills (16 words), 64-bit instruction reads (4 words), and DMA reads. Write accesses in synchronous flash burst mode are processed as simple asynchronous flash writes. This allows easy programming and re-programming of flash configuration registers while the bank is programmed in synchronous burst mode.

Asynchronous Page Mode

When asynchronous page mode access is selected (SMC_BOCTL.MODE=10), asynchronous page reads are enabled. Page sizes of 4, 8 and 16 words are supported. When performing a page mode read, the first access in the page proceeds according to the read access time configured in SMC_BOTIMregister. This opens the page and the subsequent reads in that page have a period equal to the page wait states programmed in the SMC_BOETIMregister. Besides the start of the setup phase, the read address is incremented at the start of every page cycle.

Page mode access is only supported for back-to-back accesses, such as cache line fills (16 words), 64-bit instruction reads (4 words) and DMA reads. Write accesses in asynchronous page mode are treated as simple asynchronous flash write accesses.

SMC Event Control

SMC event control consists of recording status of SMC errors. Accesses to reserved locations and writes to read only registers result in bus errors. Bus errors are translated into internal SCB crossbar errors which in turn get translated into interrupts. To report errors occurring in the slave memory devices (for both this memory interface and the MMR interface as well), the core combines the SCB crossbar response signals to generate a combined error signal indication which is routed to the fault management unit.

SMC Programmable Timing Characteristics

This section describes the programmable timing characteristics for the SMC. Timing relationships depend on the programming of the SMC bank registers, whether initiation is from the core or from DMA, and the sequence of transactions (read followed by read, read followed by write, and others).

NOTE: All memory control, address and data signals are driven out of chip with regard to the falling edge of the *CLKOUT* signal, except for burst clock. The *CLKOUT* signal is *SCLK* on the chip pins (pad delayed).

Asynchronous SRAM Reads and Writes

The following figure shows a basic single write and read operation to an external device with SMC programmed in asynchronous SRAM mode.

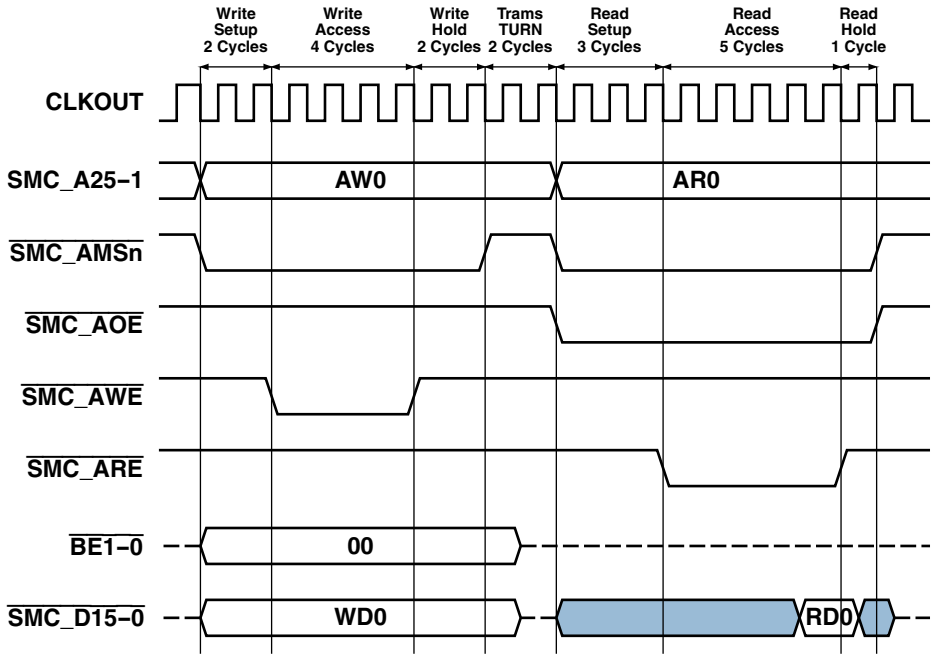


Figure 9-2: Basic Asynchronous SRAM Write Followed by Read

For the current bank, the programmed time cycles are:

- write setup time=2 cycles
- write access time=4 cycles
- write hold time is=2 cycles
- read setup time=3 cycles
- read access time=5 cycles
- read hold time=1 cycle
- turnaround transition time=2 cycles
- idle transition time=0 cycles

The asynchronous SRAM bus cycles proceed as follows.

1. At the start of the write setup period, the chip select signal ($\overline{\text{SMC\_AMSn}}$) for the target bank asserts. The write data (WD0), address (AW0) and byte enables become valid.
2. At the end of the setup phase and at the start of the write access period, the write enable ($\overline{\text{SMC\_AWE}}$) asserts.
3. At the end of the programmed write access, the $\overline{\text{SMC\_AWE}}$ signal de-asserts. The target device is assumed to have captured the write data before $\overline{\text{SMC\_AWE}}$ de-asserts.
4. At the end of the write hold period, the $\overline{\text{SMC\_AWE}}$ signal de-asserts because the pending access is a read access, and the turnaround transition time cycles start. The write data and byte enables become invalid within 1 cycle of the $\overline{\text{SMC\_AMS0}}$ signal de-asserting.
5. At the end of turnaround transition time, the read setup period starts with the assertion of the $\overline{\text{SMC\_AMS0}}$ and $\overline{\text{SMC\_AOE}}$ signals and a new read address (AR0) presented on the address bus.
6. At the start of the read access period, the read enable signal, $\overline{\text{SMC\_ARE}}$ asserts.
7. At the end of the read access period the $\overline{\text{SMC\_ARE}}$ signal de-asserts and the read hold period starts. Read data is latched along with $\overline{\text{SMC\_ARE}}$ de-asserting.
8. At the end of the read hold period, the $\overline{\text{SMC\_AMSn}}$ signal is pulled high and turnaround transition cycles are appended unless there is a pending read request to the same bank.

Asynchronous SRAM Reads with IDLE Transition Cycles Inserted

The following figure shows two consecutive asynchronous SRAM mode reads to the same bank separated by programmed IDLE transition time cycles.

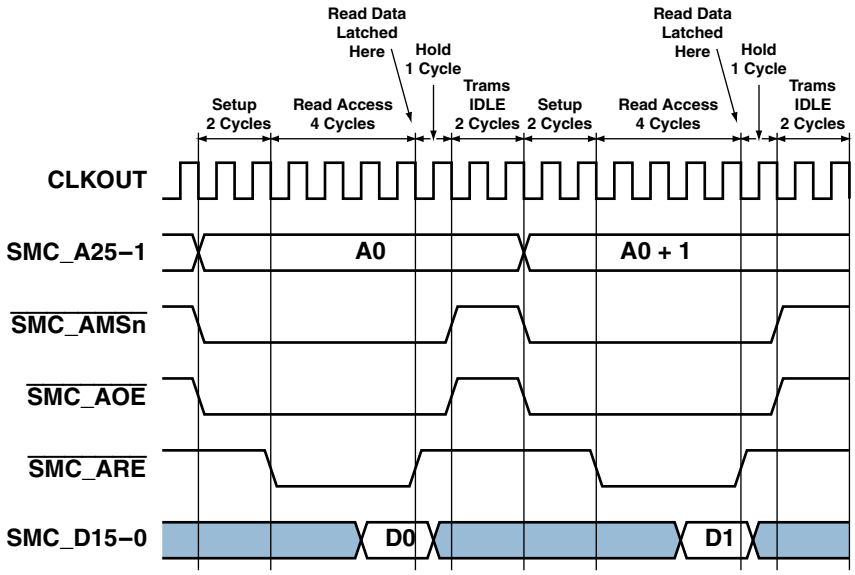


Figure 9-3: Asynchronous SRAM Read with IDLE Transition

Programmed cycle times are:

- SMC_BOTIM.RST=2cycles
- SMC_BOTIM.RAT=4 cycles
- SMC_BOTIM.RHT=1 cycle
- IDLE transition time=2 cycles

At the start of the IDLE transition cycle, $\overline{\text{SMC_AMSn}}$and $\overline{\text{SMC_AOE}}$signal are de-asserted. The setup period of the second read starts at the end of the IDLE transition cycle with the assertion of the $\overline{\text{SMC_AMSn}}$and $\overline{\text{SMC_AOE}}$signals and a new address on the address bus.

High Speed Asynchronous SRAM Read Burst

The following figure shows a high speed asynchronous SRAM read bus cycle. This is typical for SRAM devices with small access times being access through SCB read bursts, especially for boot purposes.

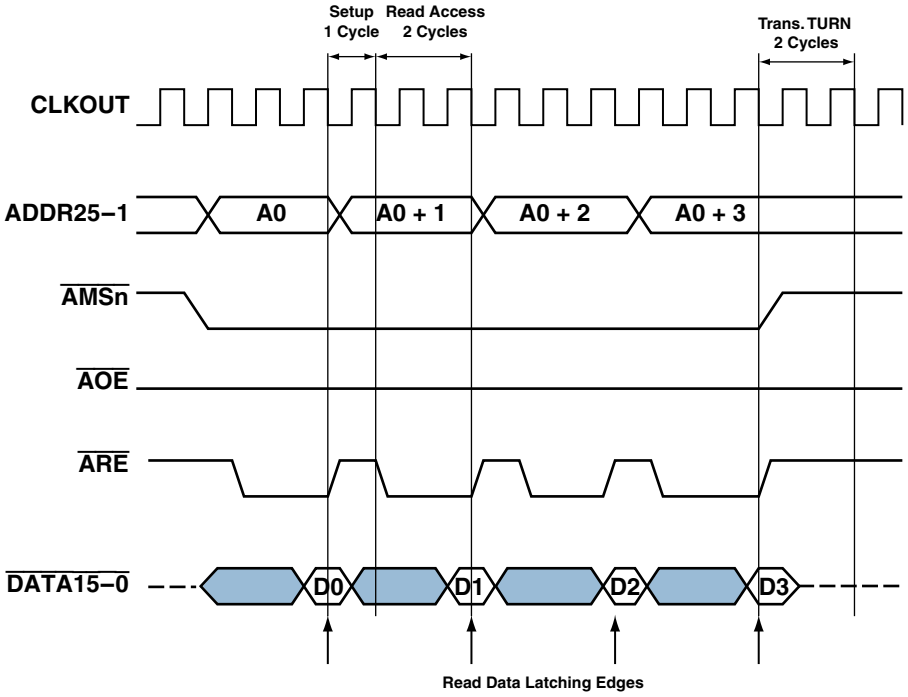


Figure 9-4: Fast Asynchronous SRAM Reads, Burst of Four Word

In this case, the target SMC bank has been programmed with:

- read setup time=1 cycle
- read access time=2 cycles
- read hold time=0
- SMC_BOETIM.PREAT=0
- SMC_BOETIM.PREST=0
- IDLE transition time=0

The SMC_AMSn signal asserts at the start of the setup cycle of the first read out of the burst. Since the hold time and the IDLE transition time have been programmed to 0, the SMC_AMSn signal does not de-assert until the entire set of reads concludes. Only the SMC_ARE signal de-asserts periodically for 1 cycle for the setup period. The read address changes to the next address at the start of each individual setup cycle. Read data words are latched at the end of each individual read access period.

High Speed Asynchronous SRAM Writes

High speed asynchronous SRAM writes are similar to the high speed read accesses. The bus protocol is shown in the following figure for a write burst of 4 words. Here, the write setup time is 1 cycle and the write access time has been programmed to 2 cycles. Write hold time, pre access time, pre-setup time and idle transition time are programmed to 0.

The chip select signal, SMC_AMSn, asserts at the start of the entire write burst and de-asserts only at the end of the last individual write access period. Write address, byte enables and write data for each individual write access are presented onto the bus at the start of each individual write setup cycle. The SMC_AWE signal asserts for the write access period and de-asserts during the setup period for each individual data write.

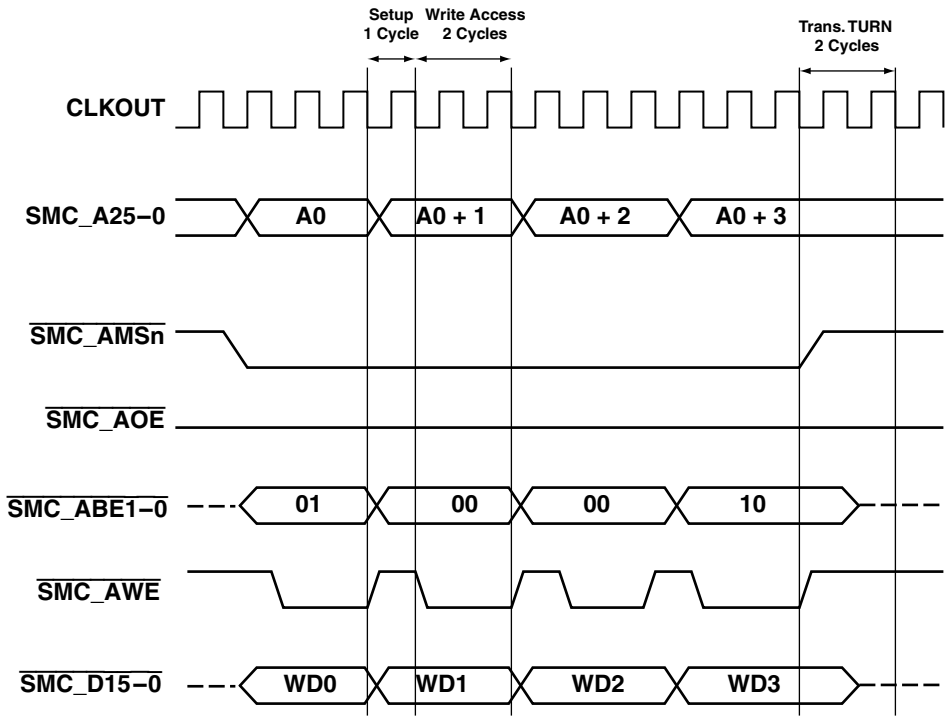


Figure 9-5: Fast Asynchronous SRAM Writes

Asynchronous SRAM Reads with ARDY

The following figure shows an extended asynchronous SRAM read bus cycle with SMC_ARDYenabled.

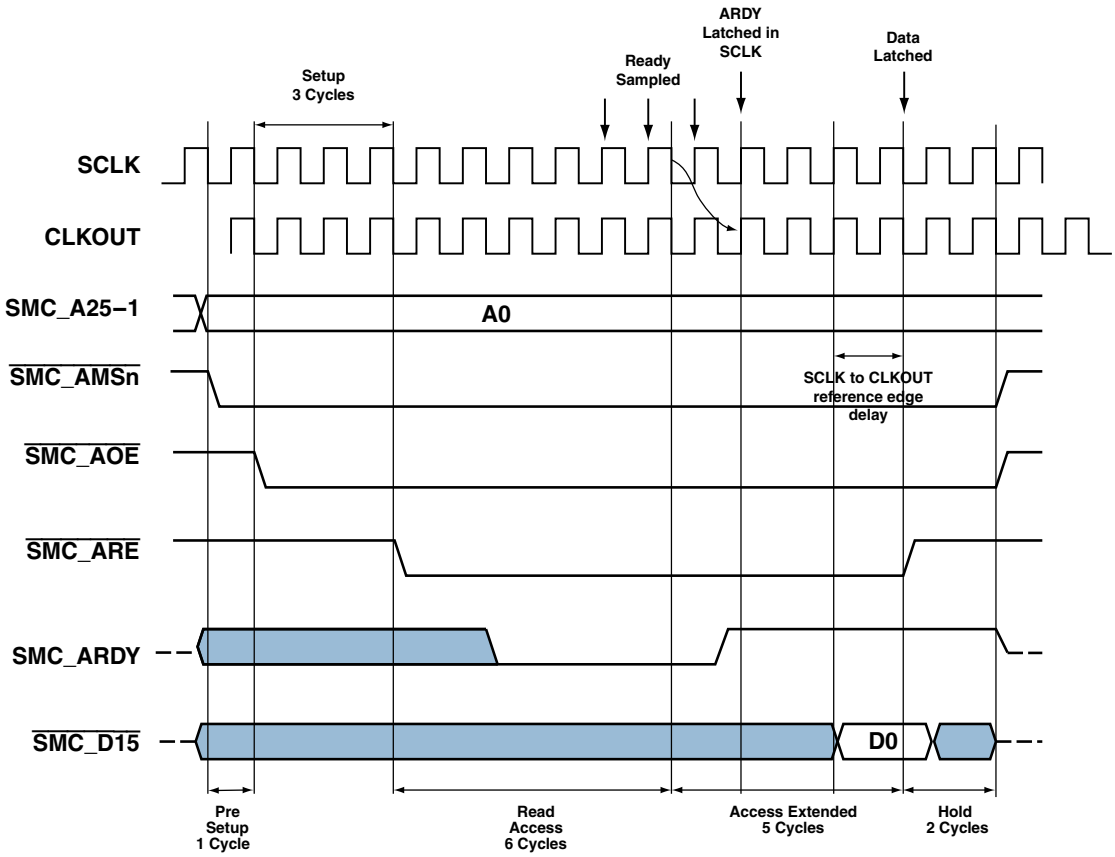


Figure 9-6: Asynchronous SRAM Read with ARDY

The programmed SMC bank control parameters are:

- Pre-Setup Time=1 cycle
- Read Setup Time=3 Cycles
- Read Access Time=6 Cycles
- Read Hold Time=2 Cycles,
- `SMC_BOCTL.RDYPOL`=1 (memory is ready when `SMC_ARDY`=1)

The bus cycles proceed as follows:

- At the start of the pre-setup phase, $\overline{\texttt{SMC\_AMSn}}$ asserts, and read address `SMC_A01` is presented on the address bus.
- At the start of the setup period, $\overline{\texttt{SMC\_AOE}}$ asserts.
- At the start of the read access, $\overline{\texttt{SMC\_ARE}}$ asserts.
- The *CLKOUT* signal is *SCLK* which is driven out of the pads. The *CLKOUT* signal falling edge can be delayed from the internal *SCLK* falling edge. See the data sheet for the specification related to this delay.

All output signals out of the pads, for example $\overline{\text{SMC\_ARE}}$ and $\overline{\text{SMC\_AOE}}$, are driven with regard to the falling edge of *CLKOUT*.

- The SMC starts sampling the SMC_ARDYsignal on every rising edge of internal *SCLK* 2 cycles before the programmed number of read access cycles expires. The read access is extended ($\overline{\text{SMC\_ARE}}$ is kept asserted) until SMC_ARDY is sampled high.
- Once the SMC_ARDY signal (asserted by memory device), is sampled high in *SCLK*, the read signal is pulled off internally in the *SCLK* domain. The total delay between the SMC_ARDY signal going high at the pads and the de-assertion of the $\overline{\text{SMC\_ARE}}$ signal at the pads can be a maximum of 5 *SCLK* cycles.
- Read data is latched at the falling edge of *CLKOUT* on the same edge where $\overline{\text{SMC\_ARE}}$ is deasserted.
- Hold bus cycles start after the $\overline{\text{SMC\_ARE}}$ signal is de-asserted.
- At the end of the hold period, the $\overline{\text{SMC\_AMSn}}$ and $\overline{\text{SMC\_AOE}}$ signals de-assert and the SMC goes into the transition state.

Asynchronous Flash Reads

The following figure illustrates a single asynchronous flash mode read bus cycle.

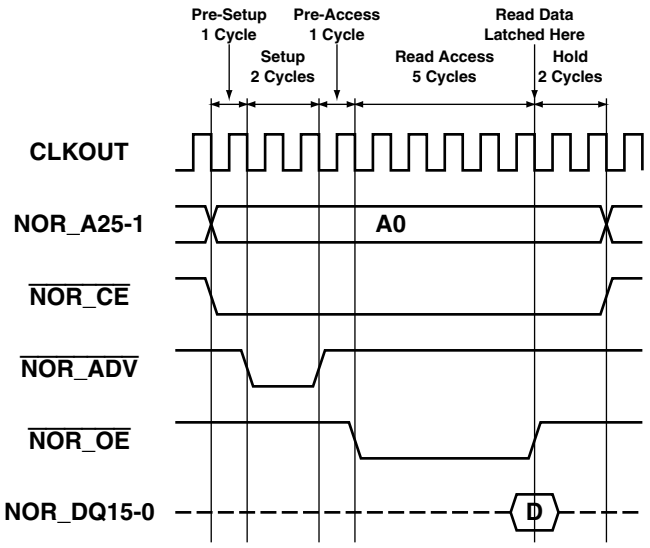


Figure 9-7: Asynchronous Flash Read with Pre-Setup and Pre-Access Cycles

In this case, the target SMC bank has been programmed with:

- pre-setup time=1 cycle
- read setup time=2 cycles
- pre-access time=1 cycle
- read access time=5 cycles

- read hold time=2 cycles.

The read bus cycle is almost identical to the asynchronous SRAM read bus cycle. The only difference is the behavior of the SMC_AOE signal which is used as the flash address valid SMC_NORDV signal. The SMC_NORDV signal asserts at the start of the setup cycle and de-asserts at the end of the setup cycle.

The pre-access cycle inserts 1 cycle gap between the de-assertion of SMC_NORDV and the assertion of the flash read strobe NOR_OE at the start of read access. asynchronous flash reads can also be used with SMC_ARDY enabled for flash devices which use SMC_NORWT in asynchronous mode. In that case, the read bus cycle operation is identical to the asynchronous SRAM with SMC_ARDY enabled except for the SMC_AOE/SMC_NORDV signal behavior.

The following figure shows a 32-bit read access to a flash device in asynchronous mode which is split into two 16-bit external memory accesses. For this bank, read setup and read hold are programmed as 2 cycles whereas the read access time is 5 cycles. Note that the flash device chip select signal (NOR_CE) remains asserted for the entire duration of both read accesses, and is de-asserted at the end of the hold period of the second read access. The SMC_NORDV signal is asserted during the setup phase of both read accesses. Read data is latched at the end of the read access period.

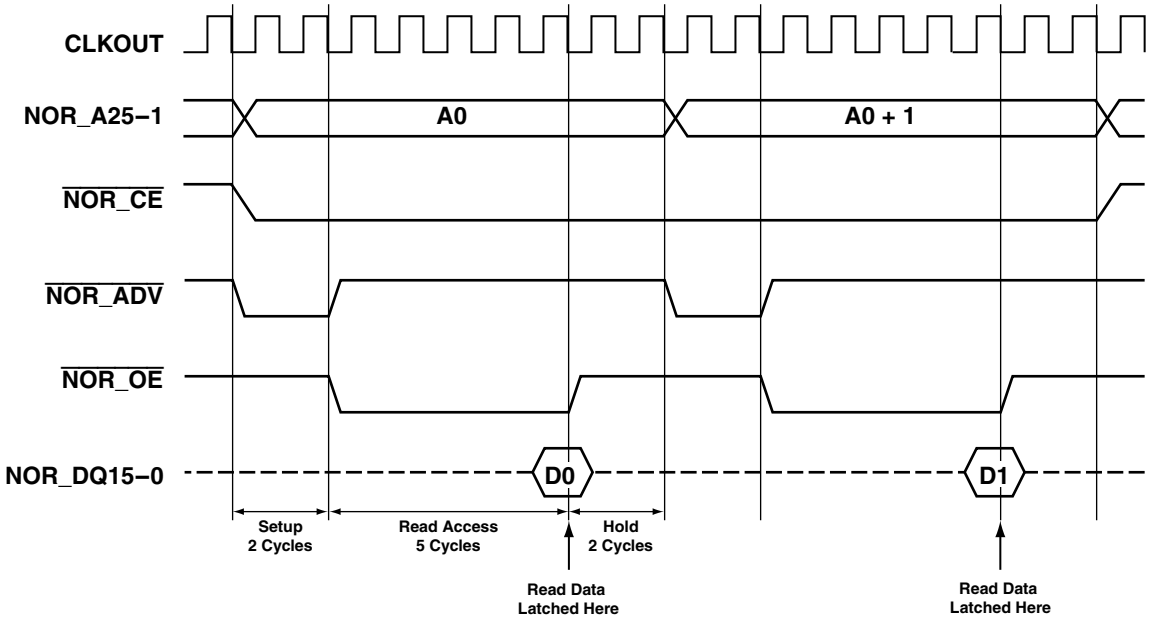


Figure 9-8: 32-bit Asynchronous Flash Read

Asynchronous Flash Writes

The following figure shows a single asynchronous flash write bus cycle.

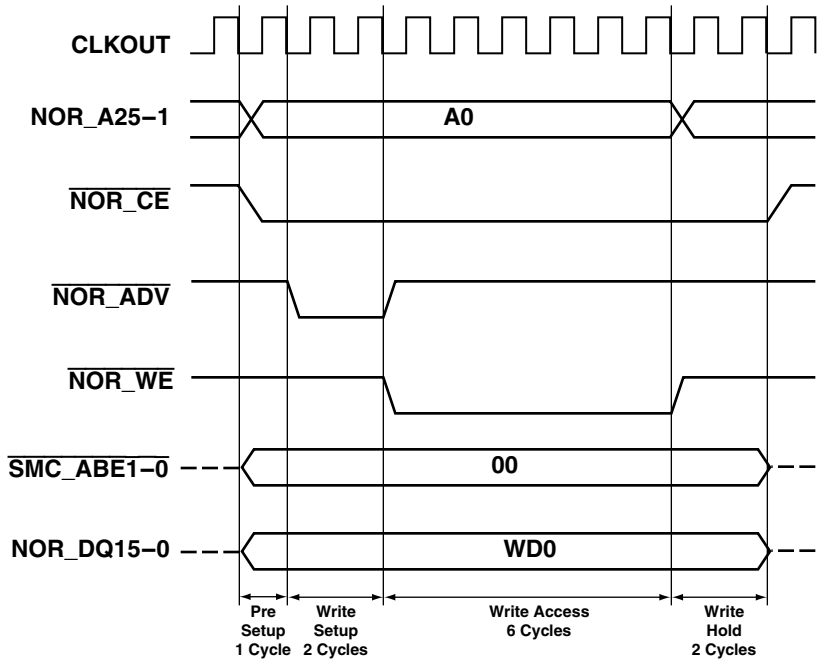


Figure 9-9: Asynchronous Flash Write Operation

For this example, the SMC has been programmed with:

- pre-setup time=1 cycle
- write setup time=2 cycles
- write access time=6 cycles
- write hold time=2 cycles
- pre-access time=0

The asynchronous flash write bus cycle is again almost identical to the asynchronous SRAM write. The SMC_AWE pin is connected to flash write enable signal (NOR_WE). However, in asynchronous flash writes, the SMC_AOE signal is used as the address valid signal (SMC_NORDV) and asserts for the duration of the setup period, unlike in asynchronous SRAM writes where the SMC_AOE signal never asserts.

Asynchronous Flash Page Mode Reads

The following figure shows an asynchronous page mode bus read cycle for a burst of 5 reads which are split into 4 reads followed by a single read.

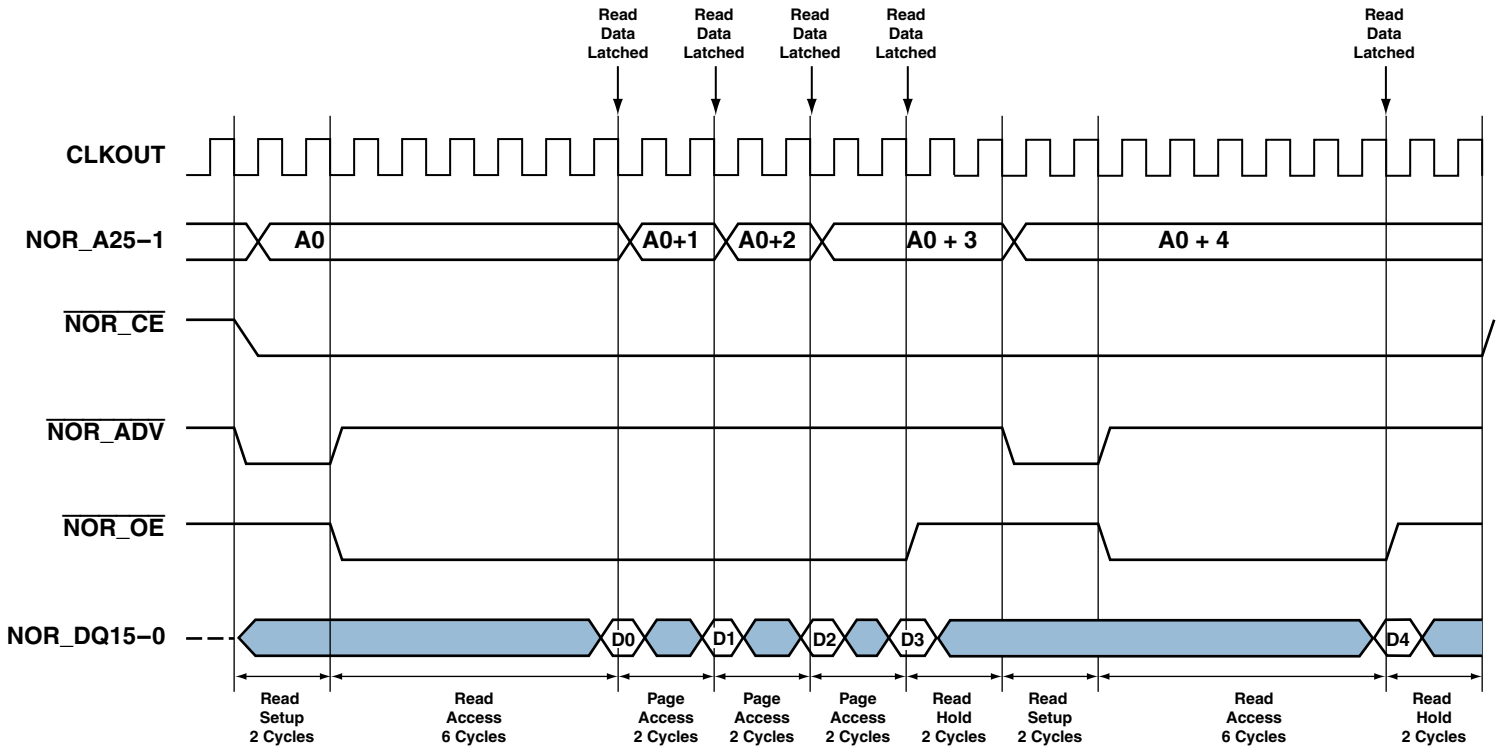


Figure 9-10: Asynchronous Page Mode Read Bus Cycle

The programmed bank parameters are:

- read setup time=2 cycles
- read access time=6 cycles
- page wait=2 cycles
- hold time=2 cycles

The maximum number of read bursts in a total page access depends on the bank `SMC_BOCTL.PGSZ` bits (00=4 words, 01=8 words, 1x=16 words). The first read access is extended for three more page-read cycles whose period is equal to the page wait states. Besides the start of the setup phase, the read address is incremented at the start of every page cycle. Read data is latched with the falling edge of *CLKOUT* the end of the read access period, and also at the end of the page cycles.

Synchronous Burst Mode Reads

The following figure shows a synchronous burst mode read operation for a four word burst. The Xlatency in the flash device has been programmed to three, and the burst clock division value is 2.

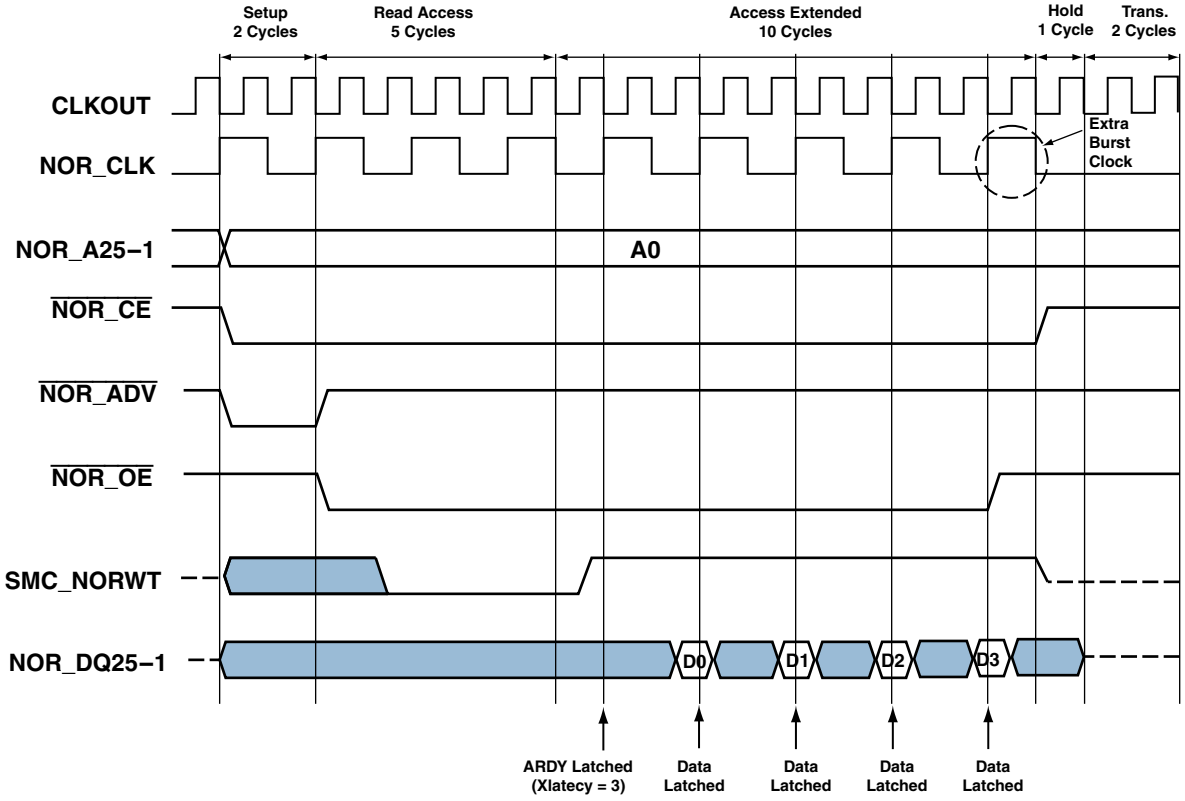


Figure 9-11: Synchronous Burst Mode Read Operation, Burst of 4 Words, DIV = 2, XLAT = 3

The computed read cycle parameters are:

- read setup time=2 cycles
- read access time=5 cycles
- read hold time=1 cycle

The SMC bus cycles proceed as follows:

- At the start of the read setup period, the NOR_CE and SMC_NORDV signals assert and the read address is presented on the NOR_A bus. The first rising edge of the burst clock signal (SMC_NORCLK) coincides with the fall of *CLKOUT* at this edge.
- At the end of the read setup period, which is exactly 1 SMC_NORCLK signal wide, the SMC_NORDV signal de-asserts, and a read access starts with NOR_OE being asserted.
- Flash de-asserts the SMC_NORWT signal after 2 SMC_NORCLK cycles after NOR_OE is asserted, because Xlatency has been fixed at 3. (Here, the SMC_NORWT signal is programmed to active low in flash)
- The read access period ends after 5 *CLKOUT* cycles, after which the SMC_NORWT signal is sampled at every rising edge of SMC_NORCLK.

- After the SMC_NORWT signal has been sampled high, read data words are latched on subsequent rising edges of SMC_NORCLK, and the read access is extended for 10 *CLKOUT* cycles, after which the SMC deasserts NOR_OE, terminating the 4 word burst.
- At the end of the hold period, NOR_CE de-asserts, and transition cycles begin.
- SMC_NORCLK toggle stops with the end of the read access. The last rising edge of SMC_NORCLK occurs before NOR_OE rises.

Asynchronous FIFO Reads and Writes

The following figure shows read bus cycles for an asynchronous FIFO device. The SMC bank has been programmed in asynchronous SRAM mode, with SMC_BOCTL.SELCTRL =01 (SMC_AMSn is OR-ed with SMC_ARE).

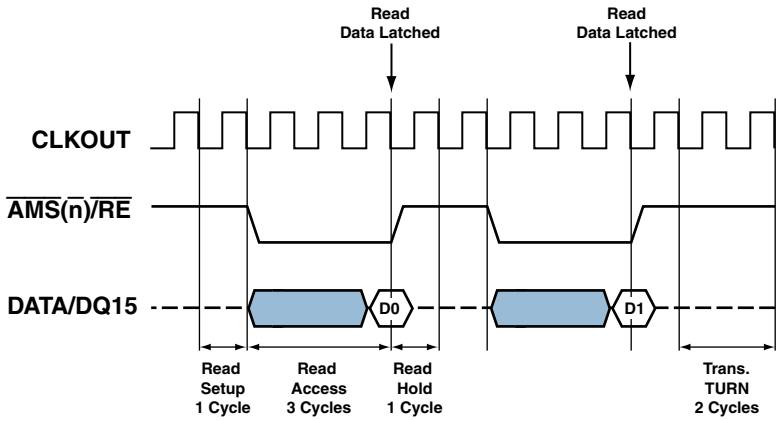


Figure 9-12: Asynchronous FIFO Read Bus Cycles

Other settings are:

- read setup time=1 cycle
- read access time=3 cycles
- read hold time=1 cycle
- idle transition time=0 cycles
- turnaround transition time=2 cycles

The SMC_AMSn signal is connected to the read enable (RE) of the FIFO device, and the data bus is connected to the output data bus (DQ) of the FIFO. The SMC_AMSn signal or the FIFO read strobe asserts only for the duration of the read access. Read data is latched at the falling edge of *CLKOUT* at the end of the read access, when SMC_AMSn is deasserted.

The following figure illustrates write bus cycles for an asynchronous FIFO device. The SMC bank has been programmed in asynchronous SRAM mode, with SMC_BOCTL.SELCTRL = 10 (SMC_AMSn is OR-ed with SMC_AWE). Other settings are:

- write setup time=1 cycle
- write access time=3 cycles
- write hold time=1 cycle
- idle transition time=0
- turnaround transition time=2 cycles

The SMC_AMSn signal is connected to the write enable (WE) of the FIFO device, and data bus is connected to the input data bus (DIN) of the FIFO. The SMC_AMSn signal or the FIFO write strobe asserts only for the duration of the write access. However, write data is asserted at the start of the setup cycle and is taken off at the end of the hold period for each individual write access.

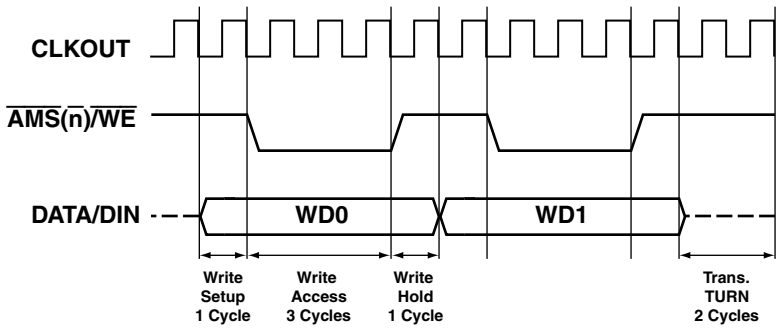


Figure 9-13: Asynchronous FIFO Write Bus Cycles

SMC Programming Model

Following are general guidelines for configuring and enabling the SMC interface. Failure to follow these guidelines can lead to erroneous behavior.

- In asynchronous page mode, the SMC_BOCTL.RDYEN bit should always be 0.
- The ARDY abort counter should be enabled (the SMC_BOCTL.RDYABTEN bit =1) whenever the SMC_ARDY signal is enabled (the SMC_BOCTL.RDYEN is set to 1). Doing so ensures that the interface does not hang due to erroneous SMC_ARDY signal behavior or erroneous sampling of the SMC_ARDY signal.
- Read access time (SMC_BOTIM.RAT), write access time (SMC_BOTIM.WAT), read setup time (SMC_BOTIM.RST), and write setup time (SMC_BOTIM.WST) should not be programmed to zero.
- Page mode wait states (SMC_BOETIM.PGWS) should never be programmed to 0 or 1.
- Program the burst type (SMC_BOCTL.BTYPE) and the page size bits (SMC_BOCTL.PGSZ) to match the configurations of the flash device that is being connected to the SMC interface.

- The SMC_BOCTL.RDYPOL bit should be selected to be the complement of the WAIT polarity that is configured in the flash device.
- In asynchronous SRAM and asynchronous flash modes with SMC_ARDY enabled and where SMC_BOTIM.RHT, SMC_BOTIM.WHT, SMC_BOTIM.RAT, SMC_BOTIM.WAT are the read and write hold and access times and SMC_BOETIM.IT and SMC_BOETIM.TT are the idle and transition times ensure that:
  - SMC_BOTIM.RHT + SMC_BOETIM.IT + SMC_BOETIM.TT>= 2
  - SMC_BOTIM.WHT + SMC_BOETIM.IT + SMC_BOETIM.TT>= 2
  - SMC_BOTIM.RAT>= 5
  - SMC_BOTIM.WAT>= 5

In synchronous burst mode, program:

- SMC_BOTIM.RST = m × (BCLKDIV + 1); where BCLKDIV is the value programmed in the SMC_BOCTL.BCLK bit field, and m = 1, 2, 3, ...
- SMC_BOETIM.PREAT = n × (BCLKDIV + 1); where BCLKDIV is the value programmed in the SMC_BOCTL.BCLK bit field, and n = 0, 1, 2, …
- SMC_BOTIM.RAT = XLAT × (BCLKDIV + 1) + 1 – SMC_BOETIM.PREAT – SMC_BOTIM.RST; where XLAT is the X latency value programmed in the synchronous burst flash device.
- SMC_BOTIM.RHT = BCLKDIV

Also in synchronous burst mode ensure that:

- SMC_BOETIM.IT + SMC_BOETIM.TT>= 2
- SMC_BOCTL.RDYEN should always be 1

ADSP-BF60x SMC Register Descriptions

Static Memory Controller (SMC) contains the following registers.

Table 9-2: ADSP-BF60x SMC Register List

Name	Description
SMC_GCTL	Grant Control Register
SMC_GSTAT	Grant Status Register
SMC_BOCTL	Bank 0 Control Register
SMC_BOTIM	Bank 0 Timing Register

Table 9-2: ADSP-BF60x SMC Register List (Continued)

Name	Description
SMC_B0ETIM	Bank 0 Extended Timing Register
SMC_B1CTL	Bank 1 Control Register
SMC_B1TIM	Bank 1 Timing Register
SMC_B1ETIM	Bank 1 Extended Timing Register
SMC_B2CTL	Bank 2 Control Register
SMC_B2TIM	Bank 2 Timing Register
SMC_B2ETIM	Bank 2 Extended Timing Register
SMC_B3CTL	Bank 3 Control Register
SMC_B3TIM	Bank 3 Timing Register
SMC_B3ETIM	Bank 3 Extended Timing Register

Grant Control Register

The SMC_GCTL register controls SMC's bus request operation.

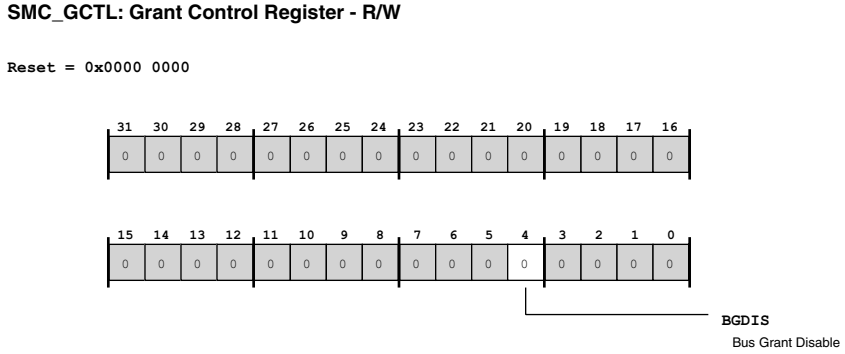


Figure 9-14: SMC_GCTL Register Diagram

Table 9-3: SMC_GCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	BGDIS	Bus Grant Disable. The SMC_GCTL.BGDIS bit disables the SMCs bus request feature, disabling operation of the $\overline{\text{SMC\_BG}}$, $\overline{\text{SMC\_BR}}$, and $\overline{\text{SMC\_BGH}}$ pins.	
		0	Enable
		1	Disable

Grant Status Register

The SMC_GSTAT register indicates the status of the $\overline{\text{SMC\_BG}}$, $\overline{\text{SMC\_BR}}$, and $\overline{\text{SMC\_BGH}}$ pins.

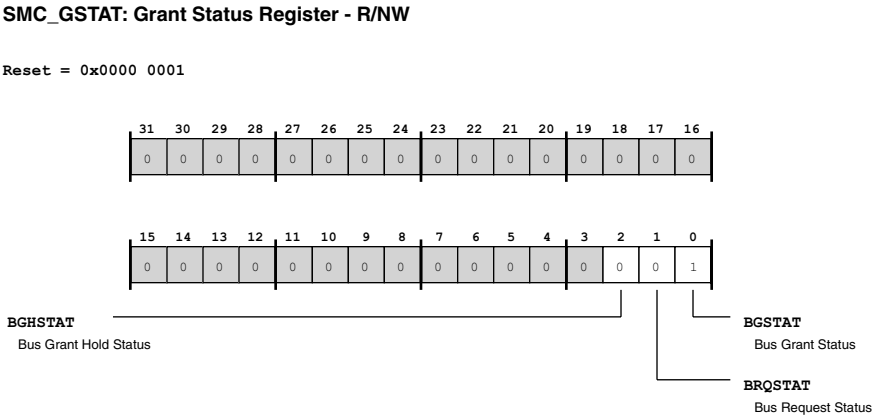


Figure 9-15: SMC_GSTAT Register Diagram

Table 9-4: SMC_GSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/NW)	BGHSTAT	Bus Grant Hold Status. The SMC_GSTAT.BGHSTAT bit indicates the $\overline{\text{SMC\_BGH}}$ pin status. This pin is active when a memory transaction is pending (held) while the bus is granted for another transaction.	
		0	Inactive BGH
		1	Active BGH (transaction pending)

Table 9-4: SMC_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	BRQSTAT	Bus Request Status. The SMC_GSTAT.BRQSTAT bit indicates the $\overline{\text{SMC\_BR}}$ pin status. This pin is active when the bus is requested for a transaction.	
		0	Inactive BR
		1	Active BR (bus requested)
0 (R/NW)	BGSTAT	Bus Grant Status. The SMC_GSTAT.BGSTAT bit indicates the $\overline{\text{SMC\_BG}}$ pin status. This pin is active when bus is granted for a transaction.	
		0	Inactive BG
		1	Active BG (bus granted)

Bank 0 Control Register

The SMC_B0CTL register enables bank 0 accesses and configures the memory access features for this bank.

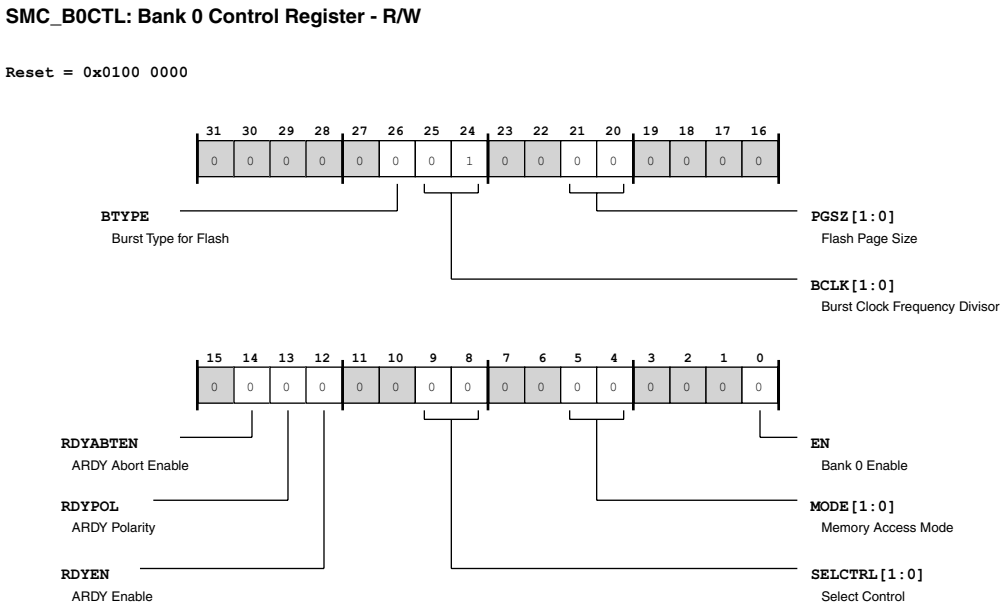


Figure 9-16: SMC_B0CTL Register Diagram

Table 9-5: SMC_B0CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/W)	BTYPE	Burst Type for Flash. The SMC_BOCTL.BTYPE bit selects the burst type that the SMC uses for accesses using sync burst flash protocol.	
		0	Wrap
		1	Sequential
25:24 (R/W)	BCLK	Burst Clock Frequency Divisor. The SMC_BOCTL.BCLK bits select the divisor that the SMC uses to determine the clock frequency for accesses using sync burst flash protocol.	
		0	Burst clock = SCLK ÷ 1
		1	Burst clock = SCLK ÷ 2
		2	Burst clock = SCLK ÷ 3
		3	Burst clock = SCLK ÷ 4
21:20 (R/W)	PGSZ	Flash Page Size. The SMC_BOCTL.PGSZ bits select the flash page size, if page flash or sync burst flash protocol has been enabled (SMC_BOCTL.MODE> 1). Note that the SMC_BOCTL.PGSZ bits must be set to match the flash protocol of the external flash memory device in the system. The typical SMC_BOCTL.PGSZ selection for external devices supporting async flash or async flash page protocols is 4 or 8 words. The typical SMC_BOCTL.PGSZ selection for external devices supporting sync burst flash protocol is 16 words.	
		0	4 words
		1	8 words
		2	16 words
		3	16 words

Table 9-5: SMC_B0CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	RDYABTEN	ARDY Abort Enable. The SMC_BOCTL.RDYABTEN bit enables the abort counter for the SMC_ARDY pin, if enabled (SMC_BOCTL.RDYEN =1). After SMC_BOTIM.RAT or SMC_BOTIM.WAT cycles, the SMC starts sampling the SMC_ARDY pin and starts the abort down counter (if enabled). The abort count is 64 cycles of SCLK. If the SMC detects that SMC_ARDY remains de-asserted when the counter expires, the SMC aborts the access and returns an error response back on the system bus.	
		0	Disable abort counter
		1	Enable abort counter
13 (R/W)	RDYPOL	ARDY Polarity. The SMC_BOCTL.RDYPOL bit selects the polarity (active high or low) for the SMC_ARDY pin, if enabled (SMC_BOCTL.RDYEN =1). When the SMC samples the SMC_ARDY pin in the selective active state, the transaction completes.	
		0	Low active ARDY
		1	High active ARDY
12 (R/W)	RDYEN	ARDY Enable. The SMC_BOCTL.RDYEN bit enables SMC_ARDY pin operation for bank 0 accesses. When enabled, the SMC uses SMC_ARDY (after the access time countdown) to determine completion of access to this memory bank. When disabled, the SMC ignores SMC_ARDY for accesses to this memory bank.	
		0	Disable ARDY
		1	Enable ARDY
9:8 (R/W)	SELCTRL	Select Control. The SMC_BOCTL.SELCTRL bits select the handling of the $\overline{\text{SMC\_AMSn}}$, $\overline{\text{SMC\_ARE}}$, $\overline{\text{SMC\_AOE}}$, and $\overline{\text{SMC\_AWE}}$ pins for memory access control.	
		0	AMS0 only
		1	AMS0 ored with ARE
		2	AMS0 ored with AOE
		3	AMS0 ored with AWE

Table 9-5: SMC_B0CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W)	MODE	Memory Access Mode. The SMC_BOCTL.MODE bits select the protocol the SMC uses for static memory read/write access. Note that the write protocol for async flash, async flash page, and sync burst flash are all similar; only the read protocols differ for these modes.	
		0	Async SRAM protocol
		1	Async flash protocol
		2	Async flash page protocol
		3	Sync burst flash protocol
0 (R/W)	EN	Bank 0 Enable. The SMC_BOCTL.EN bit enables accesses to the memory in bank 0. When this bit is disabled, accesses to bank 0 return an error response.	
		0	Disable access
		1	Enable access

Bank 0 Timing Register

The SMC_B0TIM register configures bank 0 read and write access, setup, and hold timing for this bank. Note that read and write timing configurations are independent and may differ.

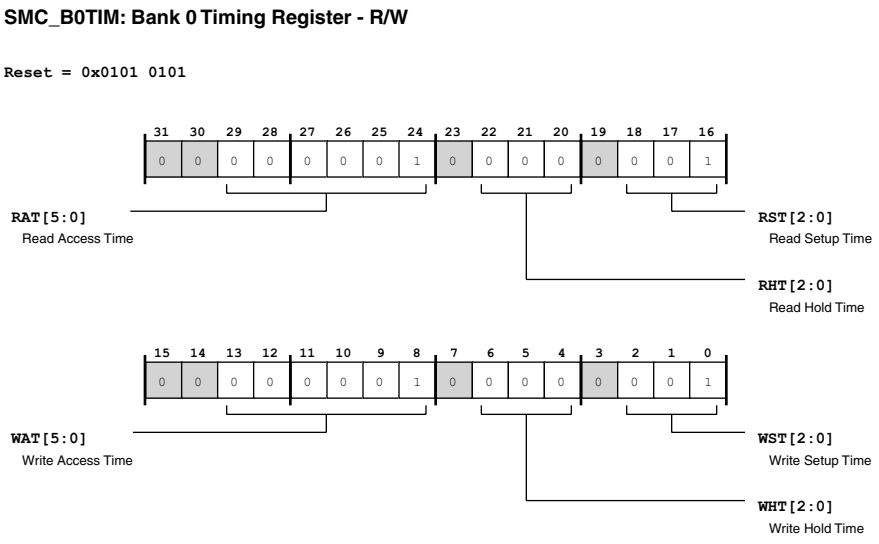


Figure 9-17: SMC_B0TIM Register Diagram

Table 9-6: SMC_B0TIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29:24 (R/W)	RAT	Read Access Time. The SMC_B0TIM.RAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_ARE pin for a read access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
22:20 (R/W)	RHT	Read Hold Time. The SMC_B0TIM.RHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_ARE pin before asserting the SMC_AOE pin for the next access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
18:16 (R/W)	RST	Read Setup Time. The SMC_B0TIM.RST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_ARE pin for an access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
13:8 (R/W)	WAT	Write Access Time. The SMC_B0TIM.WAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_AWE pin for a write access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles

Table 9-6: SMC_B0TIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6:4 (R/W)	WHT	Write Hold Time. The SMC_B0TIM.WHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the $\overline{\text{SMC\_AWE}}$ pin before de-asserting the $\overline{\text{SMC\_AOE}}$ pin for the current access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
2:0 (R/W)	WST	Write Setup Time. The SMC_B0TIM.WST bits select the setup time (in SCLK cycles) that the SMC asserts the $\overline{\text{SMC\_AOE}}$ pin before asserting the $\overline{\text{SMC\_AWE}}$ pin for a write access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Bank 0 Extended Timing Register

The SMC_B0ETIM register configures extensions to access times and idle times, augmenting the setup, hold, and access times configured with the SMC_B0TIM register.

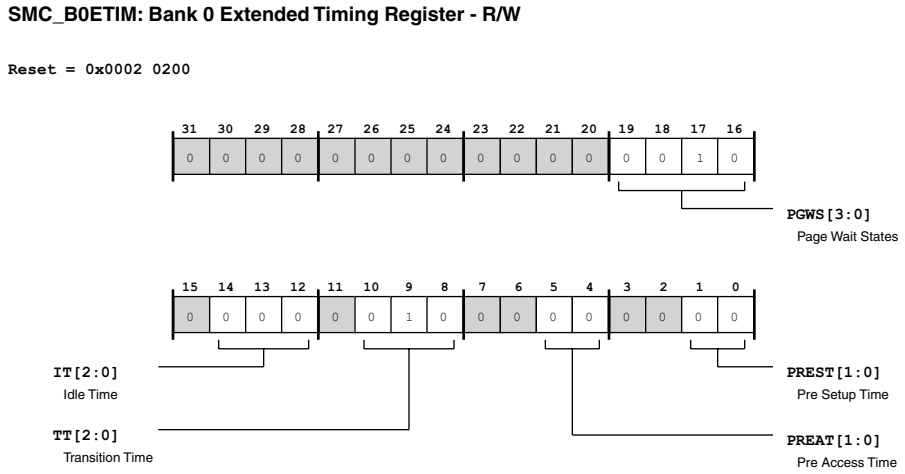


Figure 9-18: SMC_B0ETIM Register Diagram

Table 9-7: SMC_B0ETIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	PGWS	Page Wait States. The SMC_B0ETIM.PGWS bits select a page access extension time (in SCLK cycles) that the SMC waits during read accesses when configured for flash page protocol (SMC_B0CTL.MODE =2). The wait time is from 1 to 15 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycles
		15	15 SCLK clock cycles
14:12 (R/W)	IT	Idle Time. The SMC_B0ETIM.IT bits select a bus idle time (in SCLK cycles) that the SMC waits between de-asserting the $\overline{\text{SMC\_AMSn}}$ pin and asserting the $\overline{\text{SMC\_AMSn}}$ pin for the next access. Note that the SMC_B0ETIM.IT period may be extended using the SMC_B0ETIM.TT selection. The idle time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		7	7 SCLK clock cycles
10:8 (R/W)	TT	Transition Time. The SMC_B0ETIM.TT bits select a bus idle time (in SCLK cycles) that the SMC extends the SMC_B0ETIM.IT to allow for the subsequent access either using a different transfer direction or accessing a different bank. The transition time is from 1 to 7 SCLK cycles.	
		0	No bank transition
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The SMC_B0ETIM.PREAT bits select the pre-access time (in SCLK cycles) that the SMC waits after de-asserting the $\overline{\text{SMC\_AOE}}$/$\overline{\text{ADV}}$ pin before asserting the $\overline{\text{SMC\_ARE}}$/$\overline{\text{SMC\_AWE}}$ pin for the current access. The pre-access time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles

Table 9-7: SMC_B0ETIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	PREST	Pre Setup Time. The SMC_B0ETIM.PREST bits select the pre-setup time (in SCLK cycles) that the SMC asserts the SMC_AMSn pin before asserting the SMC_AOE/ADV pin for an access. The pre-setup time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles

Bank 1 Control Register

The SMC_B1CTL register enables bank 1 accesses and configures the memory access features for this bank.

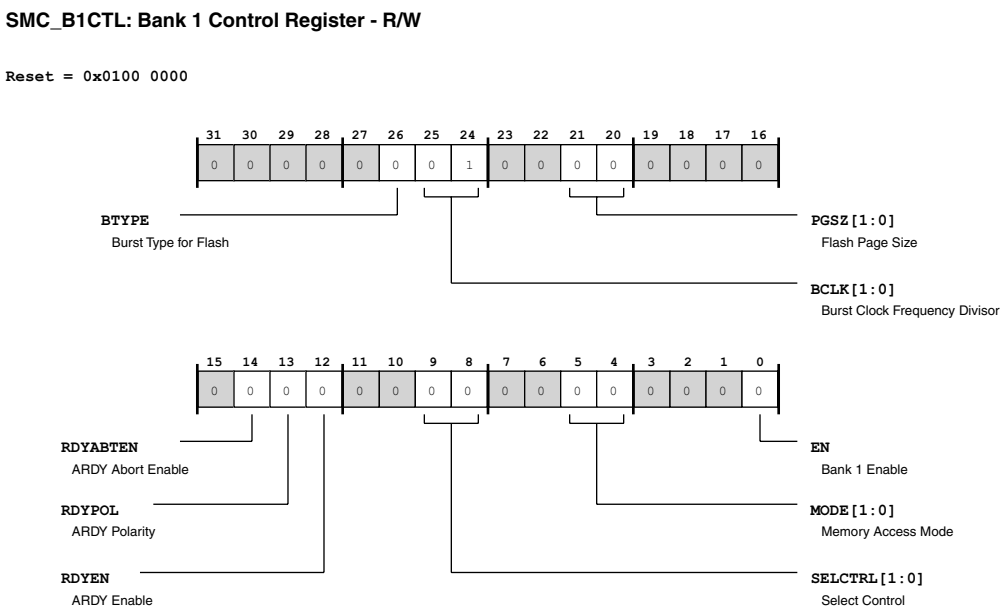


Figure 9-19: SMC_B1CTL Register Diagram

Table 9-8: SMC_B1CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/W)	BTYPE	Burst Type for Flash. The SMC_B1CTL.BTYPE bit selects the burst type that the SMC uses for accesses using sync burst flash protocol.	
		0	Wrap
		1	Sequential
25:24 (R/W)	BCLK	Burst Clock Frequency Divisor. The SMC_B1CTL.BCLK bits select the divisor that the SMC uses to determine the clock frequency for accesses using sync burst flash protocol.	
		0	Burst clock = SCLK ÷ 1
		1	Burst clock = SCLK ÷ 2
		2	Burst clock = SCLK ÷ 3
		3	Burst clock = SCLK ÷ 4
21:20 (R/W)	PGSZ	Flash Page Size. The SMC_B1CTL.PGSZ bits select the flash page size, if page flash or sync burst flash protocol has been enabled (SMC_B1CTL.MODE> 1). Note that the SMC_B1CTL.PGSZ bits must be set to match the flash protocol of the external flash memory device in the system. The typical SMC_B1CTL.PGSZ selection for external devices supporting async flash or async flash page protocols is 4 or 8 words. The typical SMC_B1CTL.PGSZ selection for external devices supporting sync burst flash protocol is 16 words.	
		0	4 words
		1	8 words
		2	16 words
		3	16 words

Table 9-8: SMC_B1CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	RDYABTEN	ARDY Abort Enable. The SMC_B1CTL.RDYABTEN bit enables the abort counter for the SMC_ARDY pin, if enabled (SMC_B1CTL.RDYEN =1). After SMC_B1TIM.RAT or SMC_B1TIM.WAT cycles, the SMC starts sampling the SMC_ARDY pin and starts the abort down counter (if enabled). The abort count is 64 cycles of SCLK. If the SMC detects that SMC_ARDY remains de-asserted when the counter expires, the SMC aborts the access and returns an error response back on the system bus.	
		0	Disable abort counter
		1	Enable abort counter
13 (R/W)	RDYPOL	ARDY Polarity. The SMC_B1CTL.RDYPOL bit selects the polarity (active high or low) for the SMC_ARDY pin, if enabled (SMC_B1CTL.RDYEN =1). When the SMC samples the SMC_ARDY pin in the selective active state, the transaction completes.	
		0	Low active ARDY
		1	High active ARDY
12 (R/W)	RDYEN	ARDY Enable. The SMC_B1CTL.RDYEN bit enables SMC_ARDY pin operation for bank 1 accesses. When enabled, the SMC uses SMC_ARDY (after the access time countdown) to determine completion of access to this memory bank. When disabled, the SMC ignores SMC_ARDY for accesses to this memory bank.	
		0	Disable ARDY
		1	Enable ARDY
9:8 (R/W)	SELCTRL	Select Control. The SMC_B1CTL.SELCTRL bits select the handling of the $\overline{\text{SMC\_AMSn}}$, $\overline{\text{SMC\_ARE}}$, $\overline{\text{SMC\_AOE}}$, and $\overline{\text{SMC\_AWE}}$ pins for memory access control.	
		0	AMS1 only
		1	AMS1 ored with ARE
		2	AMS1 ored with AOE
		3	AMS1 ored with AWE

Table 9-8: SMC_B1CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W)	MODE	Memory Access Mode. The SMC_B1CTL.MODE bits select the protocol the SMC uses for static memory read/write access. Note that the write protocol for async flash, async flash page, and sync burst flash are all similar; only the read protocols differ for these modes.	
		0	Async SRAM protocol
		1	Async flash protocol
		2	Async flash page protocol
		3	Sync burst flash protocol
0 (R/W)	EN	Bank 1 Enable. The SMC_B1CTL.EN bit enables accesses to the memory in bank 1. When this bit is disabled, accesses to bank 1 return an error response.	
		0	Disable access
		1	Enable access

Bank 1 Timing Register

The SMC_B1TIM register configures bank 1 read and write access, setup, and hold timing for this bank. Note that read and write timing configurations are independent and may differ.

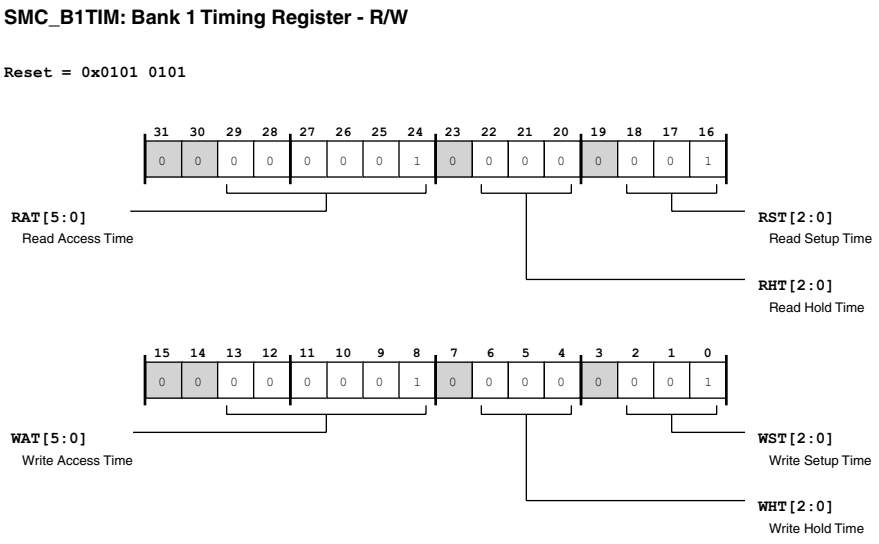


Figure 9-20: SMC_B1TIM Register Diagram

Table 9-9: SMC_B1TIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29:24 (R/W)	RAT	Read Access Time. The SMC_B1TIM.RAT bits select the access time (in SCLK cycles) that the SMC asserts the $\overline{\text{SMC\_ARE}}$ pin for a read access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
22:20 (R/W)	RHT	Read Hold Time. The SMC_B1TIM.RHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the $\overline{\text{SMC\_ARE}}$ pin before asserting the $\overline{\text{SMC\_AOE}}$ pin for the next access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
18:16 (R/W)	RST	Read Setup Time. The SMC_B1TIM.RST bits select the setup time (in SCLK cycles) that the SMC asserts the $\overline{\text{SMC\_AOE}}$ pin before asserting the $\overline{\text{SMC\_ARE}}$ pin for an access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
13:8 (R/W)	WAT	Write Access Time. The SMC_B1TIM.WAT bits select the access time (in SCLK cycles) that the SMC asserts the $\overline{\text{SMC\_AWE}}$ pin for a write access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles

Table 9-9: SMC_B1TIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6:4 (R/W)	WHT	Write Hold Time. The SMC_B1TIM.WHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the $\overline{\text{SMC\_AWE}}$ pin before de-asserting the $\overline{\text{SMC\_AOE}}$ pin for the current access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
2:0 (R/W)	WST	Write Setup Time. The SMC_B1TIM.WST bits select the setup time (in SCLK cycles) that the SMC asserts the $\overline{\text{SMC\_AOE}}$ pin before asserting the $\overline{\text{SMC\_AWE}}$ pin for a write access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Bank 1 Extended Timing Register

The SMC_B1ETIM register configures extensions to access times and idle times, augmenting the setup, hold, and access times configured with the SMC_B1TIM register.

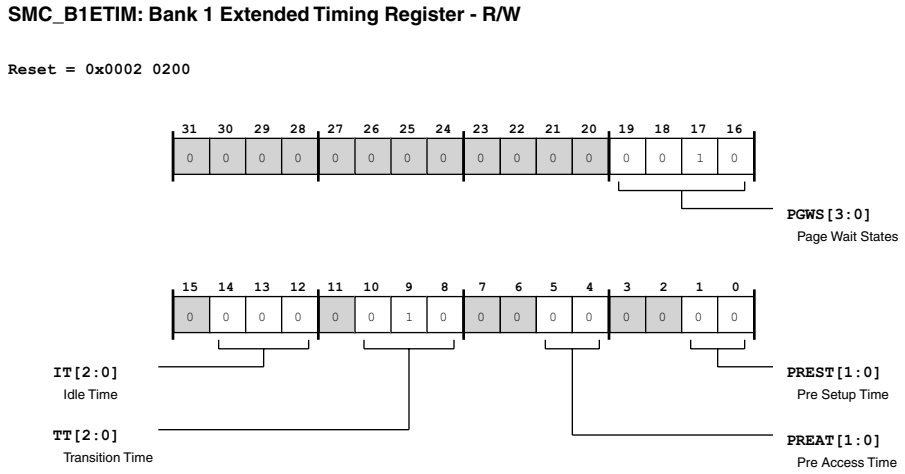


Figure 9-21: SMC_B1ETIM Register Diagram

Table 9-10: SMC_B1ETIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	PGWS	Page Wait States. The SMC_B1ETIM.PGWS bits select a page access extension time (in SCLK cycles) that the SMC waits during read accesses when configured for flash page protocol (SMC_B1CTL.MODE =2). The wait time is from 1 to 15 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycles
		15	15 SCLK clock cycles
14:12 (R/W)	IT	Idle Time. The SMC_B1ETIM.IT bits select a bus idle time (in SCLK cycles) that the SMC waits between de-asserting the $\overline{\text{SMC\_AMSn}}$ pin and asserting the $\overline{\text{SMC\_AMSn}}$ pin for the next access. Note that the SMC_B1ETIM.IT period may be extended using the SMC_B1ETIM.TT selection. The idle time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		7	7 SCLK clock cycles
10:8 (R/W)	TT	Transition Time. The SMC_B1ETIM.TT bits select a bus idle time (in SCLK cycles) that the SMC extends the SMC_B1ETIM.IT to allow for the subsequent access either using a different transfer direction or accessing a different bank. The transition time is from 1 to 7 SCLK cycles.	
		0	No bank transition
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The SMC_B1ETIM.PREAT bits select the pre-access time (in SCLK cycles) that the SMC waits after de-asserting the $\overline{\text{SMC\_AOE}}$/$\overline{\text{ADV}}$ pin before asserting the $\overline{\text{SMC\_ARE}}$/$\overline{\text{SMC\_AWE}}$ pin for the current access. The pre-access time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles

Table 9-10: SMC_B1ETIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	PREST	Pre Setup Time. The SMC_B1ETIM.PREST bits select the pre-setup time (in SCLK cycles) that the SMC asserts the SMC_AMSn pin before asserting the SMC_AOE/ADV pin for an access. The pre-setup time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles

Bank 2 Control Register

The SMC_B2CTL register enables bank 2 accesses and configures the memory access features for this bank.

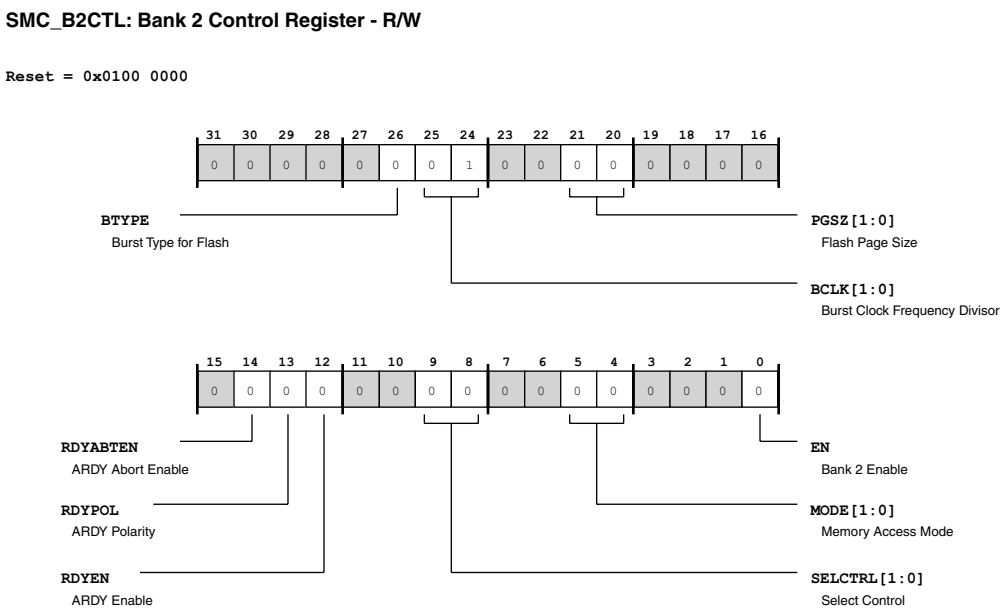


Figure 9-22: SMC_B2CTL Register Diagram

Table 9-11: SMC_B2CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/W)	BTYPE	Burst Type for Flash. The SMC_B2CTL.BTYPE bit selects the burst type that the SMC uses for accesses using sync burst flash protocol.	
		0	Wrap
		1	Sequential
25:24 (R/W)	BCLK	Burst Clock Frequency Divisor. The SMC_B2CTL.BCLK bits select the divisor that the SMC uses to determine the clock frequency for accesses using sync burst flash protocol.	
		0	Burst clock = SCLK ÷ 1
		1	Burst clock = SCLK ÷ 2
		2	Burst clock = SCLK ÷ 3
		3	Burst clock = SCLK ÷ 4
21:20 (R/W)	PGSZ	Flash Page Size. The SMC_B2CTL.PGSZ bits select the flash page size, if page flash or sync burst flash protocol has been enabled (SMC_B2CTL.MODE> 1). Note that the SMC_B2CTL.PGSZ bits must be set to match the flash protocol of the external flash memory device in the system. The typical SMC_B2CTL.PGSZ selection for external devices supporting async flash or async flash page protocols is 4 or 8 words. The typical SMC_B2CTL.PGSZ selection for external devices supporting sync burst flash protocol is 16 words.	
		0	4 words
		1	8 words
		2	16 words
		3	16 words

Table 9-11: SMC_B2CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	RDYABTEN	ARDY Abort Enable. The SMC_B2CTL.RDYABTEN bit enables the abort counter for the SMC_ARDY pin, if enabled (SMC_B2CTL.RDYEN =1). After SMC_B2TIM.RAT or SMC_B2TIM.WAT cycles, the SMC starts sampling the SMC_ARDY pin and starts the abort down counter (if enabled). The abort count is 64 cycles of SCLK. If the SMC detects that SMC_ARDY remains de-asserted when the counter expires, the SMC aborts the access and returns an error response back on the system bus.	
		0	Disable abort counter
		1	Enable abort counter
13 (R/W)	RDYPOL	ARDY Polarity. The SMC_B2CTL.RDYPOL bit selects the polarity (active high or low) for the SMC_ARDY pin, if enabled (SMC_B2CTL.RDYEN =1). When the SMC samples the SMC_ARDY pin in the selective active state, the transaction completes.	
		0	Low active ARDY
		1	High active ARDY
12 (R/W)	RDYEN	ARDY Enable. The SMC_B2CTL.RDYEN bit enables SMC_ARDY pin operation for bank 2 accesses. When enabled, the SMC uses SMC_ARDY (after the access time countdown) to determine completion of access to this memory bank. When disabled, the SMC ignores SMC_ARDY for accesses to this memory bank.	
		0	Disable ARDY
		1	Enable ARDY
9:8 (R/W)	SELCTRL	Select Control. The SMC_B2CTL.SELCTRL bits select the handling of the $\overline{\text{SMC\_AMSn}}$, $\overline{\text{SMC\_ARE}}$, $\overline{\text{SMC\_AOE}}$, and $\overline{\text{SMC\_AWE}}$ pins for memory access control.	
		0	AMS2 only
		1	AMS2 ored with ARE
		2	AMS2 ored with AOE
		3	AMS2 ored with AWE

Table 9-11: SMC_B2CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W)	MODE	Memory Access Mode. The SMC_B2CTL.MODE bits select the protocol the SMC uses for static memory read/write access. Note that the write protocol for async flash, async flash page, and sync burst flash are all similar; only the read protocols differ for these modes.	
		0	Async SRAM protocol
		1	Async flash protocol
		2	Async flash page protocol
		3	Sync burst flash protocol
0 (R/W)	EN	Bank 2 Enable. The SMC_B2CTL.EN bit enables accesses to the memory in bank 2. When this bit is disabled, accesses to bank 2 return an error response.	
		0	Disable access
		1	Enable access

Bank 2 Timing Register

The SMC_B2TIM register configures bank 2 read and write access, setup, and hold timing for this bank. Note that read and write timing configurations are independent and may differ.

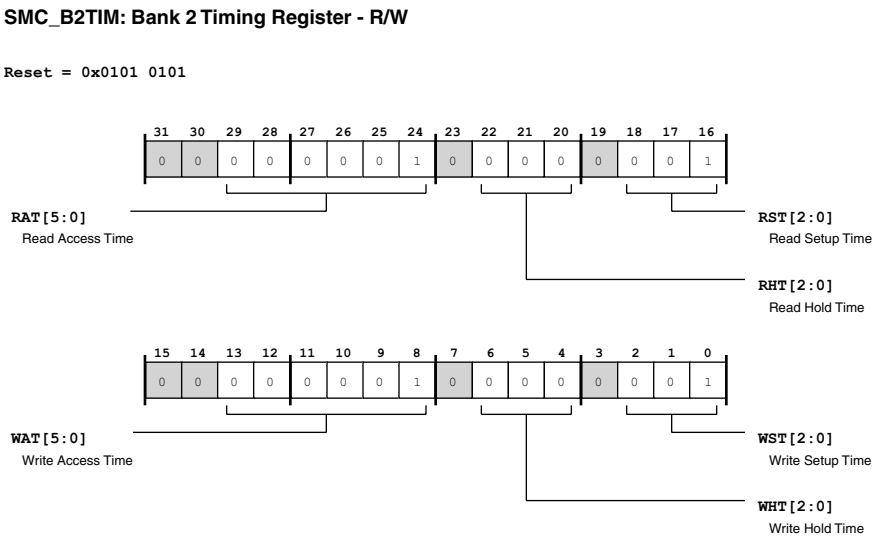


Figure 9-23: SMC_B2TIM Register Diagram

Table 9-12: SMC_B2TIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29:24 (R/W)	RAT	Read Access Time. The SMC_B2TIM.RAT bits select the access time (in SCLK cycles) that the SMC asserts the $\overline{\text{SMC\_ARE}}$ pin for a read access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
22:20 (R/W)	RHT	Read Hold Time. The SMC_B2TIM.RHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the $\overline{\text{SMC\_ARE}}$ pin before asserting the $\overline{\text{SMC\_AOE}}$ pin for the next access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
18:16 (R/W)	RST	Read Setup Time. The SMC_B2TIM.RST bits select the setup time (in SCLK cycles) that the SMC asserts the $\overline{\text{SMC\_AOE}}$ pin before asserting the $\overline{\text{SMC\_ARE}}$ pin for an access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
13:8 (R/W)	WAT	Write Access Time. The SMC_B2TIM.WAT bits select the access time (in SCLK cycles) that the SMC asserts the $\overline{\text{SMC\_AWE}}$ pin for a write access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles

Table 9-12: SMC_B2TIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6:4 (R/W)	WHT	Write Hold Time. The SMC_B2TIM.WHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the $\overline{\text{SMC\_AWE}}$ pin before de-asserting the $\overline{\text{SMC\_AOE}}$ pin for the current access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
2:0 (R/W)	WST	Write Setup Time. The SMC_B2TIM.WST bits select the setup time (in SCLK cycles) that the SMC asserts the $\overline{\text{SMC\_AOE}}$ pin before asserting the $\overline{\text{SMC\_AWE}}$ pin for a write access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Bank 2 Extended Timing Register

The SMC_B2ETIM register configures extensions to access times and idle times, augmenting the setup, hold, and access times configured with the SMC_B2TIM register.

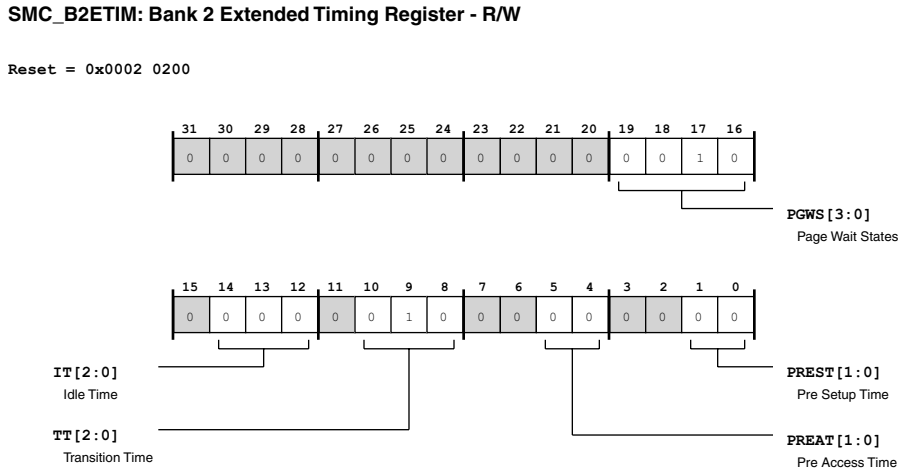


Figure 9-24: SMC_B2ETIM Register Diagram

Table 9-13: SMC_B2ETIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	PGWS	Page Wait States. The SMC_B2ETIM.PGWS bits select a page access extension time (in SCLK cycles) that the SMC waits during read accesses when configured for flash page protocol (SMC_B2CTL.MODE =2). The wait time is from 1 to 15 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycles
		15	15 SCLK clock cycles
14:12 (R/W)	IT	Idle Time. The SMC_B2ETIM.IT bits select a bus idle time (in SCLK cycles) that the SMC waits between de-asserting the $\overline{\text{SMC\_AMSn}}$ pin and asserting the $\overline{\text{SMC\_AMSn}}$ pin for the next access. Note that the SMC_B2ETIM.IT period may be extended using the SMC_B2ETIM.TT selection. The idle time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		7	7 SCLK clock cycles
10:8 (R/W)	TT	Transition Time. The SMC_B2ETIM.TT bits select a bus idle time (in SCLK cycles) that the SMC extends the SMC_B2ETIM.IT to allow for the subsequent access either using a different transfer direction or accessing a different bank. The transition time is from 1 to 7 SCLK cycles.	
		0	No bank transition
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The SMC_B2ETIM.PREAT bits select the pre-access time (in SCLK cycles) that the SMC waits after de-asserting the $\overline{\text{SMC\_AOE}}$/$\overline{\text{ADV}}$ pin before asserting the $\overline{\text{SMC\_ARE}}$/$\overline{\text{SMC\_AWE}}$ pin for the current access. The pre-access time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles

Table 9-13: SMC_B2ETIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	PREST	Pre Setup Time. The SMC_B2ETIM.PREST bits select the pre-setup time (in SCLK cycles) that the SMC asserts the SMC_AMSn pin before asserting the SMC_AOE/ADV pin for an access. The pre-setup time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles

Bank 3 Control Register

The SMC_B3CTL register enables bank 3 accesses and configures the memory access features for this bank.

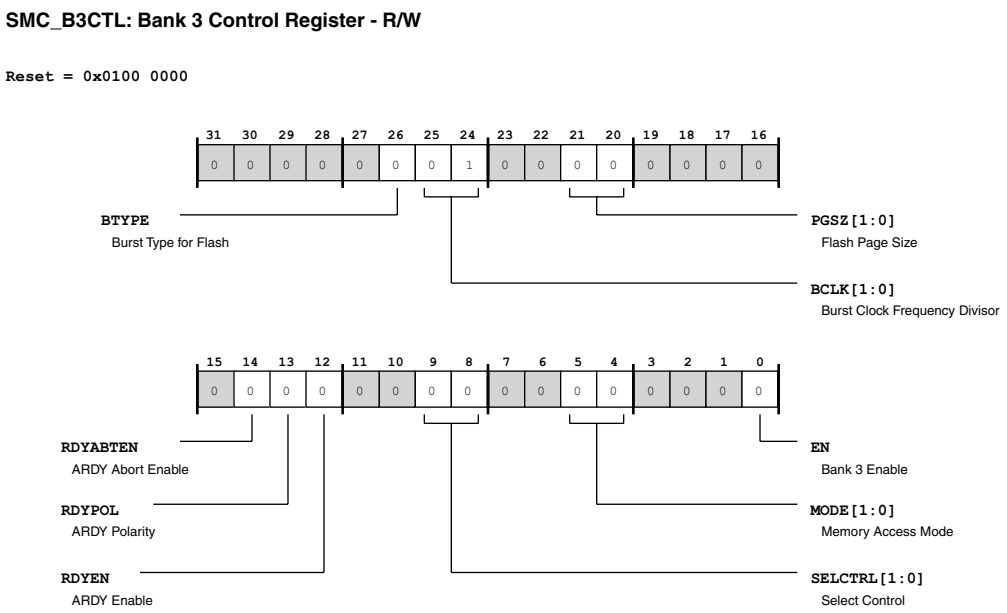


Figure 9-25: SMC_B3CTL Register Diagram

Table 9-14: SMC_B3CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/W)	BTYPE	Burst Type for Flash. The SMC_B3CTL.BTYPE bit selects the burst type that the SMC uses for accesses using sync burst flash protocol.	
		0	Wrap
		1	Sequential
25:24 (R/W)	BCLK	Burst Clock Frequency Divisor. The SMC_B3CTL.BCLK bits select the divisor that the SMC uses to determine the clock frequency for accesses using sync burst flash protocol.	
		0	Burst clock = SCLK ÷ 1
		1	Burst clock = SCLK ÷ 2
		2	Burst clock = SCLK ÷ 3
		3	Burst clock = SCLK ÷ 4
21:20 (R/W)	PGSZ	Flash Page Size. The SMC_B3CTL.PGSZ bits select the flash page size, if page flash or sync burst flash protocol has been enabled (SMC_B3CTL.MODE> 1). Note that the SMC_B3CTL.PGSZ bits must be set to match the flash protocol of the external flash memory device in the system. The typical SMC_B3CTL.PGSZ selection for external devices supporting async flash or async flash page protocols is 4 or 8 words. The typical SMC_B3CTL.PGSZ selection for external devices supporting sync burst flash protocol is 16 words.	
		0	4 words
		1	8 words
		2	16 words
		3	16 words

Table 9-14: SMC_B3CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	RDYABTEN	ARDY Abort Enable. The SMC_B3CTL.RDYABTEN bit enables the abort counter for the SMC_ARDY pin, if enabled (SMC_B3CTL.RDYEN =1). After SMC_B3TIM.RAT or SMC_B3TIM.WAT cycles, the SMC starts sampling the SMC_ARDY pin and starts the abort down counter (if enabled). The abort count is 64 cycles of SCLK. If the SMC detects that SMC_ARDY remains de-asserted when the counter expires, the SMC aborts the access and returns an error response back on the system bus.	
		0	Disable abort counter
		1	Enable abort counter
13 (R/W)	RDYPOL	ARDY Polarity. The SMC_B3CTL.RDYPOL bit selects the polarity (active high or low) for the SMC_ARDY pin, if enabled (SMC_B3CTL.RDYEN =1). When the SMC samples the SMC_ARDY pin in the selective active state, the transaction completes.	
		0	Low active ARDY
		1	High active ARDY
12 (R/W)	RDYEN	ARDY Enable. The SMC_B3CTL.RDYEN bit enables SMC_ARDY pin operation for bank 3 accesses. When enabled, the SMC uses SMC_ARDY (after the access time countdown) to determine completion of access to this memory bank. When disabled, the SMC ignores SMC_ARDY for accesses to this memory bank.	
		0	Disable ARDY
		1	Enable ARDY
9:8 (R/W)	SELCTRL	Select Control. The SMC_B3CTL.SELCTRL bits select the handling of the $\overline{\text{SMC\_AMSn}}$, $\overline{\text{SMC\_ARE}}$, $\overline{\text{SMC\_AOE}}$, and $\overline{\text{SMC\_AWE}}$ pins for memory access control.	
		0	AMS3 only
		1	AMS3 ored with ARE
		2	AMS3 ored with AOE
		3	AMS3 ored with AWE

Table 9-14: SMC_B3CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W)	MODE	Memory Access Mode. The SMC_B3CTL.MODE bits select the protocol the SMC uses for static memory read/write access. Note that the write protocol for async flash, async flash page, and sync burst flash are all similar; only the read protocols differ for these modes.	
		0	Async SRAM protocol
		1	Async flash protocol
		2	Async flash page protocol
		3	Sync burst flash protocol
0 (R/W)	EN	Bank 3 Enable. The SMC_B3CTL.EN bit enables accesses to the memory in bank 3. When this bit is disabled, accesses to bank 3 return an error response.	
		0	Disable access
		1	Enable access

Bank 3 Timing Register

The SMC_B3TIM register configures bank 3 read and write access, setup, and hold timing for this bank. Note that read and write timing configurations are independent and may differ.

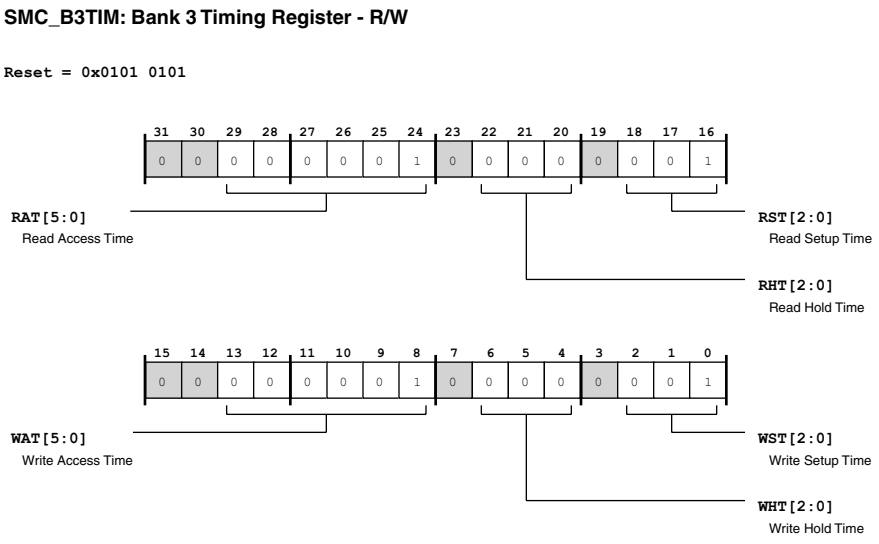


Figure 9-26: SMC_B3TIM Register Diagram

Table 9-15: SMC_B3TIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29:24 (R/W)	RAT	Read Access Time. The SMC_B3TIM.RAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_ARE pin for a read access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
22:20 (R/W)	RHT	Read Hold Time. The SMC_B3TIM.RHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_ARE pin before asserting the SMC_AOE pin for the next access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
18:16 (R/W)	RST	Read Setup Time. The SMC_B3TIM.RST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_ARE pin for an access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
13:8 (R/W)	WAT	Write Access Time. The SMC_B3TIM.WAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_AWE pin for a write access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles

Table 9-15: SMC_B3TIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6:4 (R/W)	WHT	Write Hold Time. The SMC_B3TIM.WHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the $\overline{\text{SMC\_AWE}}$ pin before de-asserting the $\overline{\text{SMC\_AOE}}$ pin for the current access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
2:0 (R/W)	WST	Write Setup Time. The SMC_B3TIM.WST bits select the setup time (in SCLK cycles) that the SMC asserts the $\overline{\text{SMC\_AOE}}$ pin before asserting the $\overline{\text{SMC\_AWE}}$ pin for a write access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Bank 3 Extended Timing Register

The SMC_B3ETIM register configures extensions to access times and idle times, augmenting the setup, hold, and access times configured with the SMC_B3TIM register.

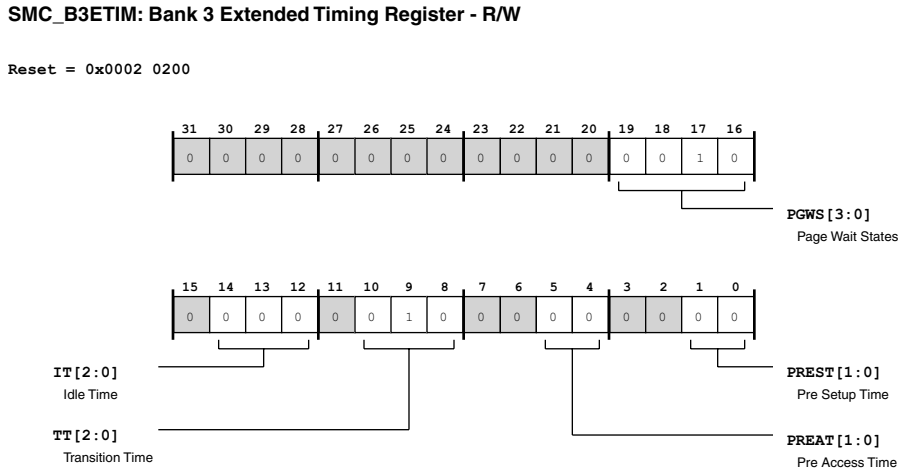


Figure 9-27: SMC_B3ETIM Register Diagram

Table 9-16: SMC_B3ETIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	PGWS	Page Wait States. The SMC_B3ETIM.PGWS bits select a page access extension time (in SCLK cycles) that the SMC waits during read accesses when configured for flash page protocol (SMC_B3CTL.MODE =2). The wait time is from 1 to 15 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycles
		15	15 SCLK clock cycles
14:12 (R/W)	IT	Idle Time. The SMC_B3ETIM.IT bits select a bus idle time (in SCLK cycles) that the SMC waits between de-asserting the $\overline{\text{SMC\_AMSn}}$ pin and asserting the $\overline{\text{SMC\_AMSn}}$ pin for the next access. Note that the SMC_B3ETIM.IT period may be extended using the SMC_B3ETIM.TT selection. The idle time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		7	7 SCLK clock cycles
10:8 (R/W)	TT	Transition Time. The SMC_B3ETIM.TT bits select a bus idle time (in SCLK cycles) that the SMC extends the SMC_B3ETIM.IT to allow for the subsequent access either using a different transfer direction or accessing a different bank. The transition time is from 1 to 7 SCLK cycles.	
		0	No bank transition
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The SMC_B3ETIM.PREAT bits select the pre-access time (in SCLK cycles) that the SMC waits after de-asserting the $\overline{\text{SMC\_AOE}}$/$\overline{\text{ADV}}$ pin before asserting the $\overline{\text{SMC\_ARE}}$/$\overline{\text{SMC\_AWE}}$ pin for the current access. The pre-access time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles

Table 9-16: SMC_B3ETIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	PREST	Pre Setup Time. The SMC_B3ETIM.PREST bits select the pre-setup time (in SCLK cycles) that the SMC asserts the $\overline{\text{SMC\_AMSn}}$ pin before asserting the $\overline{\text{SMC\_AOE}}$/$\overline{\text{ADV}}$ pin for an access. The pre-setup time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles

10 L2 Memory Controller (L2CTL)

The L2 memory controller manages L2 SRAM and ROM memories and provides the interface between these memories and the system crossbar. The L2 memory is shared resource. For example, on multi-core processors, L2 can be accessed by both processor cores, by DMA controllers, and the system debug unit (SDU).

L2 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 Memory Controller Features

The L2 memory features include:

- Operation at SYSCLK frequency
- Write protection of SRAM banks
- ECC protection of SRAM area
- ECC memory refresh
- 256K byte of SRAM grouped into eight banks, 32K bytes each
- 32K byte of ROM featuring the boot code
- Full-duplex 64-bit port dedicated to data transfers to/from the one or more processor cores
- Full-duplex 32-bit port dedicated to DMA transfers and system debug
- Support for locked TESTSET operation for semaphore handling

L2 Memory Controller Functional Description

All L2 SRAM and ROM memory banks are managed by the L2 memory controller (L2CTL). The controller interfaces the memories to the system, arbitrates competing accesses and write protection, and ensures SRAM data integrity. The L2 memory domain is a unified instruction and data memory and can hold any mixture of code and data required by the system design.

The table shows the L2 memory map for the ADSP-BF60x processor.

Table 10-1: ADSP-BF60x Processor - L2 Memory Address Mapping

Start Address	End Address	Description
0xC8000000	0xC8007FFF	L2 Bank 0 ROM (32KB)
0xC8080000	0xC8087FFF	L2 Bank 0 RAM (32KB)
0xC8088000	0xC808FFFF	L2 Bank 1 RAM (32KB)
0xC8090000	0xC8097FFF	L2 Bank 2 RAM (32KB)
0xC8098000	0xC809FFFF	L2 Bank 3 RAM (32KB)
0xC80A0000	0xC80A7FFF	L2 Bank 4 RAM (32KB)
0xC80A8000	0xC80AFFFF	L2 Bank 5 RAM (32KB)
0xC80B0000	0xC80B7FFF	L2 Bank 6 RAM (32KB)
0xC80B8000	0xC80BFFFF	L2 Bank 7 RAM (32KB)

The following sections provide a functional description of the L2CTL:

- *ADSP-BF60x L2CTL Register List*
- *ADSP-BF60x L2CTL Interrupt List*
- *L2 Memory Controller Block Diagram*
- *L2 Memory Controller Architectural Concepts*

ADSP-BF60x L2CTL Register List

The internal L2 memory controller (L2CTL) includes the controls to manage each L2 memory bank independently. This controller supports ECC and non-ECC operation with support for error tracking (by address or bus ID) and support for error type determination. A set of registers govern L2CTL operations. For more information on L2CTL functionality, see the L2CTL register descriptions.

Table 10-2: ADSP-BF60x L2CTL Register List

Name	Description
L2CTL_CTL	Control Register
L2CTL_ACTL_C0	Access Control Core 0 Register
L2CTL_ACTL_C1	Access Control Core 1 Register
L2CTL_ACTL_SYS	Access Control System Register

Table 10-2: ADSP-BF60x L2CTL Register List (Continued)

Name	Description
L2CTL_STAT	Status Register
L2CTL_RPCR	Read Priority Count Register
L2CTL_WPCR	Write Priority Count Register
L2CTL_RFA	Refresh 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_ET0	Error Type 0 Register
L2CTL_EADDR0	Error Type 0 Address Register
L2CTL_ET1	Error Type 1 Register
L2CTL_EADDR1	Error Type 1 Address Register

ADSP-BF60x L2CTL Interrupt List

Table 10-3: ADSP-BF60x L2CTL Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
L2CTL0 ECC Error	4		LEVEL

L2 Memory Controller Block Diagram

As shown in the following figure, the L2 controller has two ports that interface to system crossbars. Port 0 is a 64-bit interface that is dedicated to core traffic, and port 1 is a 32-bit interface that connects through DMA access. Each port has a read channel and a write channel.

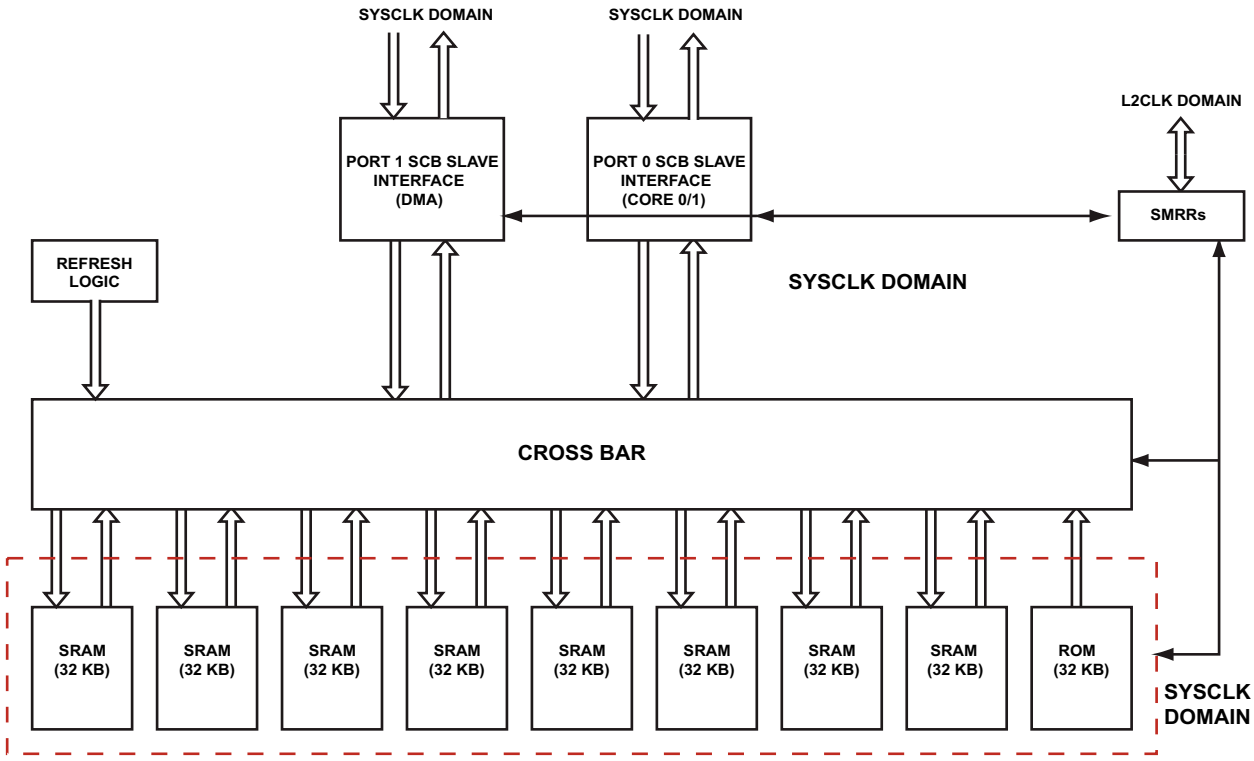


Figure 10-1: L2 Memory Controller Block Diagram

The SRAM is organized in multiple banks, and each bank has 32K Bytes of data. Within each bank, data is organized into 4096 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.

L2 Memory Controller Architectural Concepts

The following sections describe L2 memory controller architecture features.

- *Access Ordering*
- *Read/Write Latency and Throughput*
- *Arbitration and Priority*

Access Characteristics

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

The L2 controller supports restriction of both core and DMA write accesses to a particular memory bank. The write access by a port to a particular RAM bank can be disabled by setting the appropriate bank write disable bit in that port's `L2CTL_ACTL_C0` and `L2CTL_ACTL_C1` registers. Illegal access attempts generate an error response.

Read/Write Latency and Throughput

The L2 memory design is optimized for burst accesses at the crossbar interface. Write data of 8/16/32-bit is buffered and converted to an equivalent 64-bit access. This conversion creates modulo-32-bit writes if the starting addresses are 32-bit aligned. A single 8- 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 additional latency of two *L2CLK* 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.

If two cores simultaneously try to access L2 for the same kind of access (both read or both write), even to different banks, only one core access is allowed at a time, as there is only one read port and one write port shared between the cores. However, if one core issues a write and the other issues a read, then access can proceed simultaneously, and inside L2 there is no extra latency, as long as the accesses are to different banks (assuming pending DMA traffic is also to a non-conflicting bank).

If a core and DMA both access the same bank, then the best access rate that DMA can achieve is one 64-bit access in every three *L2CLK* cycles during the conflict period. This 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.

Arbitration and Priority

Each bank of L2 RAM/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.

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

Table 10-4: Fixed Priority

Channel	Priority Level
L2 Refresh Request	5 (highest)
Port 0 Read Channel	4
Port 0 Write Channel	3
Port 1 Read Channel	2
Port 1 Write Channel	1 (lowest)

The arbiters also support priority elevation for a particular channel that has been starved of grants for a very large number of *L2CLK* 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 causes that particular channel to become the highest priority master for the next grant cycle (this is 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.

If there is 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, and with the pipelined implementation of urgent priority, the best grant rate DMA can achieve is 1 in 3 *L2CLK* cycles during the conflict period. This can be achieved by programming the bits in the `L2CTL_RPCR` and `L2CTL_WPCR` registers appropriately.

Urgent priority requests can be disabled by setting the `L2CTL_CTL.DISURP` bit. This 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 following table provides the various priority levels for the L2 controller.

Table 10-5: 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

To ensure data integrity, all L2 SRAM is protected with an error-correcting code (ECC). The ROM is not protected. Each 32-bit SRAM entity is protected by a 7-bit ECC checksum. If the L2 controller detects a single bit error at read time, the controller identifies the failing bit and auto-corrects the value outputted to the system. Also, the L2 controller can detect 2-bit errors safely and can detect a large range of multi-bit errors. This scheme is often referred to as (39,32) single-error correction, dual-error detection (SECDED) checksum protection.

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 L2 memory is written by 8-bit and 16-bit values. ECC protection can be disabled for individual SRAM banks by setting the L2CTL_CTL.BK0EDIS through L2CTL_CTL.BK7EDIS disable bits. Due to caching mechanisms of the processor core(s) and data bursting of the DMA channels, 8- and 16-bit write accesses are rather uncommon, and these writes are typically triggered only by 2-dimensional DMA operation or un-cached 8- and 16-bit store instructions.

For system integrity testing, the L2 controller also provides a method for accessing the ECC checksum area directly. The L2CTL_CTL.ECCMAP0 through L2CTL_CTL.ECCMAP7 bits map the ECC checksum values into the address space of the data bits. This feature can be activated per SRAM bank. In this mode, only 32-bit accesses are allowed. 32-bit reads return the checksum value in the lower seven bits while the upper bits read zero. Any 32-bit write overwrites the checksum. The upper bits are ignored.

Using this checksum mapping feature, safety critical applications can verify the ECC hardware during boot up sequence or even at run time. It is not required to explicitly set the L2CTL_CTL.BK0EDIS through L2CTL_CTL.BK7EDIS disable bits. To test the ECC hardware, use the following steps:

1. Write data values to L2 SRAM destination (preferable an even number of 32-bit words)
2. If data cache enabled, make sure it flushes data out
3. Execute SSYNC instruction
4. Set L2CTL_CTL.ECCMAP7—L2CTL_CTL.ECCMAP0 bits of interest
5. Execute SSYNC instruction
6. Write checksum values using 32-bit store instructions
7. If data cache enabled, make sure it flushes checksum values out
8. Execute SSYNC instruction
9. Clear L2CTL_CTL.ECCMAP7—L2CTL_CTL.ECCMAP0 bits
10. Execute SSYNC instruction
11. Read data values back

ECC Error Management

The L2 controller flags two- 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, and
- 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, the user declares ECC_ERR events as system faults in the system event controller (SEC). Whether or not these are reported, the interrupt service routine can consult 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 or whether the data at the failing L2 address was less critical or can be restored. The L2CTL_STAT.ECCERR0 through L2CTL_STAT.ECCERR7 flags need to be 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 controller provides a special memory refresh mechanism.

Software can initiate a memory refresh cycle of a 64-bit SRAM entity by writing the address of interest into the refresh address register, L2CTL_RFA. The write triggers an atomic operation. In this operation, the L2 controller reads a 64-bit entity from the targeted memory, applies an ECC algorithm to the two 32-bit words, and writes the corrected data back to memory. In case of dual- or multi-bit errors, the ECC_ERR interrupt is raised, and data is not written back to memory.

While the atomic refresh operation is ongoing, other accesses to the same SRAM bank are locked out. An ongoing refresh operation is signaled by the L2CTL_STAT.RFRS status bit. The bit is cleared by hardware after the operation has finished. The content of the L2CTL_RFA register must not change while the refresh operation is ongoing.

In safety critical applications, software may refresh all L2 SRAM by periodically writing to the L2CTL_RFA register with values, incrementing by a value of 8 until all SRAM locations have been refreshed.

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

Access Control

The L2 controller provides write protection, which prevents unauthorized data sources from (over) writing individual SRAM banks. By default, write protection is disabled, permitting the processor cores, DMA controller, and system debug unit (SDU) write access to every SRAM bank. Using L2 controller features, programs may selectively disable write privileges for the processor core(s) and/or the DMA controller to any number of L2 SRAM banks.

The following bits disable L2 SRAM bank access:

- The `L2CTL_ACTL_C0.BK0WDIS` - `L2CTL_ACTL_C0.BK7WDIS` bits disable core 0 write privileges to L2 SRAM banks 0-7.
- The `L2CTL_ACTL_C1.BK0WDIS` - `L2CTL_ACTL_C1.BK7WDIS` bits disable core 1write privileges to L2 SRAM banks 0-7.
- The `L2CTL_ACTL_SYS.BK0WDIS` - `L2CTL_ACTL_SYS.BK7WDIS` bits disable DMA controller write privileges to L2 SRAM banks 0-7.

When an unauthorized write is detected, the L2 controller generates an error response to the system buses. At any time, any master may read all SRAM locations, because read privileges are not controlled.

NOTE: Note that the `TESTSET` instruction (for semaphore handling) requires full write access.

Another level of protection is available with the `LOCK` bit available in the `L2CTL_CTL`, `L2CTL_ACTL_C0`, `L2CTL_ACTL_C1`, and `L2CTL_ACTL_SYS` registers. When this bit is set and global locking is enabled with the system protection unit (SPU), these control registers become write protected (locked).

By using the combination of the write disable bits for banks and the lock bits for control registers, systems can ensure that multiple steps are required before any data source can accidentally overwrite protected data.

L2 Memory Controller Event Control

The following sections describe L2 memory controller event control features, such as error response.

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
- A write access (including a TESTSET) to a restricted memory bank
- An ECC multi-bit error in an ECC enabled bank. A non-modulo32-bit write to an ECC enabled bank can also potentially create a bus error response due to an ECC multi-bit error. This is because the L2 controller implements a 32-bit ECC, and therefore a non-modulo32-bit write results in a read. This read may create multi-bit errors even if the memory was initialized.

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

ADSP-BF60x L2CTL Register Descriptions

L2 Memory Controller (L2CTL) contains the following registers.

Table 10-6: ADSP-BF60x L2CTL Register List

Name	Description
L2CTL_CTL	Control Register
L2CTL_ACTL_C0	Access Control Core 0 Register
L2CTL_ACTL_C1	Access Control Core 1 Register
L2CTL_ACTL_SYS	Access Control System Register
L2CTL_STAT	Status Register
L2CTL_RPCR	Read Priority Count Register
L2CTL_WPCR	Write Priority Count Register
L2CTL_RFA	Refresh 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_ET0	Error Type 0 Register

Table 10-6: ADSP-BF60x L2CTL Register List (Continued)

Name	Description
L2CTL_EADDR0	Error Type 0 Address Register
L2CTL_ET1	Error Type 1 Register
L2CTL_EADDR1	Error Type 1 Address 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).

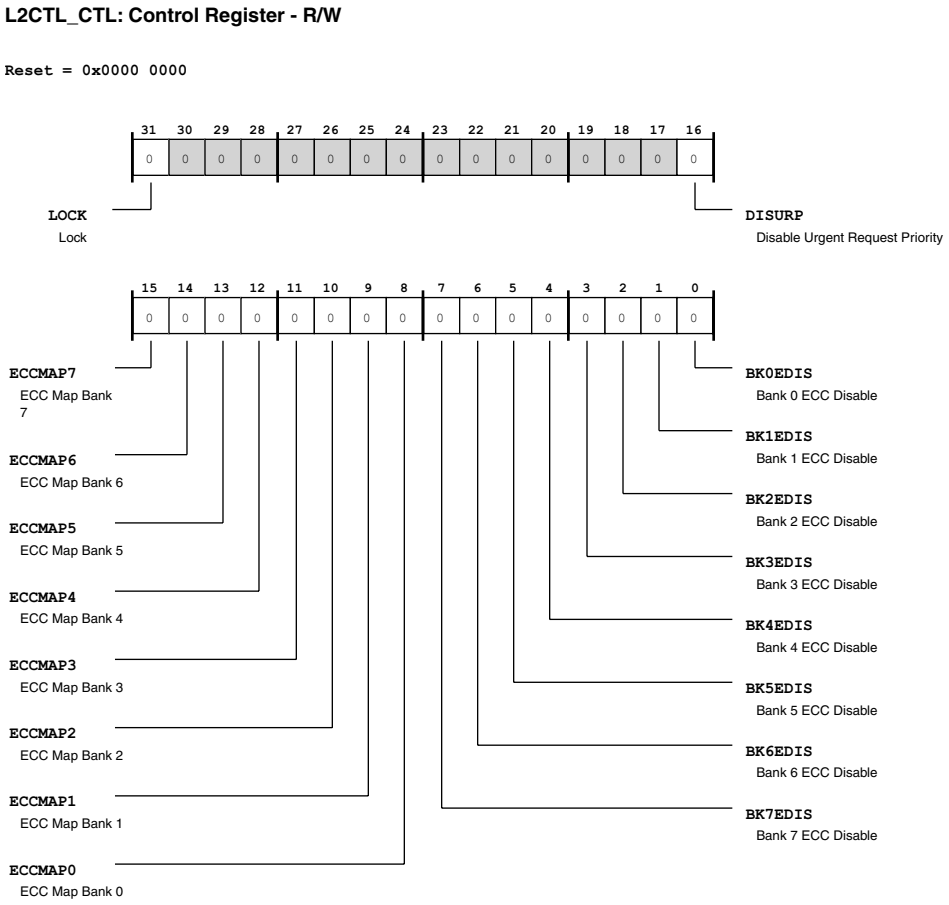


Figure 10-2: L2CTL_CTL Register Diagram

Table 10-7: L2CTL_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the L2CTL_CTL.LOCK bit is set, the L2CTL_CTL register is read only (locked).	
		0	Unlock
		1	Lock
16 (R/W)	DISURP	Disable Urgent Request Priority. The L2CTL_CTL.DISURP disables urgent request priority mode for all L2 banks.	
		0	Enable URP
		1	Disable URP
15 (R/W)	ECCMAP7	ECC Map Bank 7. The L2CTL_CTL.ECCMAP7 bit selects whether L2 bank 7 addresses ECC RAM or data RAM.	
		0	Data RAM
		1	ECC RAM
14 (R/W)	ECCMAP6	ECC Map Bank 6. The L2CTL_CTL.ECCMAP6 bit selects whether L2 bank 6 addresses ECC RAM or data RAM.	
		0	Data RAM
		1	ECC RAM
13 (R/W)	ECCMAP5	ECC Map Bank 5. The L2CTL_CTL.ECCMAP5 bit selects whether L2 bank 5 addresses ECC RAM or data RAM.	
		0	Data RAM
		1	ECC RAM
12 (R/W)	ECCMAP4	ECC Map Bank 4. The L2CTL_CTL.ECCMAP4 bit selects whether L2 bank 4 addresses ECC RAM or data RAM.	
		0	Data RAM
		1	ECC RAM

Table 10-7: L2CTL_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	ECCMAP3	ECC Map Bank 3. The L2CTL_CTL.ECCMAP3 bit selects whether L2 bank 3 addresses ECC RAM or data RAM.	
		0	Data RAM
		1	ECC RAM
10 (R/W)	ECCMAP2	ECC Map Bank 2. The L2CTL_CTL.ECCMAP2 bit selects whether L2 bank 2 addresses ECC RAM or data RAM.	
		0	Data RAM
		1	ECC RAM
9 (R/W)	ECCMAP1	ECC Map Bank 1. The L2CTL_CTL.ECCMAP1 bit selects whether L2 bank 1 addresses ECC RAM or data RAM.	
		0	Data RAM
		1	ECC RAM
8 (R/W)	ECCMAP0	ECC Map Bank 0. The L2CTL_CTL.ECCMAP0 bit selects whether L2 bank 0 addresses ECC RAM or data RAM.	
		0	Data RAM
		1	ECC RAM
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

Table 10-7: L2CTL_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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

Access Control Core 0 Register

The L2CTL_ACTL_C0 register includes a write protection bit and enables core 0 write access for L2 banks.

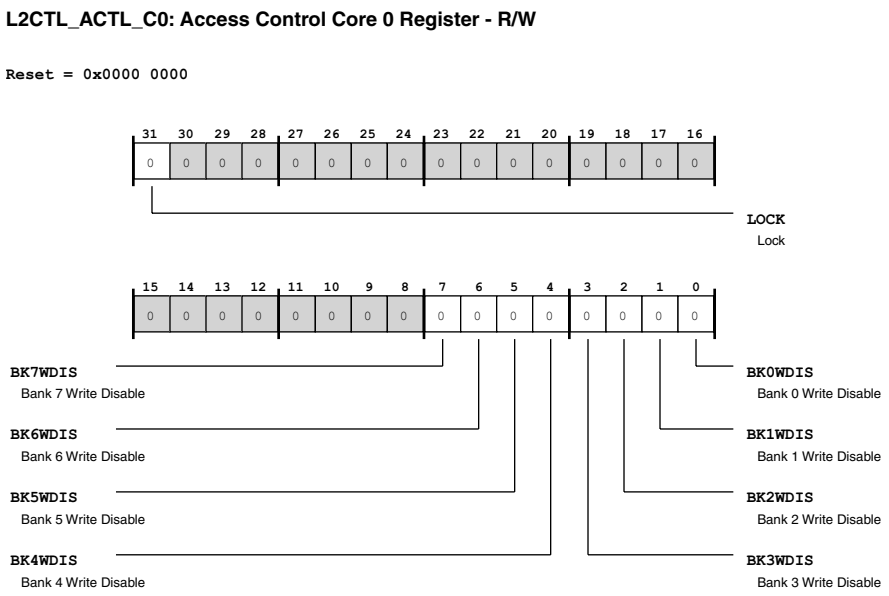


Figure 10-3: L2CTL_ACTL_C0 Register Diagram

Table 10-8: L2CTL_ACTL_C0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the L2CTL_ACTL_C0.LOCK bit is set, the L2CTL_ACTL_C0 register is read only (locked).	
		0	Unlock
		1	Lock
7 (R/W)	BK7WDIS	Bank 7 Write Disable. The L2CTL_ACTL_C0.BK7WDIS bit disables core 0 writes to L2 bank 7 RAM.	
		0	Enable Write
		1	Disable Write
6 (R/W)	BK6WDIS	Bank 6 Write Disable. The L2CTL_ACTL_C0.BK6WDIS bit disables core 0 writes to L2 bank 6 RAM.	
		0	Enable Write
		1	Disable Write

Table 10-8: L2CTL_ACTL_C0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	BK5WDIS	Bank 5 Write Disable. The L2CTL_ACTL_C0.BK5WDIS bit disables core 0 writes to L2 bank 5 RAM.	
		0	Enable Write
		1	Disable Write
4 (R/W)	BK4WDIS	Bank 4 Write Disable. The L2CTL_ACTL_C0.BK4WDIS bit disables core 0 writes to L2 bank 4 RAM.	
		0	Enable Write
		1	Disable Write
3 (R/W)	BK3WDIS	Bank 3 Write Disable. The L2CTL_ACTL_C0.BK3WDIS bit disables core 0 writes to L2 bank 3 RAM.	
		0	Enable Write
		1	Disable Write
2 (R/W)	BK2WDIS	Bank 2 Write Disable. The L2CTL_ACTL_C0.BK2WDIS bit disables core 0 writes to L2 bank 2 RAM.	
		0	Enable Write
		1	Disable Write
1 (R/W)	BK1WDIS	Bank 1 Write Disable. The L2CTL_ACTL_C0.BK1WDIS bit disables core 0 writes to L2 bank 1 RAM.	
		0	Enable Write
		1	Disable Write
0 (R/W)	BK0WDIS	Bank 0 Write Disable. The L2CTL_ACTL_C0.BK0WDIS bit disables core 0 writes to L2 bank 0 RAM.	
		0	Enable Write
		1	Disable Write

Access Control Core 1 Register

The L2CTL_ACTL_C1 register includes a write protection bit and enables core 1 write access for L2 banks.

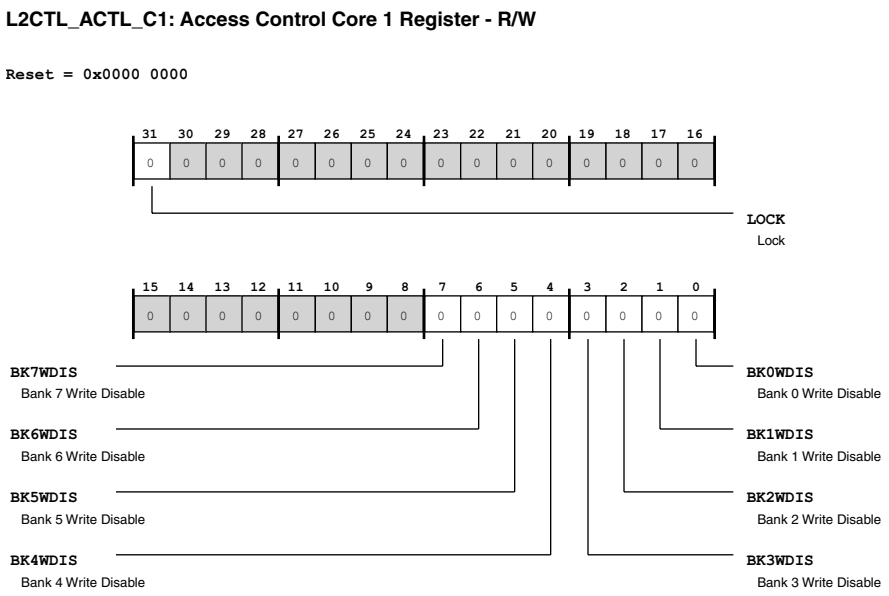


Figure 10-4: L2CTL_ACTL_C1 Register Diagram

Table 10-9: L2CTL_ACTL_C1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the L2CTL_ACTL_C1.LOCK bit is set, the L2CTL_ACTL_C1 register is read only (locked).	
		0	Unlock
		1	Lock
7 (R/W)	BK7WDIS	Bank 7 Write Disable. The L2CTL_ACTL_C1.BK7WDIS bit disables core 1 writes to L2 bank 7 RAM.	
		0	Enable Write
		1	Disable Write
6 (R/W)	BK6WDIS	Bank 6 Write Disable. The L2CTL_ACTL_C1.BK6WDIS bit disables core 1 writes to L2 bank 6 RAM.	
		0	Enable Write
		1	Disable Write

Table 10-9: L2CTL_ACTL_C1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	BK5WDIS	Bank 5 Write Disable. The L2CTL_ACTL_C1.BK5WDIS bit disables core 1 writes to L2 bank 5 RAM.	
		0	Enable Write
		1	Disable Write
4 (R/W)	BK4WDIS	Bank 4 Write Disable. The L2CTL_ACTL_C1.BK4WDIS bit disables core 1 writes to L2 bank 4 RAM.	
		0	Enable Write
		1	Disable Write
3 (R/W)	BK3WDIS	Bank 3 Write Disable. The L2CTL_ACTL_C1.BK3WDIS bit disables core 1 writes to L2 bank 3 RAM.	
		0	Enable Write
		1	Disable Write
2 (R/W)	BK2WDIS	Bank 2 Write Disable. The L2CTL_ACTL_C1.BK2WDIS bit disables core 1 writes to L2 bank 2 RAM.	
		0	Enable Write
		1	Disable Write
1 (R/W)	BK1WDIS	Bank 1 Write Disable. The L2CTL_ACTL_C1.BK1WDIS bit disables core 1 writes to L2 bank 1 RAM.	
		0	Enable Write
		1	Disable Write
0 (R/W)	BK0WDIS	Bank 0 Write Disable. The L2CTL_ACTL_C1.BK0WDIS bit disables core 1 writes to L2 bank 0 RAM.	
		0	Enable Write
		1	Disable Write

Access Control System Register

The L2CTL_ACTL_SYS register includes a write protection bit and enables system and DMA write access for L2 banks.

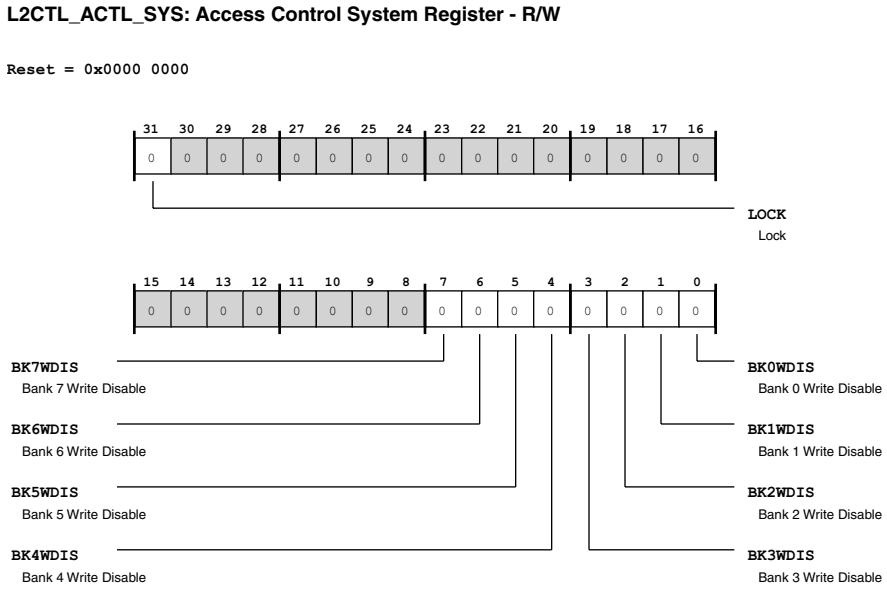


Figure 10-5: L2CTL_ACTL_SYS Register Diagram

Table 10-10: L2CTL_ACTL_SYS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the L2CTL_ACTL_SYS.LOCK bit is set, the L2CTL_ACTL_SYS register is read only (locked).	
		0	Unlock
		1	Lock
7 (R/W)	BK7WDIS	Bank 7 Write Disable. The L2CTL_ACTL_SYS.BK7WDIS bit disables system or DMA writes to L2 bank 7 RAM.	
		0	Enable Write
		1	Disable Write

Table 10-10: L2CTL_ACTL_SYS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	BK6WDIS	Bank 6 Write Disable. The L2CTL_ACTL_SYS.BK6WDIS bit disables system or DMA writes to L2 bank 6 RAM.	
		0	Enable Write
		1	Disable Write
5 (R/W)	BK5WDIS	Bank 5 Write Disable. The L2CTL_ACTL_SYS.BK5WDIS bit disables system or DMA writes to L2 bank 5 RAM.	
		0	Enable Write
		1	Disable Write
4 (R/W)	BK4WDIS	Bank 4 Write Disable. The L2CTL_ACTL_SYS.BK4WDIS bit disables system or DMA writes to L2 bank 4 RAM.	
		0	Enable Write
		1	Disable Write
3 (R/W)	BK3WDIS	Bank 3 Write Disable. The L2CTL_ACTL_SYS.BK3WDIS bit disables system or DMA writes to L2 bank 3 RAM.	
		0	Enable Write
		1	Disable Write
2 (R/W)	BK2WDIS	Bank 2 Write Disable. The L2CTL_ACTL_SYS.BK2WDIS bit disables system or DMA writes to L2 bank 2 RAM.	
		0	Enable Write
		1	Disable Write
1 (R/W)	BK1WDIS	Bank 1 Write Disable. The L2CTL_ACTL_SYS.BK1WDIS bit disables system or DMA writes to L2 bank 1 RAM.	
		0	Enable Write
		1	Disable Write

Table 10-10: L2CTL_ACTL_SYS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	BK0WDIS	Bank 0 Write Disable. The L2CTL_ACTL_SYS.BK0WDIS bit disables system or DMA writes to L2 bank 0 RAM.	
		0	Enable Write
		1	Disable Write

Status Register

The L2CTL_STAT register indicates ECC error status, refresh register status, and bus error status.

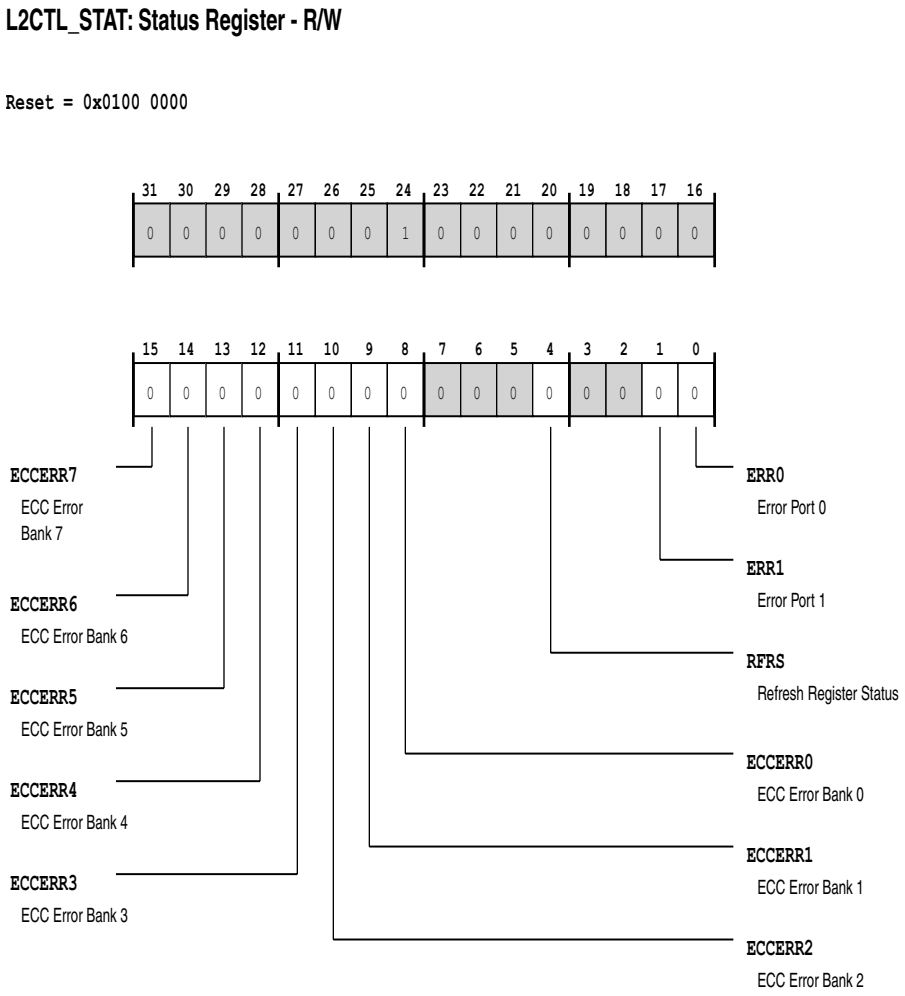


Figure 10-6: L2CTL_STAT Register Diagram

Table 10-11: L2CTL_STAT Register Fields

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

Table 10-11: L2CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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
4 (R/NW)	RFRS	Refresh Register Status. The L2CTL_STAT.RFRS bit indicates whether a refresh request is pending (in progress) or that there are no pending requests.	
		0	No Pending Requests
		1	Request Pending (Refresh in Progress)
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

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 L2CLK 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.

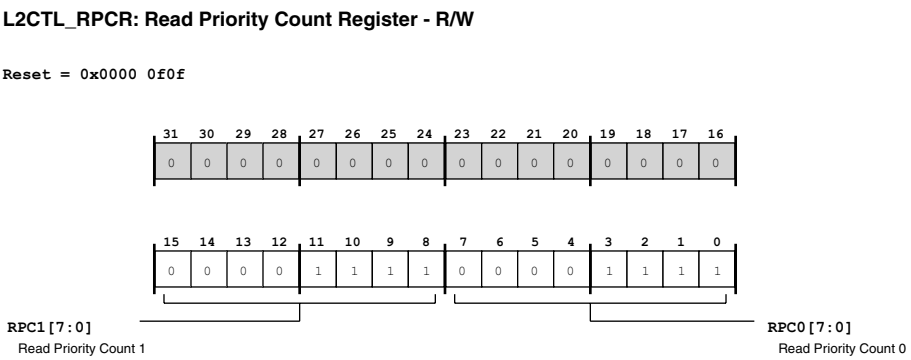


Figure 10-7: L2CTL_RPCR Register Diagram

Table 10-12: L2CTL_RPCR Register Fields

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

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 L2CLK 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.

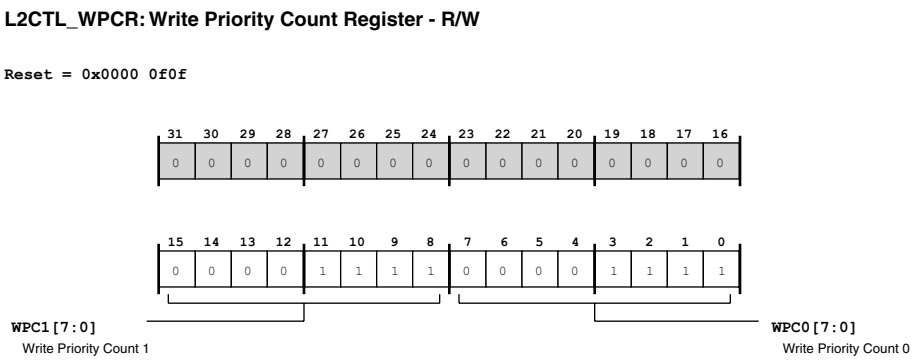


Figure 10-8: L2CTL_WPCR Register Diagram

Table 10-13: L2CTL_WPCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	WPC1	Write Priority Count 1. The L2CTL_WPCR.WPC1 bits hold the priority count for L2 bus write channel 0.
7:0 (R/W)	WPC0	Write Priority Count 0. The L2CTL_WPCR.WPC0 bits hold the priority count for L2 bus write channel 1.

Refresh Address Register

The L2CTL_RFA register stores the refresh address value. When this register is written, L2 initiates an atomic read-write operation to the address value written into the register. This is a read/write register, but a new value in the corresponding field has to be written only when there are no outstanding refresh request pending (L2CTL_STAT.RFRS =0). If a write occurs while a request is pending, the L2CTL generates a bus error, and the write does not take effect.

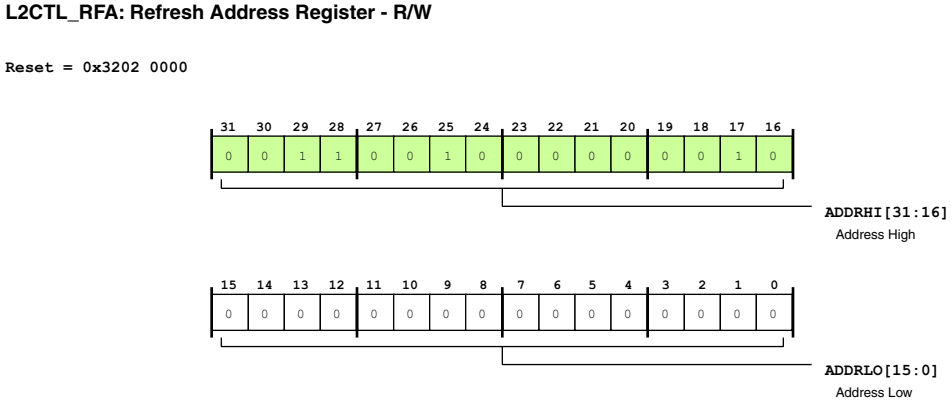


Figure 10-9: L2CTL_RFA Register Diagram

Table 10-14: L2CTL_RFA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	ADDRHI	Address High. The L2CTL_RFA.ADDRHI bits hold the high 16-bits of the L2 refresh address. Note that the upper 14 bits are hard-coded to the upper bits of the L2 address map.
15:0 (R/W)	ADDRLO	Address Low. The L2CTL_RFA.ADDRLO bits hold the low 16-bits of the L2 refresh address. Note that the lowest three bits are do not care.

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.

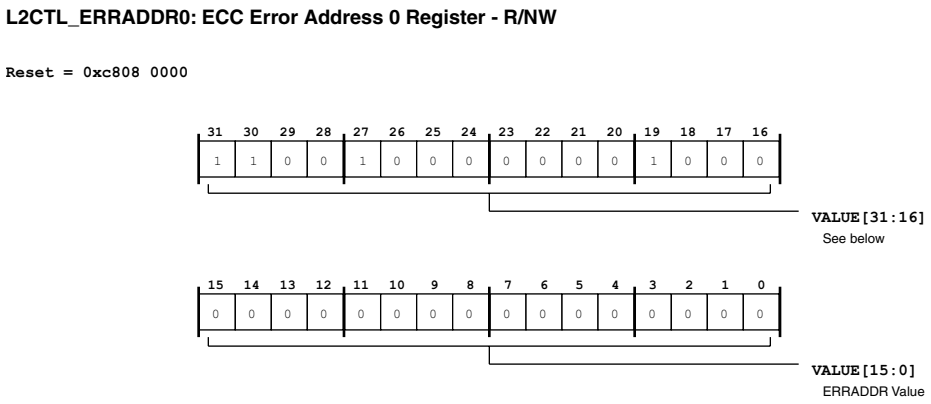


Figure 10-10: L2CTL_ERRADDR0 Register Diagram

Table 10-15: L2CTL_ERRADDR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_ERRADDR0.VALUE 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.

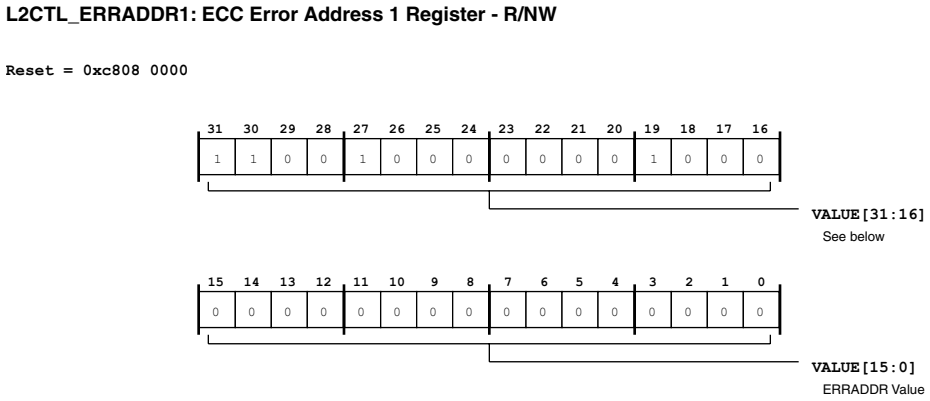


Figure 10-11: L2CTL_ERRADDR1 Register Diagram

Table 10-16: L2CTL_ERRADDR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_ERRADDR1.VALUE 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.

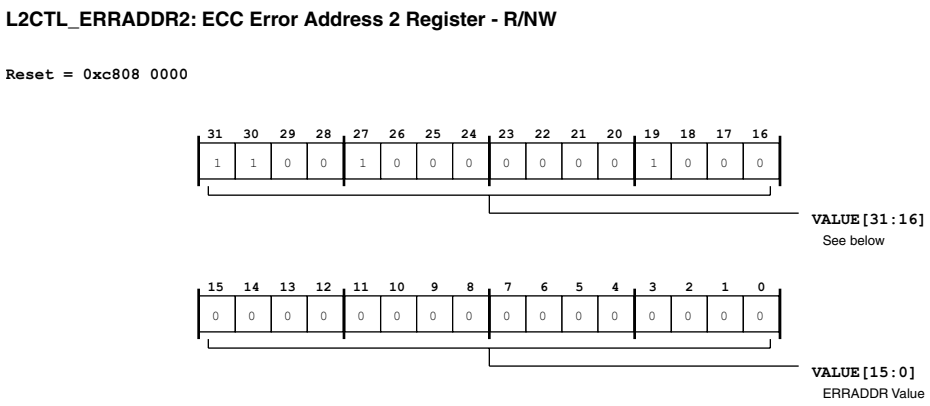


Figure 10-12: L2CTL_ERRADDR2 Register Diagram

Table 10-17: L2CTL_ERRADDR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_ERRADDR2.VALUE 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.

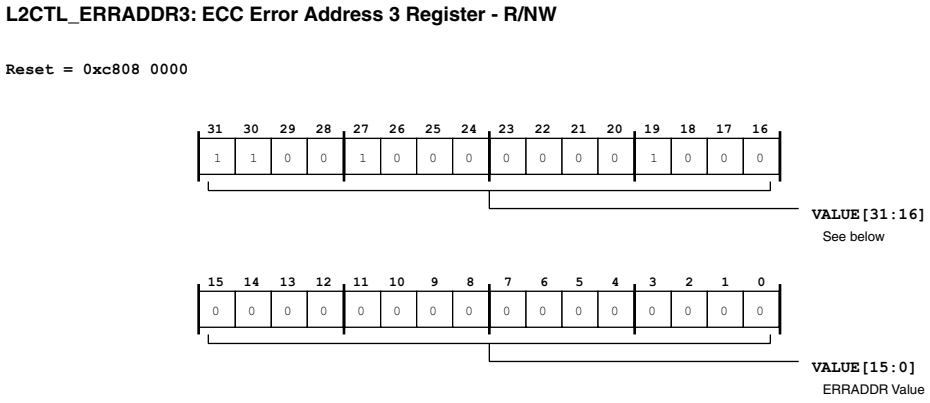


Figure 10-13: L2CTL_ERRADDR3 Register Diagram

Table 10-18: L2CTL_ERRADDR3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_ERRADDR3.VALUE 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.

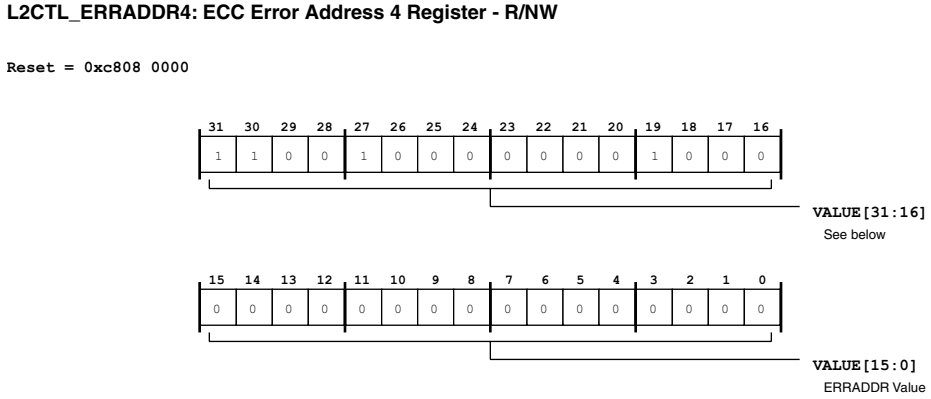


Figure 10-14: L2CTL_ERRADDR4 Register Diagram

Table 10-19: L2CTL_ERRADDR4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_ERRADDR4.VALUE 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.

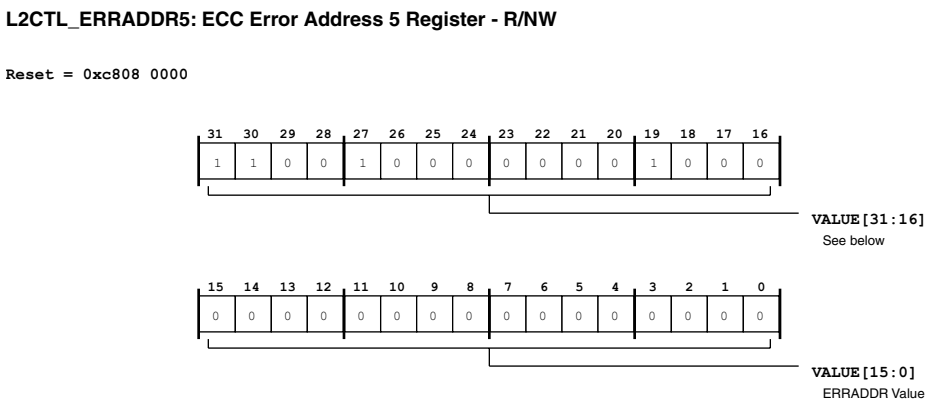


Figure 10-15: L2CTL_ERRADDR5 Register Diagram

Table 10-20: L2CTL_ERRADDR5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_ERRADDR5.VALUE 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.

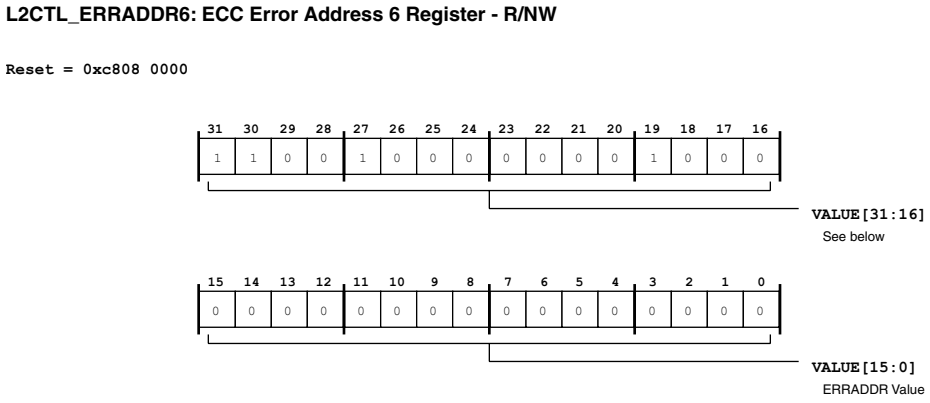


Figure 10-16: L2CTL_ERRADDR6 Register Diagram

Table 10-21: L2CTL_ERRADDR6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_ERRADDR6.VALUE 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.

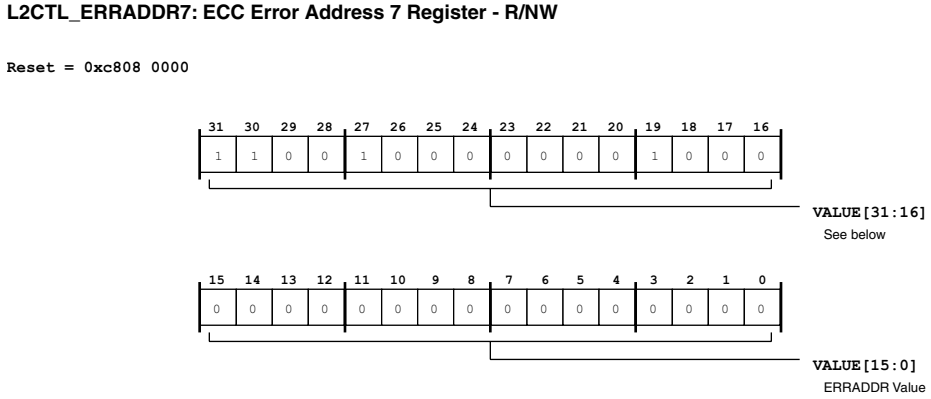


Figure 10-17: L2CTL_ERRADDR7 Register Diagram

Table 10-22: L2CTL_ERRADDR7 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_ERRADDR7.VALUE 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. 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).

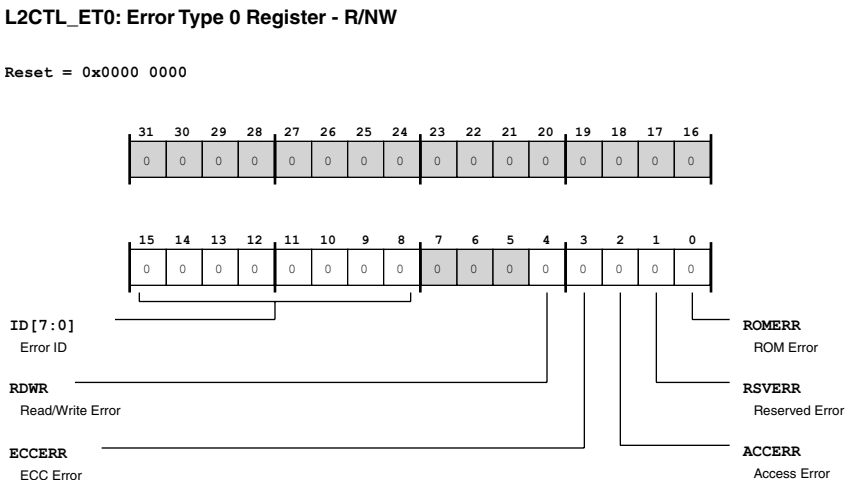


Figure 10-18: L2CTL_ET0 Register Diagram

Table 10-23: L2CTL_ET0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:8 (R/NW)	ID	Error ID. The L2CTL_ET0.ID bits hold the bus master ID of the access that caused error.	
4 (R/NW)	RDWR	Read/Write Error. The L2CTL_ET0.RDWR bit indicates whether a read or write access caused error.	
		0	Read Access created Error
		1	Write Access created Error
3 (R/NW)	ECCERR	ECC Error. The L2CTL_ET0.ECCERR bit indicates whether the access had an ECC double bit error.	
2 (R/NW)	ACCERR	Access Error. The L2CTL_ET0.ACCERR bit indicates whether the access went to a restricted bank.	
1 (R/NW)	RSVERR	Reserved Error. The L2CTL_ET0.RSVERR bit indicates whether the access went to a reserved location.	
0 (R/NW)	ROMERR	ROM Error. The L2CTL_ET0.ROMERR bit indicates whether a write access went to a ROM area.	

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 be 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.

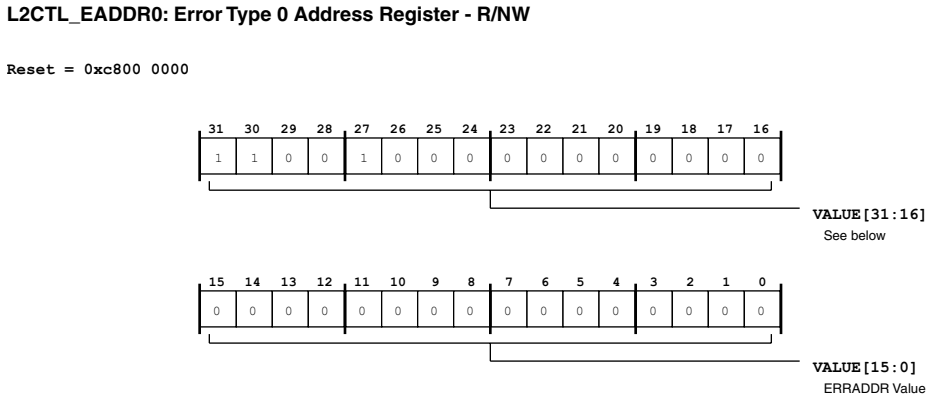


Figure 10-19: L2CTL_EADDR0 Register Diagram

Table 10-24: L2CTL_EADDR0 Register Fields

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

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. 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).

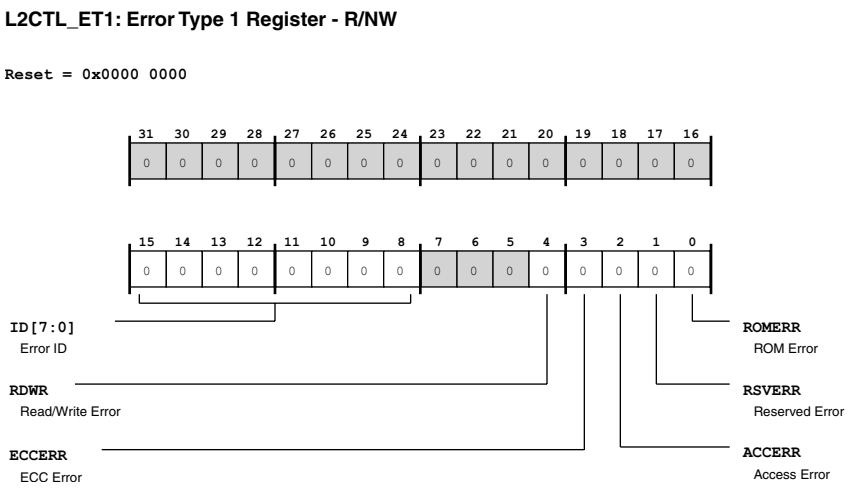


Figure 10-20: L2CTL_ET1 Register Diagram

Table 10-25: L2CTL_ET1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:8 (R/NW)	ID	Error ID. The L2CTL_ET1.ID bits hold the bus master ID of the access that caused error.	
4 (R/NW)	RDWR	Read/Write Error. The L2CTL_ET1.RDWR bit indicates whether a read or write access caused error.	
		0	Read Access created Error
		1	Write Access created Error
3 (R/NW)	ECCERR	ECC Error. The L2CTL_ET1.ECCERR bit indicates whether the access had an ECC double bit error.	
2 (R/NW)	ACCERR	Access Error. The L2CTL_ET1.ACCERR bit indicates whether the access went to a restricted bank.	
1 (R/NW)	RSVERR	Reserved Error. The L2CTL_ET1.RSVERR bit indicates whether the access went to a reserved location.	
0 (R/NW)	ROMERR	ROM Error. The L2CTL_ET1.ROMERR bit indicates whether a write access went to a ROM area.	

Error Type 1 Address Register

The L2CTL_EADDR1 register holds the address that created an access error on the L2 port 0 bus interface. This register is be 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.

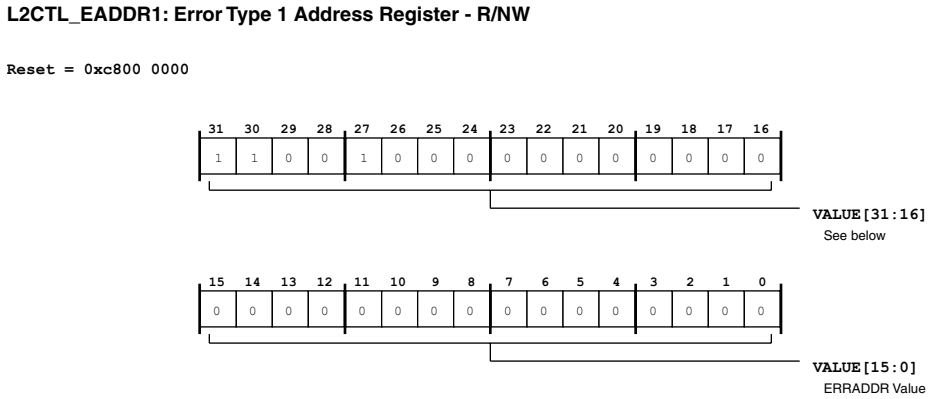


Figure 10-21: L2CTL_EADDR1 Register Diagram

Table 10-26: L2CTL_EADDR1 Register Fields

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

ADSP-BF60x Processor-Specific Information

The L2CLK runs at the same rate as SYSCLK.

The L2 memory subsystem has 2 SCB ports accessing 256 KB of RAM in 8 banks and 32 KB of ROM in a single bank.

ADSP-BF60x L2 Memory Controller Throughput

Table 10-27: ADSP-BF60x Throughput

Master	Condition	Core Throughput (MBPS	DMA Throughput (MBPS
Core Read	Data Cache – burst read (cache fill) single core, data cache enabled. Instruction execution is from L1. Cache fill results in L2 burst access.	540	N/A
Core Read	Data Cache – dual burst read single core (with port preference) (cache fill).	940	N/A
Core Read	Instruction Cache – burst read, single core. Execution from L2 with instruction caching enabled.	550	N/A
Core Read	Instruction Cache and data cache enabled – burst read, single core. Execution from L2 with data access also from L2. Here the instruction and data use must be aligned in such a way that an instruction miss is immediately followed by a data cache miss in the same instruction. This results in back-to-back burst read access to L2.	666	N/A
Core Write	Data Cache – burst write (write back), single core, cache enabled	1028	N/A
Core Write	Non data cache – single write, cache disabled, single writes to L2, single core. (1000 MBPS achieved by running in higher interrupt level)	800, 1000	N/A
DMA Read	No conflict in the memory bank, multiple DMAs initiated to use the same bank, DDE burst length = 8	N/A	880
DMA Read	No conflict in the memory bank, multiple DMAs initiated to use the same bank, DDE burst length = 4	N/A	800
DMA Read	No conflict in the memory bank, multiple DMAs initiated to use the same bank, DDE burst length = 2	N/A	666
DMA Write	No conflict in the memory bank, multiple DMAs initiated to use the same bank, DDE burst length = 8	N/A	880
DMA Write	No conflict in the memory bank, multiple DMAs initiated to use the same bank, DDE burst length = 4	N/A	800
DMA Write	No conflict in the memory bank, multiple DMAs initiated to use the same bank, DDE burst length = 2	N/A	666

Table 10-27: ADSP-BF60x Throughput (Continued)

Master	Condition	Core Throughput (MBPS	DMA Throughput (MBPS
Core/DMA	Core and DMA reading from the same memory bank with RPC0/WPC0 = 0 and RPC1/WPC1 = 1, DDE burst length = 8	1070	885
Core/DMA	Core and DMA writing to the same memory bank with RPC0/WPC0 = 0 and RPC1/WPC1 = 1, DDE burst length = 8	1340	670

11 Dynamic Memory Controller (DMC)

The dynamic memory controller (DMC) provides a glueless interface between DDR2/LPDDR SDRAMs and the system crossbar interface (SCB). The DMC enables execution of instructions from, as well as transfer of data to and from, DDR2 SDRAM or LPDDR 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 re-ordering 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 controller can be accessed using the MMR access bus.

The DMC supports access to the external memory by core and DMA accesses. The external memory address space is divided into four banks.

DMC Features

The DMC includes a protocol controller that supports JESD79-2E compatible double data rate (DDR2) SDRAM and JESD209A low power DDR (LPDDR) SDRAM devices.

The dynamic memory controller features are listed below.

- Provides 16-bit data interface to SDRAM devices
- Supports a single external rank (one chip select)
- Supports burst lengths of 4 and 8 words
- Provides page hit detection to support 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.

The DDR2 features are:

- 256M bit to 2G bit device sizes
- SDRAM data width of 16 (x16 devices only)

- Support for additive latency
- Support for ODT

The LPDDR features are:

- 64M bit to 2G bit device sizes
- SDRAM data width of 16 (x16 devices only)
- Support for deep power down mode
- PHY DLL calibration block
- DDR2 MEMIO I/O buffers
- 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
- Create parameters in the SCB slave interface with integrated buffers that support out of order transaction processing
-

Feature Exclusions

The DMC exclusions are as follows:

For DDR2:

- 4-bit and 8-bit wide DDR2 DRAM memories are not supported.
- OCD is not supported.
- Burst interleaved accesses are not supported.

For LPDDR:

- 32-bit wide LPDDR memory devices are not supported.
- Status register read (SRR) is not supported

- Sampling the optional Temperature output (TQ) signal is not supported.
- Clock Stop mode is not supported
- Bursts of 2 and 16 words are not supported.
- No support for BURST_TERMINATE command.
- Dual-die, two CS# and two CKE packages are not supported.

Functional Description

The Dynamic Memory Controller consists of master and slave interfaces, a protocol controller, and an efficiency controller. These function of these interfaces and controllers are described in the following sections.

ADSP-BF60x DMC Register List

The double data rate-synchronous DRAM (DMC) module provides an interface to external SDRAM. This interface support DDR2 and LPDDR operations. A set of registers govern DMC operations. For more information on DMC functionality, see the DMC register descriptions.

Table 11-1: ADSP-BF60x DMC Register List

Name	Description
DMC_CTL	Control Register
DMC_STAT	Status Register
DMC_EFFCTL	Efficiency Control Register
DMC_PRIO	Priority ID Register
DMC_PRIOMSK	Priority ID Mask Register
DMC_CFG	Configuration Register
DMC_TR0	Timing 0 Register
DMC_TR1	Timing 1 Register
DMC_TR2	Timing 2 Register
DMC_MSK	Mask (Mode Register Shadow) Register
DMC_MR	Shadow MR Register

Table 11-1: ADSP-BF60x DMC Register List (Continued)

Name	Description
DMC_EMR1	Shadow EMR1 Register
DMC_EMR2	Shadow EMR2 Register
DMC_EMR3	Shadow EMR3 Register
DMC_DLLCTL	DLL Control Register
DMC_PHY_CTL1	PHY Control 1 Register
DMC_PHY_CTL3	PHY Control 3 Register
DMC_PADCTL	PAD Control Register

DMC Protocol Controller

The DDR2/LPDDR SDRAM protocol controller translates memory access requests from the SCB (system crossbar) interface to JEDEC protocol specific transactions used by DDR2/LPDDR 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 is capable of issuing reads and writes and it can precharge a row in a bank, activate a row in a bank, and also put the SDRAM devices in self refresh and power down.

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 via a mask register.

DMC 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. A number of schemes can be used to increase the throughput.

Read/Write Turnaround

Read/Write turnaround reduces read – write turnaround on the memory bus and is the default method for optimizing bandwidth.

If read and write commands are outstanding in their respective buffers, the state of the efficiency controller determines the direction of the next transfer. When the controller is in the read state, and if both read and write commands are pending, write commands are given priority. Similarly, if the controller is in the write

state, and if both read and write commands are pending, read commands are given priority. This ensures that the DMC does not perform transfers in a single direction continuously.

If read commands are outstanding in the read command buffer, a snapshot of the entries is taken. Commands that have the same SCB ID are scheduled in the order in which they were received. When the transfers complete, they are removed from the snapshot. The remaining transfers are scheduled in the order in which they appear in the buffer (0, 1, 2, and 3). After all transactions in the snapshot are complete, the controller moves to the write command buffer.

If write commands are outstanding, a snapshot of all valid entries is taken. A valid write entry is one that has the required amount of data available in the write buffer. Commands are scheduled in the order in which they appear in the buffer (0, 1, 2, and 3). After all transactions in the snapshot are complete they are moved to the read command buffer.

For writes and reads that have a SCB burst that is larger than the DDR2 burst length, one SCB burst is divided into several DDR2 bursts. As soon as the required amount of data is available for a DDR2 burst, a write command is scheduled at the protocol controller. Because of write data interleaving, subsequent SCB data continues to be buffered. If data for the next DDR2 burst is not available for an entry in the snapshot after the first pass is complete, the controller does not wait for the data to become available and moves on to the read snapshot.

For example assume that addresses RD1 and RD2 are present the in read buffer and WR1 and WR2 are present in the write buffer. Each of these addresses has a different SCB ID and the efficiency controller is in the write state.

1. The efficiency controller looks into the read buffer, takes a snapshot of all the commands and prepares to send the commands RD1 and RD2 to the protocol controller.
2. The efficiency controller remembers the location in the buffer where these transactions reside (assume locations 1 and 3). It picks location 1 (RD1) to start transfers at the protocol controller.
3. While this transfer happens, if RD3 arrived at location 2, it is ignored for the current snapshot.
4. After RD1 is finished, the controller moves to location 3 (RD2). Once RD2 completes, the controller moves to the write buffers.
5. While the reads occur, WR3 arrives in the WR address buffer. However, since the interleaving depth is only two, WR1 and WR2 data arrive but WR3's data is still being buffered and the required amount of data is not available. In this case, WR1 and WR2 make it to the snapshot as they are valid and have data ready to be sent. Even though WR3 data may arrive during the transfer of WR1 and WR2, it is only considered in the next snapshot.
6. After WR1 and WR2 are complete, the controller moves to the read buffer.
7. Finally, if WR1 requires two bursts through DDR2 and WR2 requires only one, if all data for WR1 is available, the two DDR2 bursts of WR1 are performed back-to-back before completing WR2. However, if only one DDR2 burst of data is available for WR1, the first DDR2 burst of WR1 is performed, followed by WR2. Then the efficiency controller returns to finish the rest of WR1. If WR1 data is not available, the controller does not wait in the write state and moves to the read buffer.

Closed Page Per Bank

The DER_EFFCTL register provides per-bank granularity for closing pages. If software determines that most accesses to a given bank in memory always result in a missed page, the PREC_BANK bit corresponding to the required bank can be set to close the row after every transfer. This proactive step may 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 average request service time can be reduced. However, to service critical requests from any master in the system, a mechanism to elevate priority of a given access is provided. The DMC priority ID register (DMC_PRIO) can be programmed with up to two SCB IDs whose priority is elevated in the default setup described in Read/Write Turnaround.

After every access in a snapshot, the command buffers are searched to determine whether a commands ID matches with the ID programmed in the DMC_PRIO register. If a match occurs and the direction of the access (for example write) is the same as the direction of the snapshot (write), then the priority SCB ID access is sent before sending the subsequent access in the snapshot.

As an alternative to providing priority to a specific SCB ID, if a number of IDs from the same master require priority, the DMC priority mask ID register (DMC_PRIOMSK) can be programmed so that the corresponding bits are 0. A combination of the DMC_PRIO register and the DMC_PRIOMSK register can then be used 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 PRIO_ID1_MASK field of the DMC_PRIOMSK register is set to 0000. If a single ID (7234) needs priority, the PRIO_ID1_MASK field should be set to FFFF and the PRIO_ID1 field of the DMC_PRIO register is set to 7234
- If the PRIO_ID1_MASK field is set to FFFE, the SCB IDs 7234 and 7235 are given priority.
- If the PRIO_ID1_MASK field is set to FFFC, the SCB IDs 7234, 7235, 7236 and 7237 are given priority.
- f two transactions with priority are outstanding with one being a read and the other being a write, 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: SCB-ID-based priority should be used judiciously because it can significantly reduce the throughput of the DMC.

Delaying up to Eight Auto-Refresh Commands

This method is used to ensure that auto-refresh does not come in the way of any critical data transfers. Up to eight auto-refresh commands can be accumulated in the DMC and the exact number of auto-refresh commands can be programmed using the NUM_REF bit in the DMC efficiency control register.

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 (which implies that no access is outstanding) for the programmed number of clock cycles (IDLE_CYCLES) in the DMC efficiency control register, the accumulated number of refresh commands are sent back to back to the DRAM.

After every refresh, the SCB command buffers are checked to ensure they stay empty. However, if the SCB command buffers are always filled, once the programmed number of refresh commands gets accumulated, the refresh operation is elevated to urgent priority and one refresh command is sent immediately. After this, the DMC continues to wait for an opportunity to send out refresh commands. If self-refresh is enabled, all pending refresh commands are given out only after that DMC enters into self-refresh.

Page Interleaving and Bank Interleaving

In this method, the DMC provides a way to allow consecutive row accesses to fall into the same bank (bank interleaving) or into different bank (page interleaving). The ADDR_MODE bit in the DMC_CTL register indicates the interleaving option that the DMC is working on. By default, the DMC uses bank interleaving. If the ADDR_MODE bit is 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 system crossbar slave interface is used to move all data. The system crossbar interface accepts interleaved write transactions and is capable of sending out of order response. The read and write interfaces consist of buffers for address, data and control information transferred to/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 will be accumulated in the system crossbar interface and sent out to protocol controller once the initialization completes.

To increase throughput, system crossbar write response is sent out as soon as the final DDR2 burst is scheduled to be transferred into the SDRAM. However, if an auto-refresh needs to be performed, the scheduled write data will be sent only after the auto-refresh. There is a possibility of a delay of a maximum of 64 clock cycles from the moment write response is sent on system crossbar to the actual write of the data into SDRAM.

The system crossbar interface performs the following operations.

- Buffers read/write command requests from system crossbar bus.
- Processes the requests by converting them to protocol controller user interface transfers.

- Sends and receives data to/from the protocol controller.
- Creates 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, 16 or 32-bits
- arrival of write data before write address
- generation of error responses which includes
  - any access to un-implemented region of the external memory space
  - any access when the SDRAM is in self-refresh, power-down or deep power down (in case of LPDDR)
  - any access when the direct command interface is in operation

Read/Write Command and Data Buffers

The system crossbar interface comprises 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
- Ability to extend a transfer using PREADY
- Generation of PSLVERR when unimplemented registers are accessed or when read-only registers are written.

Architectural Concepts

The following sections provide information on the architecture of the interface.

DMC Clocking

The DMC controller uses a divided down version of the PLLCLK (PLL clock) to generate an internal clock used to clock the DMC block and interface. The specific value of the DCLK frequency is programmed in the CGU_DIV register, and the procedure is highlighted under the section "Changing the DCLK Clock Frequency".

The maximum clock frequency is 250 MHz if interfacing to DDR2 SDRAM, and 200 MHz if interfacing to LPDDR SDRAM.

DMC DMA

The DMC controller supports DMA-based transfers to and from external DDR2/LPDDR 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.

Two DDEs are used for memory-to-memory DMA (MemDMA). One channel is the source channel, and the second, the destination channel.

DMA transfers on the processor can be descriptor-based or register-based. Register-based DMA allows the processor 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, if needed. Descriptor-based DMA transfers require a set of parameters stored within memory to initiate a DMA sequence. This sort of transfer allows the chaining together of multiple DMA sequences. In descriptor-based DMA operations, a DMA channel can be programmed to automatically set up and start another DMA transfer after the current sequence completes.

Please refer to the DDE chapter for further details.

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. These registers are accessed using the MMR access bus and are described below.

- Control and status registers. These registers control the various operation modes of the dynamic memory controller and also provide status.
- Timing parameter registers. The value programmed in these registers depends on the speed grade of DDR2 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 – These registers are used to control the operation of the PHY.
- PAD control registers – These registers are used 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 may cause the dynamic memory controller to function erroneously.

Configuring the DMC

PREREQUISITE:

After a processor's hardware or software reset, the DMC clocks are enabled. However the DMC must be configured and initialized before any data transfer can take place. Before programming the DMC and executing the power up (initialization) sequence, ensure that the clock to the SDRAM is enabled after the power has stabilized for the proper amount of time (as specified by the SDRAM specification).

1. Check to first ensure that the DMC is idle and not in the midst of any activity.
2. Please the DMC in self-refresh mode.
3. Program the PLL frequency to a new value (if required).
4. Wait the appropriate number of core cycles to ensure that the DLL has locked.
5. Bring the DMC out of self-refresh mode.
6. Program the `DMC_CFG`, `DMC_CTL`, `DMC_TRO`,-`DMC_TR2DMC_MR`, `DMC_PHY_CTL1` and `DMC_PHY_CTL3` registers to the appropriate values, to set proper SDRAM cycle timing options (for example t_{RAS}, t_{RC}, t_{RP}, t_{RCD}, t_{WR}, t_{FAW} are some of the parameters).
7. Program the shadow registers `DMC_EMR1-DMC_EMR3`, with the required burst length, CAS latency, additive latency and other parameters.
8. Finally, after these registers are programmed, write the `INIT` bit to the DMC control register (0x0004) to begin the power-up initialization sequence.
9. Wait for the SDRAM initialization sequence to complete.
10. Write the `DMC_PADCTL` register with values that reflect the type of SDRAM connected to the processor. Specific bit fields within this register determine if the pad is operating in LPDDR mode or DDR2 mode. The drive strengths required at the pads can also be programmed in this register.

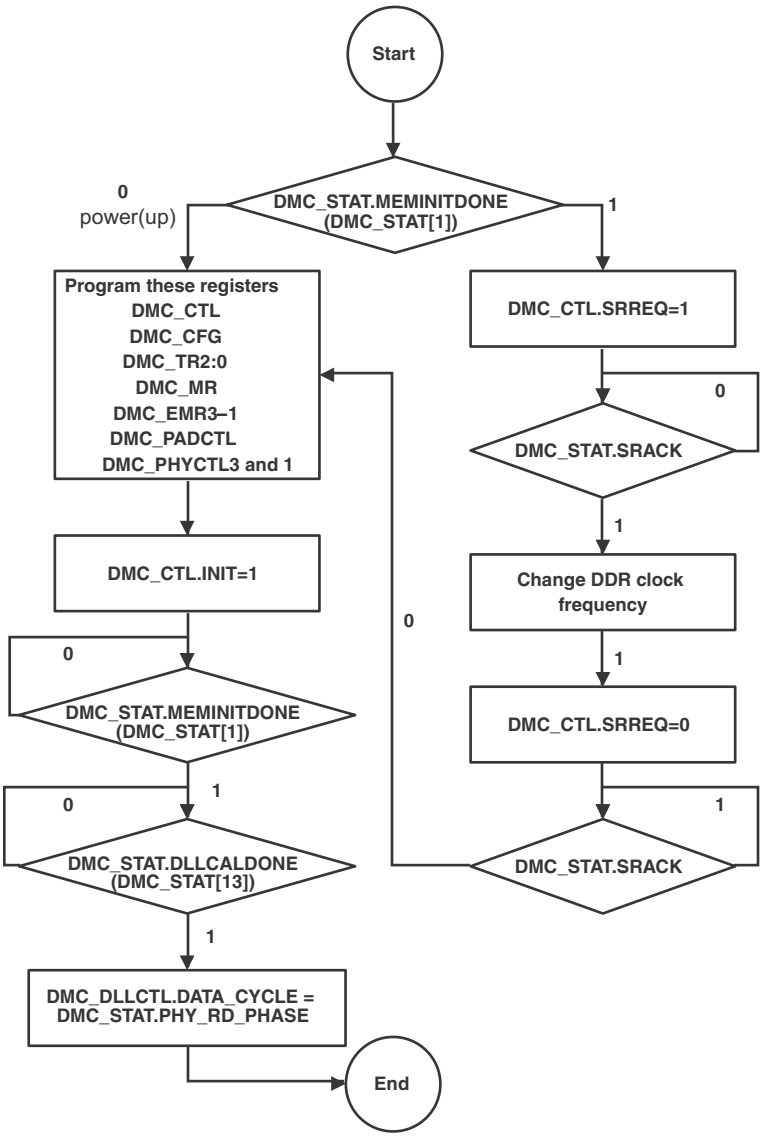


Figure 11-1: DMC Initialization Flow

System crossbar transactions that occur during or before initialization are accumulated by the DMC and sent to SDRAM once the SDRAM initialization and/or DLL calibration is complete.

DLL Initialization

When initializing the DLL for the first time, it is important to follow the following steps:

1) Check for (MEMINITDONE & DLLCALDONE) in the DMC_STAT register

2) If this is NOT set, set init bit in DMC_CTL register and wait for (MEMINITDONE & DLLCALDONE) bits to be set in the DMC_STAT register

3) If this is set:

a) Enter into self-refresh

b) Change the clock frequency, as required

c) Wait for DLL lock

d) Exit self refresh

e) Write new control register values

f) Set init bit in DMC_CTL register and poll the MEMINITDONE and DLLCALDONE bits within the DMC_STAT register

g) Write to the DMC_DLLCTL register.

Note that for cases where the DDR2 interface has already been initialized (whether it is via an XML file loaded in during a debug session, or through code executed during the booting process), the user needs to perform second-time initialization as described above.

Saving Power with the DMC

This section discusses the suggested flow to enter and exit DDR self-refresh before and after the processor enters the HIBERNATE state.

For this procedure, the system is in normal operation and the EXT_WAKE signal is high.

1. Put the SDRAM in self-refresh mode by setting the DMC_CTL.SRREQ bit.

   STEP RESULT: The DDR goes through the self-refresh entry sequence and enters the Self Refresh state.

   STEP RESULT: The DMC_STAT.SRACK bit is set.

   STEP RESULT: The CKE pin is driven low by the controller when the DDR has entered Self-Refresh.

2. Read the DMC_DLLCTL register to get the current DLL tap and calibration settings. This can be used later to quickly lock and start normal DDR operation. The values read are stored in the DPM registers.

3. Initialize the Power-On Reset Delay register to the appropriate values to count off the time required for core V_{DD} to reach a safe value when exiting the Hibernate state.

4. Enter the Hibernate state by following the procedure detailed in the DPM chapter. Hibernate is indicated by the EXT_WAKE signal going low.

5. When EXT_WAKE goes low, the part enters Hibernate state and remains here until brought out by this state through SYS_PWRGD pin/counter expiry.

6. When a wake-up event occurs, first the EXT_WAKE signal goes high.

   ADDITIONAL INFORMATION: When core V_{DD} power reaches a proper value, the Core domain logic is reset. The DDR controller drives the input of the CKE pad low. When this counter reaches 0 a Counter

Expiry signal is generated and Core V_{DD} is deemed to have reached a safe value. Now, the CKE pin which is driven by the DDR controller is released, so whatever is driven by the DDR controller (at this time driven to 0) at its input pin is driven out to the pads.

7. The DDR controller acts as if it has just come out of reset. The software should now program the DMC_CTL.SRREQ bit to write a 1. The DMC_CTL.INIT bit should not be set during this write, because starting an INIT sequence is not desired.

   ADDITIONAL INFORMATION: The controller interprets this Self-Refresh request as a command to directly jump to the Self-Refresh state in the state machine.

   ADDITIONAL INFORMATION: Once the DDR state machine goes to the Self-Refresh state, it should set up all the state variables and status indications to appropriate values. It should appear as though the state machine just entered a Self-Refresh state (as in step 2 above).

   ADDITIONAL INFORMATION: Further, the INIT_DONE bit is now set to 1 (programs should not be performing an INIT now because it was already done before Hibernate) and also the SRACK bit is set to 1.

8. Software retrieves the DLL data stored in the HV domain and programs the DMC_DLLCTL register. This acts as a guess value for the DLL lock and DLL calibration and shortens the time taken to achieve DLL lock.

9. The software should poll for DLL lock status to go high and then clear the SRREQ bit in the DDR control register which kick starts a Self-Refresh exit process.

10. After Self-Refresh exit (indicated by the DMC_STAT.SRACK bit), start DLL calibration by writing into the DMC_CTL.DLLCAL bit. The number of reads required for calibration this time are less than normal power up (cold start) calibration, since the guess value for this has been programmed in the DMC_DLLCTL register.

RESULT:

Once the calibration process is complete, the DMC_STAT.DLLCALDONE bit is set. Normal operations to DDR memory can now start.

ADSP-BF60x DMC Register Descriptions

DDR (DMC) contains the following registers.

Table 11-2: ADSP-BF60x DMC Register List

Name	Description
DMC_CTL	Control Register
DMC_STAT	Status Register

Table 11-2: ADSP-BF60x DMC Register List (Continued)

Name	Description
DMC_EFFCTL	Efficiency Control Register
DMC_PRIO	Priority ID Register
DMC_PRIOMSK	Priority ID Mask Register
DMC_CFG	Configuration Register
DMC_TR0	Timing 0 Register
DMC_TR1	Timing 1 Register
DMC_TR2	Timing 2 Register
DMC_MSK	Mask (Mode Register Shadow) Register
DMC_MR	Shadow MR Register
DMC_EMR1	Shadow EMR1 Register
DMC_EMR2	Shadow EMR2 Register
DMC_EMR3	Shadow EMR3 Register
DMC_DLLCTL	DLL Control Register
DMC_PHY_CTL1	PHY Control 1 Register
DMC_PHY_CTL3	PHY Control 3 Register
DMC_PADCTL	PAD Control Register

Control Register

The DMC_CTL register controls DMC modes, DLL calibration, and DRAM initialization.

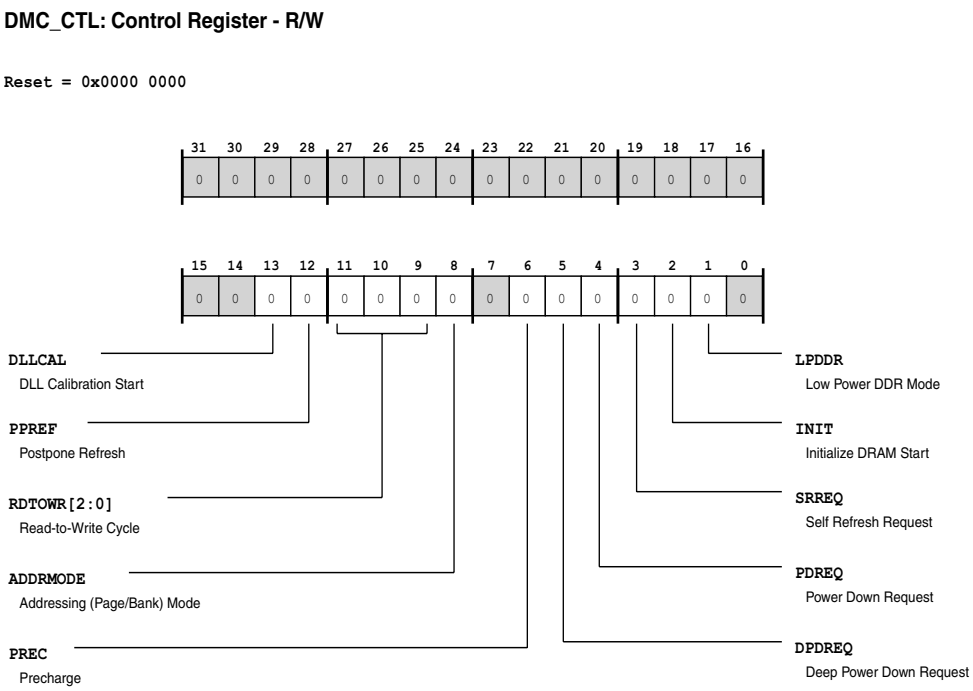


Figure 11-2: DMC_CTL Register Diagram

Table 11-3: DMC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
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.	
		0	Disable Postpone Refresh
		1	Enable Postpone Refresh

Table 11-3: DMC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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. Note that values 101 through 111 are reserved.	
		0	0 Cycles Added
		1	1 Cycle Added
		2	2 Cycles Added
		3	3 Cycles Added
		4	4 Cycles Added
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
6 (R/W)	PREC	Precharge. The DMC_CTL.PREC bit enables pre-charge, 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
5 (R/W)	DPDREQ	Deep Power Down Request. The DMC_CTL.DPDREQ bit enables deep powerdown mode if low power DMC operation is enabled (DMC_CTL.LPDDR =1}). When the processor does not require the data stored in SDRAM (assume reset state of SDRAM), the DMC may put the SDRAM in deep powerdown mode. When the DMC is in deep powerdown, any data accesses cause the DMC to generate a bus error.	
		0	Disable Deep Powerdown
		1	Enable Deep Powerdown

Table 11-3: DMC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	PDREQ	Power Down Request. The DMC_CTL.PDREQ bit enables powerdown mode. When the DMC is in powerdown, any data accesses cause the DMC to generate a bus error.	
		0	Disable Powerdown
		1	Enable Powerdown
3 (R/W)	SRREQ	Self Refresh Request. The DMC_CTL.SRREQ bit enables self refresh mode. When the DMC is in self-refresh, any data accesses cause the DMC to generate a bus error.	
		0	Disable Self Refresh
		1	Enable Self Refresh
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
1 (R/W)	LPDDR	Low Power DDR Mode. The DMC_CTL.LPDDR bit selects whether the DMC operates in low power DDR mode or DDR2 mode.	
		0	DDR2 mode
		1	LPDDR mode

Status Register

The DMC_STAT register indicates status for modes selected with the DMC_CTL register and indicates status DMC operations.

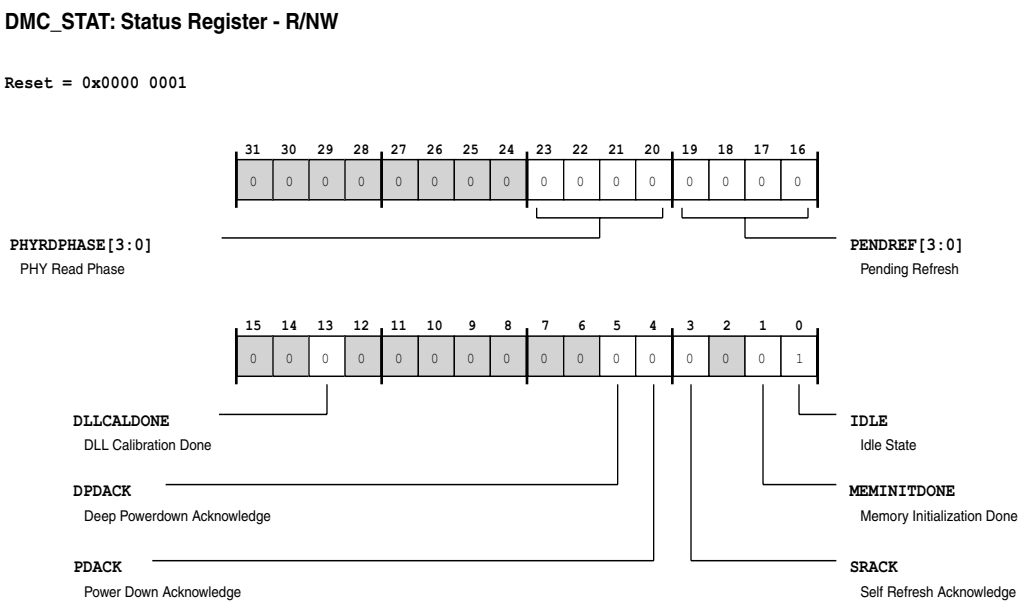


Figure 11-3: DMC_STAT Register Diagram

Table 11-4: DMC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:20 (R/NW)	PHYRDPHASE	PHY Read Phase. The DMC_STAT.PHYRDPHASE 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
		3	3 Clock Cycles Latency
		4	4 Clock Cycles Latency
		5	5 Clock Cycles Latency
19:16 (R/NW)	PENDREF	Pending Refresh. The DMC_STAT.PENDREF bits indicate the number of pending auto-refresh commands. When the DMC is in low power DDR mode (DMC_CTL.LPDDR =1), the maximum value for DMC_STAT.PENDREF is 3.	

Table 11-4: DMC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/NW)	DLLCALDONE	DLL Calibration Done. The DMC_STAT.DLLCALDONE indicates that the PHY DLL calibration sequence is complete.	
		0	No Status
		1	Completed PHY DLL Calibration
5 (R/NW)	DPDACK	Deep Powerdown Acknowledge. The DMC_STAT.DPDACK bit indicates that deep powerdown mode is active. Note that this status is available in low power DDR mode (DMC_CTL.LPDDR =1) only.	
		0	Not in Deep Powerdown Mode
		1	Deep Powerdown Mode Active
4 (R/NW)	PDACK	Power Down Acknowledge. The DMC_STAT.PDACK bit indicates that powerdown mode is active.	
		0	Not in Powerdown Mode
		1	Powerdown Mode Active
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
1 (R/NW)	MEMINITDONE	Memory Initialization Done.	
		0	Init not done
		1	Init complete
0 (R/NW)	IDLE	Idle State. The DMC_STAT.IDLE bit indicates whether the DMC is idle or busy.	
		0	Busy
		1	Idle

Efficiency Control Register

The DMC_EFFCTL register control DMC features that improve throughput efficiency. These include features such as auto-refresh management, pre-charge options, and write data options.

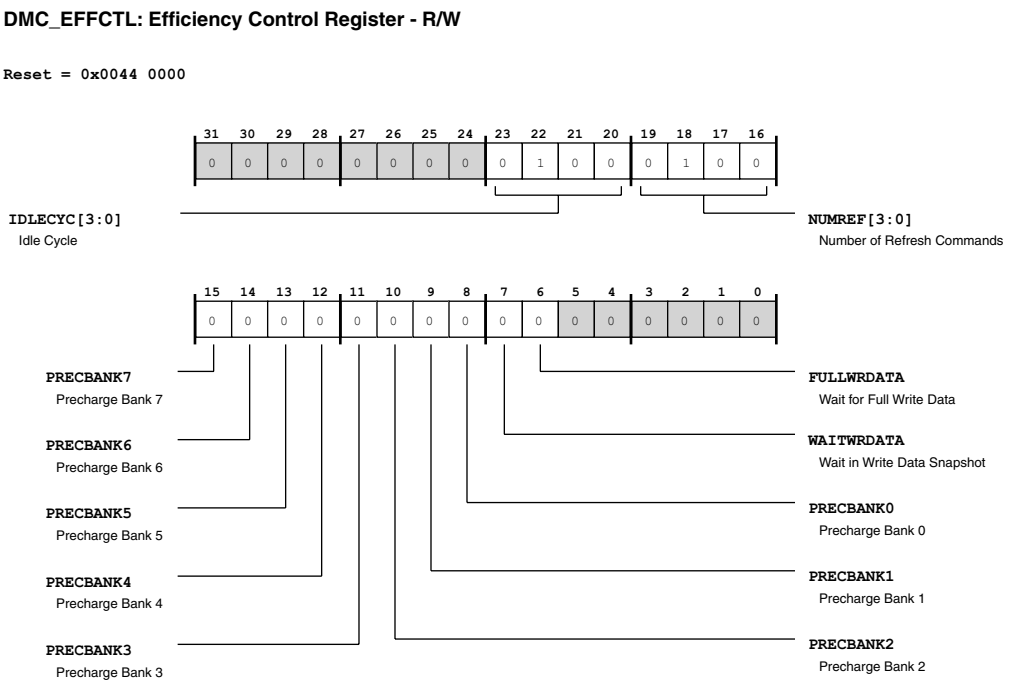


Figure 11-4: DMC_EFFCTL Register Diagram

Table 11-5: DMC_EFFCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:20 (R/W)	IDLECYC	Idle Cycle. The DMC_EFFCTL.IDLECYC bits select the number of cycles after which the DMC issues any accumulated auto-refresh commands if postpone refresh is enabled (DMC_CTL.PPREF =1). When DMC_EFFCTL.IDLECYC is set to 0, the DMC ignores the DMC_CTL.PPREF selection and does not accumulate/postpone periodic auto refresh commands.	
		xxxx	Idle Cycles to Postpone Refresh Commands

Table 11-5: 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 may accumulate if postpone refresh is enabled (DMC_CTL.PPREF =1). The number of auto-refresh commands that may be accumulated depends on whether the DMC is in DDR2 or LPDDR mode as selected by the DMC_CTL.LPDDR bit. In LPDDR mode, the DMC may accumulate up to four auto-refresh commands. In DDR2 mode, the DMC may accumulate up to eight auto-refresh commands.	
		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).	
		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).	
		0	Disable Precharge Bank 6
		1	Enable Precharge Bank 6

Table 11-5: 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).	
		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).	
		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).	
		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)	
		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).	
		0	Disable Precharge Bank 1
		1	Enable Precharge Bank 1

Table 11-5: 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).	
		0	Disable Precharge Bank 0
		1	Enable Precharge Bank 0
7 (R/W)	WAITWRDATA	Wait in Write Data Snapshot. The DMC_EFFCTL.WAITWRDATA bit enables waiting in write snapshot if the DDR2 or LPDDR burst is not available for the transfer. If disabled, the DMC does not wait in write snapshot when the burst is unavailable.	
		0	Disable Wait for Burst
		1	Enable Wait for Burst
6 (R/W)	FULLWRDATA	Wait for Full Write Data. The DMC_EFFCTL.FULLWRDATA bit enables waiting until all data is available from the bus to start the DMC transfer. If disabled, the DMC does to wait for full write data, instead the DMC starts the transfer when DDR2 or LPDDR burst becomes available.	
		0	Disable Wait for Full Data
		1	Enable Wait for Full Data

Priority ID Register

The DMC_PRIO register selects up to two internal bus master IDs for banks to receive elevated access priority by the DMC efficiency controller.

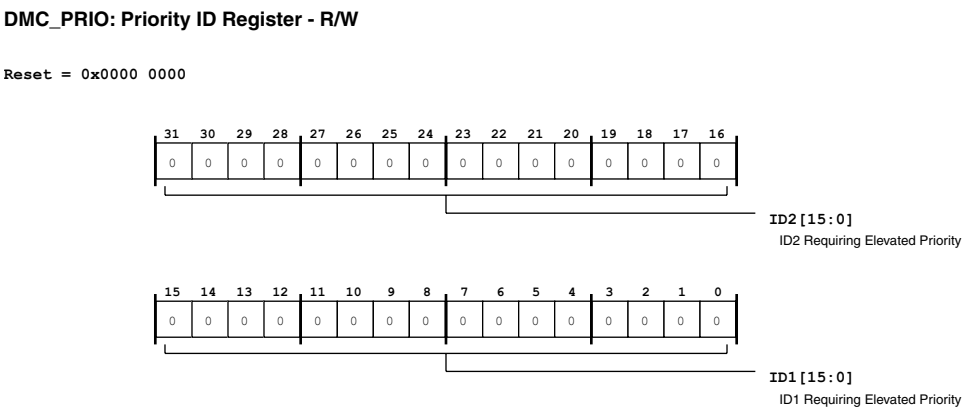


Figure 11-5: DMC_PRIO Register Diagram

Table 11-6: DMC_PRIO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	ID2	ID2 Requiring Elevated Priority.
15:0 (R/W)	ID1	ID1 Requiring Elevated Priority.

Priority ID Mask Register

The DMC_PRIOMSK register permits masking portions of up to two internal bus master IDs for banks to receive elevated access priority by the DMC efficiency controller. This masking provides for elevating the access priority of either a single ID or a range of IDs.

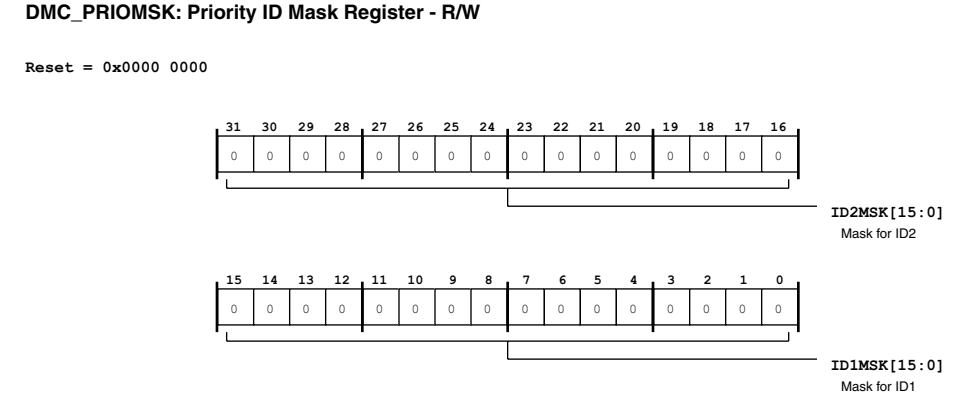


Figure 11-6: DMC_PRIOMSK Register Diagram

Table 11-7: DMC_PRIOMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	ID2MSK	Mask for ID2.
15:0 (R/W)	ID1MSK	Mask for ID1.

Configuration Register

The DMC_CFGregister selects SDRAM device specific parameters and selects the SDRAM interface width.

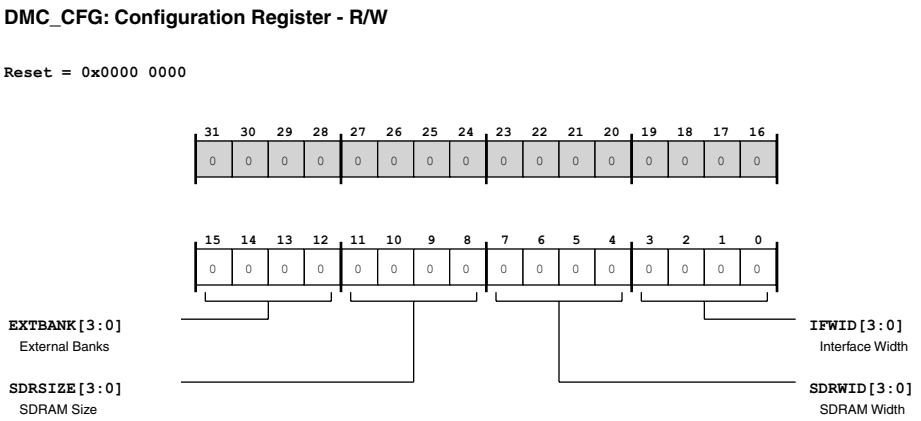


Figure 11-7: DMC_CFG Register Diagram

Table 11-8: DMC_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:12 (R/W)	EXTBANK	External Banks. The DMC_CFG.EXTBANK 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

Table 11-8: DMC_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11:8 (R/W)	SDRSIZE	SDRAM Size. The DMC_CFG.SDRSIZE bits select the size of individual SDRAM connected to the DMC. Note that all values other than those shown are reserved.	
		0	64M Bit SDRAM (LPDDR Only)
		1	128M Bit SDRAM (LPDDR Only)
		2	256M Bit SDRAM
		3	512M Bit SDRAM
		4	1G Bit SDRAM
		5	2G Bit SDRAM
7:4 (R/W)	SDRWID	SDRAM Width. The DMC_CFG.SDRWID bits select the width of the individual SDRAM connected to the DMC. Note that all values other than those shown are reserved.	
		2	16-Bit Wide SDRAM
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.	
		2	16-Bit Wide Interface All Other Values: Reserved This field specifies the interface width between the controller and the SDRAM.

Timing 0 Register

The DMC_TRO 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_TRO are in increments of clock cycle time (t_{CK}).

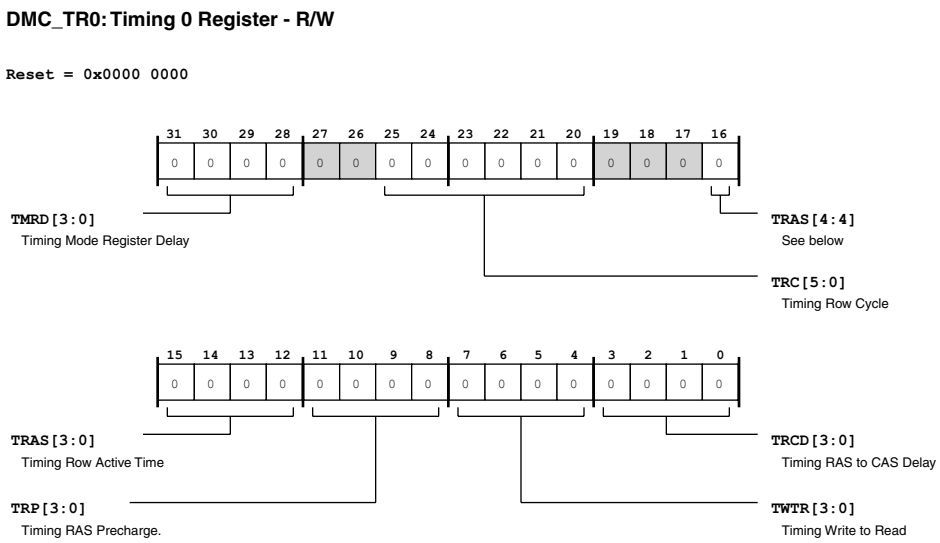


Figure 11-8: DMC_TR0 Register Diagram

Table 11-9: DMC_TR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:28 (R/W)	TMRD	Timing Mode Register Delay. The DMC_TR0.TMRD 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 DMC_TR0.TRC 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 DMC_TR0.TRAS 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 DMC_TR0.TRP 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 11-9: 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}).

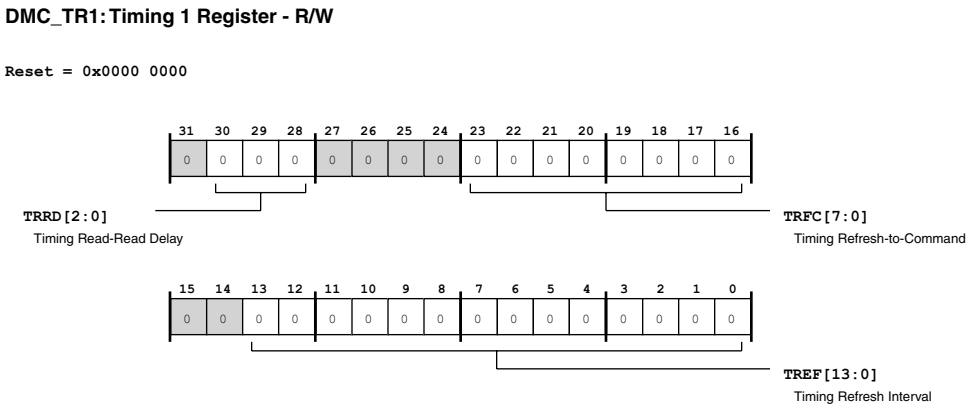


Figure 11-9: DMC_TR1 Register Diagram

Table 11-10: DMC_TR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30:28 (R/W)	TRRD	Timing Read-Read Delay. The DMC_TR1.TRRD 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.

Table 11-10: DMC_TR1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	TRFC	Timing Refresh-to-Command. The DMC_TR1.TRFC 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 DMC_TR1.TREF 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 must be programmed before initializing the SDRAM Note that all values for bit fields in DMC_TR2 are in increments of clock cycle time (t_{CK}).

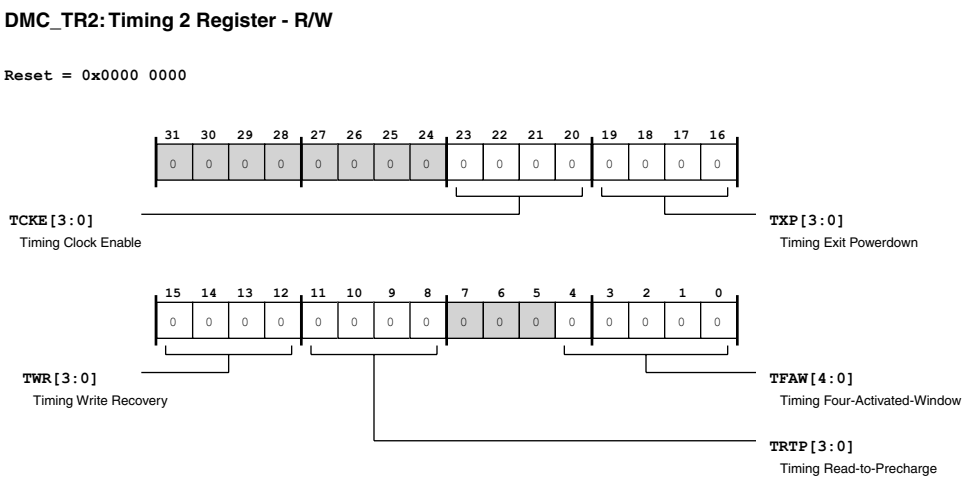


Figure 11-10: DMC_TR2 Register Diagram

Table 11-11: DMC_TR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:20 (R/W)	TCKE	Timing Clock Enable. The DMC_TR2.TCKE field selects the CKE minimum pulsewidth (t_{CKE}).
19:16 (R/W)	TXP	Timing Exit Powerdown. The DMC_TR2.TXP field selects the exit powerdown to next valid command time (t_{XP}).
15:12 (R/W)	TWR	Timing Write Recovery. The DMC_TR2.TWR field selects the write recovery time (t_{WR}). Note that this parameter applies to LPDDR only.
11:8 (R/W)	TRTP	Timing Read-to-Precharge. The DMC_TR2.TRTP field selects the internal read to precharge time (t_{RTP}).
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.

Mask (Mode Register Shadow) Register

The DMC_MSK register permits masking (disabling) writes to the MR and EMRn registers in the SDRAM. When masked, writes to these registers go instead to shadow copies of these registers (DMC_MR, DMC_EMR1, DMC_EMR2, and DMC_EMR3), 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).

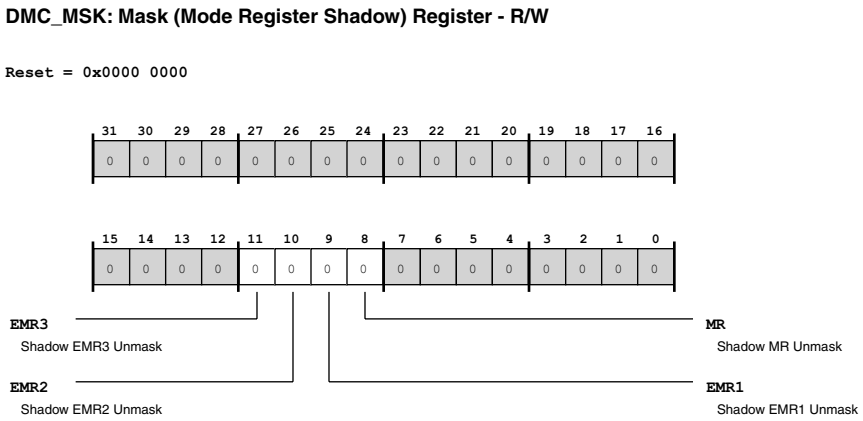


Figure 11-11: DMC_MSK Register Diagram

Table 11-12: DMC_MSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	EMR3	Shadow EMR3 Unmask. The DMC_MSK.EMR3 bit masks or unmasks writes to the EMR3 register (in DDR2) in the SDRAM. When masked, writes to this register instead go to the DMC_EMR3 register. When unmasked, the DMC writes the DMC_EMR3 value to the EMR3 register (in DDR2) in the SDRAM. After completing the write, the DMC clears this bit. Note that this bit must not be enabled when in LPDDR mode (DMC_CTL.LPDDR =1).	
		0	Mask (Disable) Write to EMR3
		1	Unmask (Enable) Write to EMR3
10 (R/W)	EMR2	Shadow EMR2 Unmask. The DMC_MSK.EMR2 bit masks or unmasks writes to the EMR2 register (in DDR2) or the EMR register (in LPDDR) in the SDRAM. When masked, writes to this register instead go to the DMC_EMR2 register. When unmasked, the DMC writes the DMC_EMR2 value to the EMR2 register (in DDR2) or the EMR register (in LPDDR) in the SDRAM. After completing the write, the DMC clears this bit.	
		0	Mask (Disable) Write to EMR2
		1	Unmask (Enable) Write to EMR2
9 (R/W)	EMR1	Shadow EMR1 Unmask. The DMC_MSK.EMR1 bit masks or unmasks writes to the EMR1 register in the SDRAM. When masked, writes to this register instead go to the DMC_EMR1 register. When unmasked, the DMC writes the DMC_EMR1 value to the EMR1 register in the SDRAM. After completing the write, the DMC clears this bit. Note that this bit must not be enabled when in LPDDR mode (DMC_CTL.LPDDR =1).	
		0	Mask (Disable) Write to EMR1
		1	Unmask (Enable) Write to EMR1
8 (R/W)	MR	Shadow MR Unmask. The DMC_MSK.MR bit masks or unmasks writes to the MR register in the SDRAM. When masked, writes to this register instead go to the DMC_MR register. When unmasked, the DMC writes the DMC_MR 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

Shadow MR Register

The DMC_MR register in the DMC shadows the MR register in the SDRAM when the DMC is in DDR2 mode or LPDDR mode (DMC_CTL.LPDDR =0 or =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.

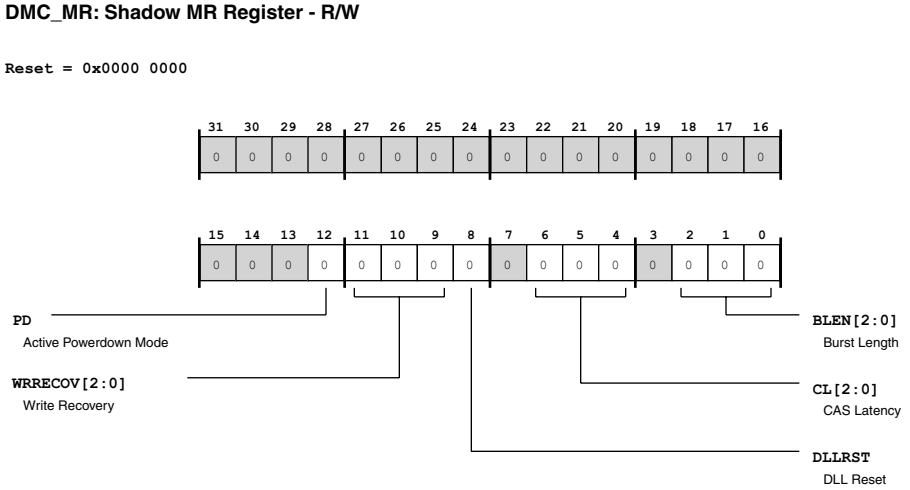


Figure 11-12: DMC_MR Register Diagram

Table 11-13: DMC_MR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	PD	Active Powerdown Mode. The DMC_MR.PD bit selects the active powerdown mode. Note that this parameter applies only for DDR2 mode and is reserved for LPDDR 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)

Table 11-13: DMC_MR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11:9 (R/W)	WRRECOV	Write Recovery. The DMC_MR.WRRECOV bit selects the write recovery time in terms of clock cycles (t_{CK}). Note that this parameter applies only for DDR2 mode and is reserved for LPDDR mode. For more information about this mode, see the data sheet for the SDRAM being used in your system.	
		1	2 Clock Cycles
		2	3 Clock Cycles
		3	4 Clock Cycles
		4	5 Clock Cycles
		5	6 Clock Cycles
		6	7 Clock Cycles
		7	8 Clock Cycles
8 (R/W)	DLLRST	DLL Reset. The DMC_MR.DLLRST bit initiates a DLL reset on the SDRAM. Note that this parameter applies only for DDR2 mode and is reserved for LPDDR mode. For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	Normal Operation
		1	Reset DLL
6:4 (R/W)	CL	CAS Latency. The DMC_MR.CL bits 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 (t_{CK}). For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		2	2 clock cycle latency
		3	3 clock cycle latency
		4	4 clock cycle latency (DDR2)
		5	5 clock cycle latency (DDR2)
		6	6 clock cycle latency (DDR2)

Table 11-13: DMC_MR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W)	BLEN	Burst Length. 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.	
		2	4-Bit Burst Length
		3	8-Bit Burst Length

Shadow EMR1 Register

The DMC_EMR1 register in the DMC shadows the EMR1 register in the SDRAM when the DMC is in DDR2 mode (DMC_CTL.LPDDR =0). Note that this register must not be used when the DMC is in LPDDR mode (DMC_CTL.LPDDR =1). If unmasked by the corresponding bit in the shadow mask register (DMC_MSK.EMR1 =1), a write to DMC_EMR1 triggers an extended "mode register set" command on the memory interface. If masked, a write to DMC_EMR1 only updates the register in the DMC, not the register in the SDRAM.

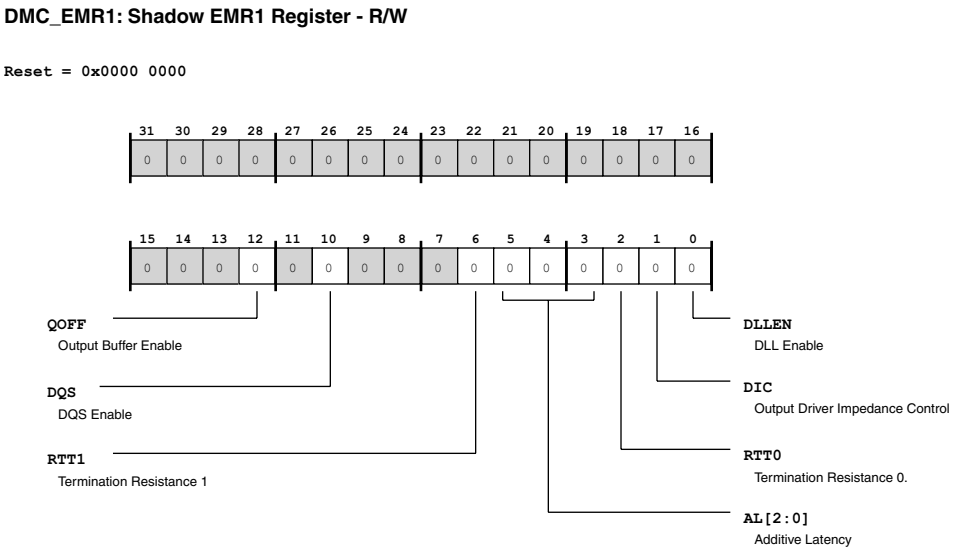


Figure 11-13: DMC_EMR1 Register Diagram

Table 11-14: DMC_EMR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	QOFF	Output Buffer Enable. The DMC_EMR1.QOFF 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	Enable
		1	Disable
10 (R/W)	DQS	DQS Enable. The DMC_EMR1.DQS bit enables operation of the DQS pin. For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	Enable
		1	Disable
6 (R/W)	RTT1	Termination Resistance 1. The DMC_EMR1.RTT1 bit combines with the DMC_EMR1.RTT0 bit to set the termination resistance. See the DMC_EMR1.RTT0 bit description for more information.	
		0	Disable RTT1
		1	Enable RTT1
5:3 (R/W)	AL	Additive Latency. The DMC_EMR1.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	0 Clock Cylces Added
		1	1 Clock Cylce Added
		2	2 Clock Cylces Added
		3	3 Clock Cylces Added
		4	4 Clock Cylces Added
		5	5 Clock Cylces Added

Table 11-14: DMC_EMR1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	RTT0	Termination Resistance 0.. The DMC_EMR1.RTT0 bit and the DMC_EMR1.RTT1 bits select the SDRAM termination resistance. RTT1=0, RTT0=0 : No ODT at memory device RTT1=0, RTT0=1 : 75 Ohm ODT at memory device RTT1=1, RTT0=0 : 150 Ohm ODT at memory device RTT1=1, RTT0=1 : 50 Ohm ODT at memory device For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	Disable RTT0
		1	Enable RTT0
1 (R/W)	DIC	Output Driver Impedance Control. The DMC_EMR1.DIC bit selects the drive strength mode for the SDRAM. For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	Full Strength
		1	Reduced Strength
0 (R/W)	DLLEN	DLL Enable. The DMC_EMR1.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 DLL (Normal Operation)
		1	Disable DLL (Test/Debug Operation)

Shadow EMR2 Register

The DMC_EMR2 register in the DMC shadows the EMR2 register in the SDRAM when the DMC is in DDR2 mode (DMC_CTL.LPDDR =0) and shadows the EMR register in the SDRAM when the DMC is in LPDDR mode (DMC_CTL.LPDDR =1). If unmasked by the corresponding bit in the shadow mask register (DMC_MSK.EMR2 =1), a write to DMC_EMR2 triggers an extended "mode register set" command on the memory interface. If masked, a write to DMC_EMR2 only updates the register in the DMC, not the register in the SDRAM.

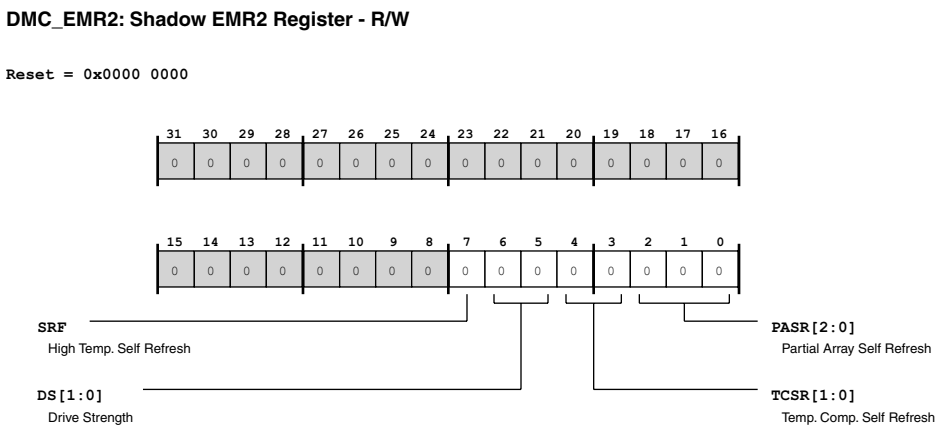


Figure 11-14: DMC_EMR2 Register Diagram

Table 11-15: DMC_EMR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	SRF	High Temp. Self Refresh. The DMC_EMR2.SRF bit enables the SDRAM's high temperature self refresh rate feature when the DMC is in DDR2 mode. (This bit is reserved in LPDDR mode.) For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	1x Refresh Rate (0C to 85C)
		1	2x Refresh Rate (>85C)
6:5 (R/W)	DS	Drive Strength. The DMC_EMR2.DS bits select the drive strength value when the DMC is in LPDDR mode. (These bits are reserved when the DMC is in DDR2 mode.) Note that all values other than those shown are reserved. For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	Full Drive Strength
		1	1/2 Drive Strength
		2	1/4 Drive Strength
		3	3/4 Drive Strength

Table 11-15: DMC_EMR2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4:3 (R/W)	TCSR	Temp. Comp. Self Refresh. The DMC_EMR2.TCSR bits select the temperature for applying temperature compensated self refresh when the DMC is in LPDDR mode. (These bits are reserved when the DMC is in DDR2 mode.) For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	70 degree C (in LPDDR Mode)
		1	45 degree C
		2	15 degree C
		3	85 degree C
2:0 (R/W)	PASR	Partial Array Self Refresh. The DMC_EMR2.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	Full Array (in LPDDR Mode)
		1	1/2 Array
		2	1/4 Array
		3	Reserved
		4	Reserved
		5	1/8 Array
		6	1/16 Array
		7	Reserved

Shadow EMR3 Register

The DMC_EMR3 register in the DMC shadows the EMR3 register in the SDRAM when the DMC is in DDR2 mode (DMC_CTL.LPDDR =0). Note that this register must not be used when the DMC is in LPDDR mode (DMC_CTL.LPDDR =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.

DMC_EMR3: Shadow EMR3 Register - R/W

Reset = 0x0000 0000

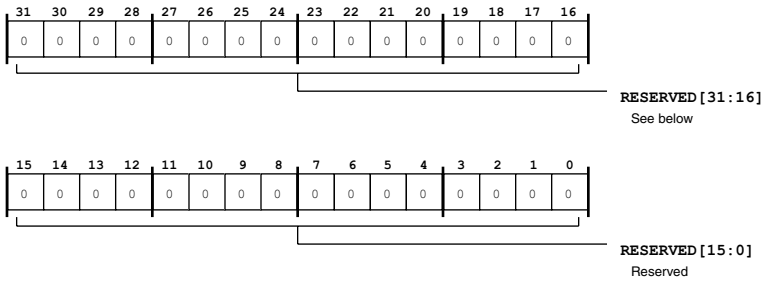


Figure 11-15: DMC_EMR3 Register Diagram

Table 11-16: DMC_EMR3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	RESERVED	Reserved. All bits the DMC_EMR3 register are reserved.

DLL Control Register

The DMC_DLLCTL register holds the programmable parameters associated with the DLLs within the DMC PHY.

DMC_DLLCTL: DLL Control Register - R/W

Reset = 0x0000 034b

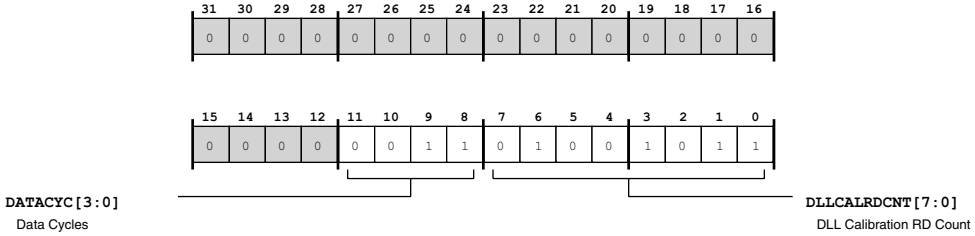


Figure 11-16: DMC_DLLCTL Register Diagram

Table 11-17: DMC_DLLCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:8 (R/W)	DATACYC	Data Cycles. The DMC_DLLCTL.DATACYC bits select the latency after which the DMC reads data from the PHY. This field must be written with the value indicated in the DMC_STAT.PHYRDPHASE field, or data corruption occurs on all SDRAM reads. Taking round trip delay into account, the DLL indicates whether a latency of 2 cycles is supported by means of status bits.	
		2	2 Clock Cycles Latency
		3	3 Clock Cycles Latency
		4	4 Clock Cycles Latency
		5	5 Clock Cycles Latency
7:0 (R/W)	DLLCALRDCNT	DLL Calibration RD Count. The DMC_DLLCTL.DLLCALRDCNT field selects the number of read operations that the PHY uses for DLL calibration.	

PHY Control 1 Register

The DMC_PHY_CTL1 register controls programmable PHY features.

DMC_PHY_CTL1: PHY Control 1 Register - R/W

Reset = 0x0000 0000

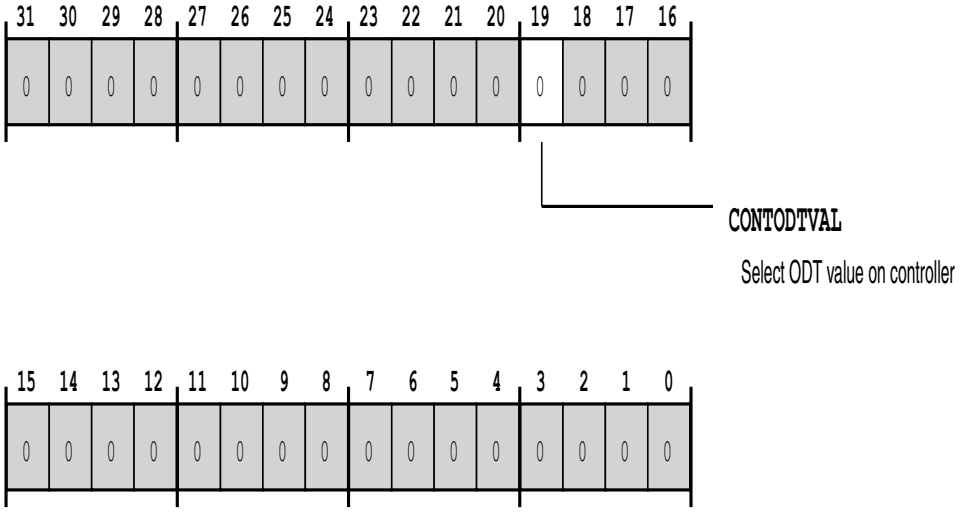


Figure 11-17: DMC_PHY_CTL1 Register Diagram

Table 11-18: DMC_PHY_CTL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	CONTODTVAL	Select ODT value on controller. The DMC_PHY_CTL1.CONTODTVAL bit selects the output drive termination (ODT) value.	
		0	75 Ohms Termination
		1	150 Ohms Termination

PHY Control 3 Register

The DMC_PHY_CTL3 register controls programmable PHY features.

DMC_PHY_CTL3: PHY Control 3 Register - R/W

Reset = 0x0000 0000

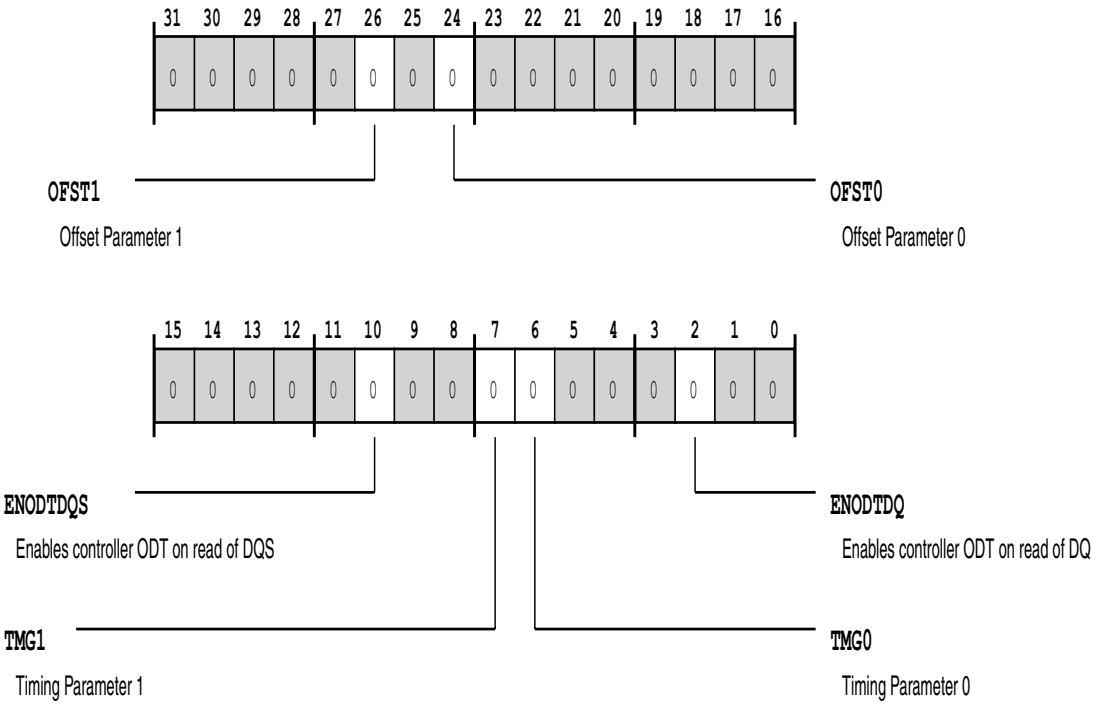


Figure 11-18: DMC_PHY_CTL3 Register Diagram

Table 11-19: DMC_PHY_CTL3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/W)	OFST1	Offset Parameter 1. Controls latching of data by the controller. . See HW Reference for proper setting. Needs to be set to 1 for BF609.	
24 (R/W)	OFST0	Offset Parameter 0. Controls latching of data by the controller. See section on product specific register and bit settings for more information. (For example, this bit needs to =1 for the ADSP-BF609 Blackfin processor.)	
10 (R/W)	ENODTDQS	Enables controller ODT on read of DQS. The DMC_PHY_CTL3.ENODTDQS bit enables ODT for DQS pads.	
		0	Disable
		1	Enable

Table 11-19: DMC_PHY_CTL3 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	TMG1	Timing Parameter 1. Controls latching of data by the controller. Needs to be set to 1 for BF609.	
6 (R/W)	TMG0	Timing Parameter 0. Controls latching of data by the controller. Needs to be set to 1 for BF609.	
2 (R/W)	ENODTDQ	Enables controller ODT on read of DQ. The DMC_PHY_CTL3.ENODTDQ bit enables ODT for DQ pads.	
		0	Disable
		1	Enable

PAD Control Register

This register allows programming control parameters associated with the DQ and DQS pads of the SDRAM memory interface. This register also allows programming control parameters associated with CK, CKE and CMD pads (RAS_b, CAS_b, WE_b, CS_b, ODT, CS_b, A and BA) of the SDRAM memory interface.

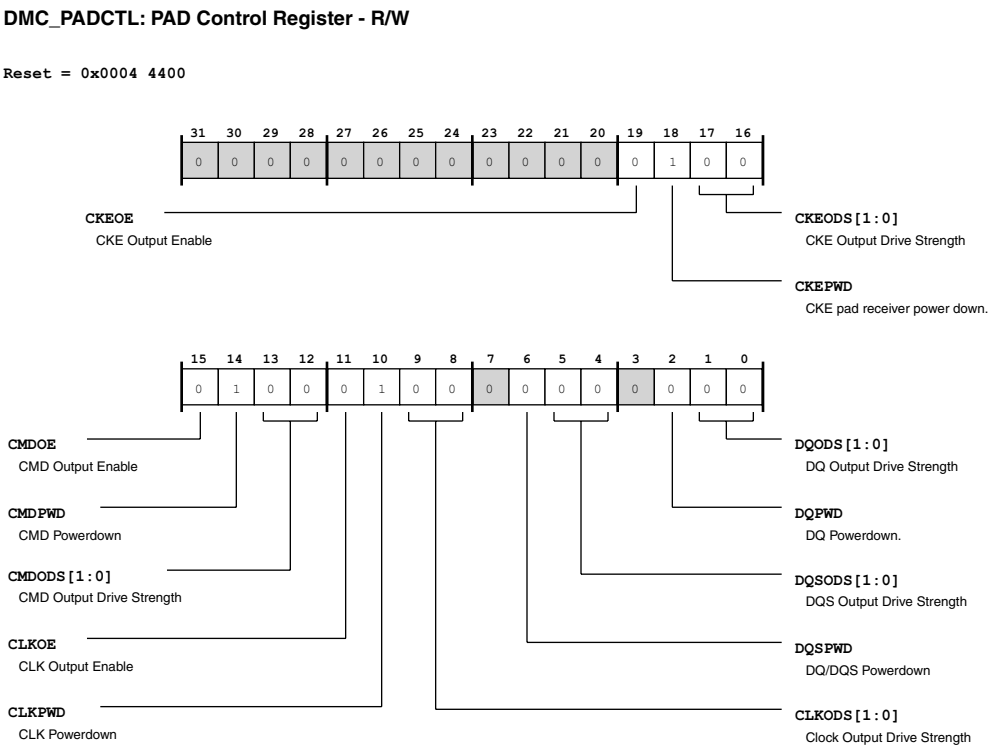


Figure 11-19: DMC_PADCTL Register Diagram

Table 11-20: DMC_PADCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	CKEOE	CKE Output Enable. The DMC_PADCTL.CKEOE bit selects the CKE pad output enable value.	
		0	Active OE value
		1	Inactive OE value
18 (R/W)	CKEPWD	CKE pad receiver power down..	
		0	Pad Receiver Powered Up
		1	Pad Receiver Powered Down
17:16 (R/W)	CKEODS	CKE Output Drive Strength. The DMC_PADCTL.CKEODS bits select the CKE pads output drive strength.	
		0	SSTL18 full drive / LPDDR 10mA
		1	SSTL 18 half drive / LPDDR 4mA
		2	LPDDR 8mA Reserved for DDR2 mode
		3	LPDDR 2mA Reserved for DDR2 mode
15 (R/W)	CMDOE	CMD Output Enable. The DMC_PADCTL.CMDOE bit selects the CMD pads output enable value.	
		0	Active OE Value
		1	Inactive OE Value
14 (R/W)	CMDPWD	CMD Powerdown. The DMC_PADCTL.CMDPWD bit selects whether the command, address, and control signal pads receiver is powered up or down.	
		0	Pad Receiver Powered up
		1	Pad Receive Powered down
13:12 (R/W)	CMDODS	CMD Output Drive Strength. The DMC_PADCTL.CMDODS bits select the command, address, and control signal drive strength.	
		0	SSTL18 full drive / LPDDR 10mA
		1	SSTL18 half drive / LPDDR 4mA
		2	LPDDR 8 mA
		3	LPDDR 2 mA

Table 11-20: DMC_PADCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	CLKOE	CLK Output Enable. The DMC_PADCTL.CLKOE bit selects the CLK pads output enable value.	
		0	Actvie OE Value
		1	Inactive OE Value
10 (R/W)	CLKPWD	CLK Powerdown. The DMC_PADCTL.CLKPWD bit selects whether the CLK pads receiver is powered up or down.	
		0	Pad Receiver Powered Up
		1	Pad Receiver Powered Down
9:8 (R/W)	CLKODS	Clock Output Drive Strength. The DMC_PADCTL.CLKODS bits select clock pad output drive strength.	
		0	SSTL18 full drive / LPDDR 10mA
		1	SSTL 18 half drive / LPDDR 4mA
		2	LPDDR 8 mA
		3	LPDDR 2 mA
6 (R/W)	DQSPWD	DQ/DQS Powerdown. The DMC_PADCTL.DQSPWD bit selects whether the DQ and DQS pads receiver is powered up or down.	
		0	Pad Receiver Powered Up
		1	Pad Receiver Powered Down
5:4 (R/W)	DQSODS	DQS Output Drive Strength. The DMC_PADCTL.DQSODS bits select the DQS pads output drive strength. Note that DMC_PADCTL.DQSODS[3] is connected to S1 of PAD, and DMC_PADCTL.DQSODS[4] is connected to S0 of PAD.	
		0	SSTL18 full drive / LPDDR 10mA
		1	SSTL 18 half drive / LPDDR 4mA
		2	LPDDR 8mA Reserved for DDR2 mode
		3	LPDDR 2mA
2 (R/W)	DQPWD	DQ Powerdown.. The DMC_PADCTL.DQPWD bit selects whether the DQ pads receiver is powered up or down.	
		0	Pad Receiver Powered Up
		1	Pad Receiver Powered Down

Table 11-20: DMC_PADCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	DQODS	DQ Output Drive Strength. The DMC_PADCTL.DQODS bits select the output drive strength for the DQ pads. Note that DMC_PADCTL.DQODS[0] is connected to A2 of PAD, and DMC_PADCTL.DQODS[1] is connected to A6 of PAD.	
		0	ODT Disable
		1	75 Ohm
		2	150 Ohm

ADSP-BF60x Specific Register/Bit Settings

1. Bits 6, 7, 24, and 26 of the DMC_PHY_CTL3 register all need to be set to 1.
2. DMC_DLLCTL register needs to be set to 0x54B. This keeps the default reset values and programs DATACYC to a value of 5.

12 Cyclic Redundancy Check (CRC)

The CRC peripheral is used to perform the Cyclic Redundancy Check (CRC) of the block of data that is presented to the peripheral. The peripheral provides a means to periodically verify the integrity of the system memory, the contents of memory-mapped registers (MMRs), or communication message objects, and 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 and if the two fail to match, the peripheral generates an error.

Data may be provided by the source channel of the memory-to-memory DMA channels and optionally forwarded to memory via the destination DMA channel. Alternatively, the peripheral also supports data presented by core write transactions.

The CRC peripheral implements a reduced table-lookup algorithm to compute the signature of the data. A programmable 32-bit CRC polynomial is used to automatically generate the lookup table (LUT) contents.

Additional 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.

CRC Features

The CRC peripheral supports a number of key features, including 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, and data mirroring options.

The CRC module 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 with CRC compute or compare
- Memory transfer with CRC compute or compare
- Memory fill with 32-bit data patterns
- Memory verify
- 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/monitor

CRC Functional Description

The CRC peripheral supports a number of modes of operation that allows 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 CRC signature to be calculated and verified for a memory region and also 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 may be written-to or read-from the peripheral using DMA transactions, core transactions, or a combination of both.

Two DMA channels are supported, providing both a data input and data output. CRC0 is connected to the MDMA0 channel pair and CRC1 is connected to the MDMA1 channel pair.

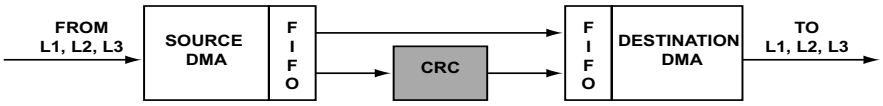


Figure 12-1: Memory Flow

The following sections describe in further detail the functional operation of the CRC peripheral:

- *ADSP-BF60x CRC Register List*
- *CRC Definitions*
- *CRC Block Diagram*
- *CRC Architectural Concepts*

ADSP-BF60x 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 the ASIL (Automobile Safety Integrity Level) requirements for the ADAS (Advanced Driver Assistance System) segment. This unit meets the requirements that the system software should be able to periodically check the correctness of the code/data available in the memory. A set of registers govern CRC operations. For more information on CRC functionality, see the CRC register descriptions.

Table 12-1: ADSP-BF60x CRC Register List

Name	Description
CRC_CTL	Control Register
CRC_DCNT	Data Word Count Register
CRC_DCNTRLD	Data Word Count Reload Register
CRC_COMP	Data Compare Register
CRC_FILLVAL	Fill Value Register
CRC_DFIFO	Data FIFO Register
CRC_INEN	Interrupt Enable Register
CRC_INEN_SET	Interrupt Enable Set Register
CRC_INEN_CLR	Interrupt Enable Clear Register
CRC_POLY	Polynomial Register
CRC_STAT	Status Register
CRC_DCNTCAP	Data Count Capture Register
CRC_RESULT_FIN	CRC Final Result Register
CRC_RESULT_CUR	CRC Current Result Register

ADSP-BF60x CRC Interrupt List

Table 12-2: ADSP-BF60x CRC Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
CRC0 Datacount expiration	88		LEVEL
CRC0 Error	89		LEVEL
CRC1 Datacount expiration	92		LEVEL
CRC1 Error	93		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 is capable of detecting 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 via a DMA channel allowing data to be distributed 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 following figure shows the functional block diagram of the CRC. The following sections describe the blocks.

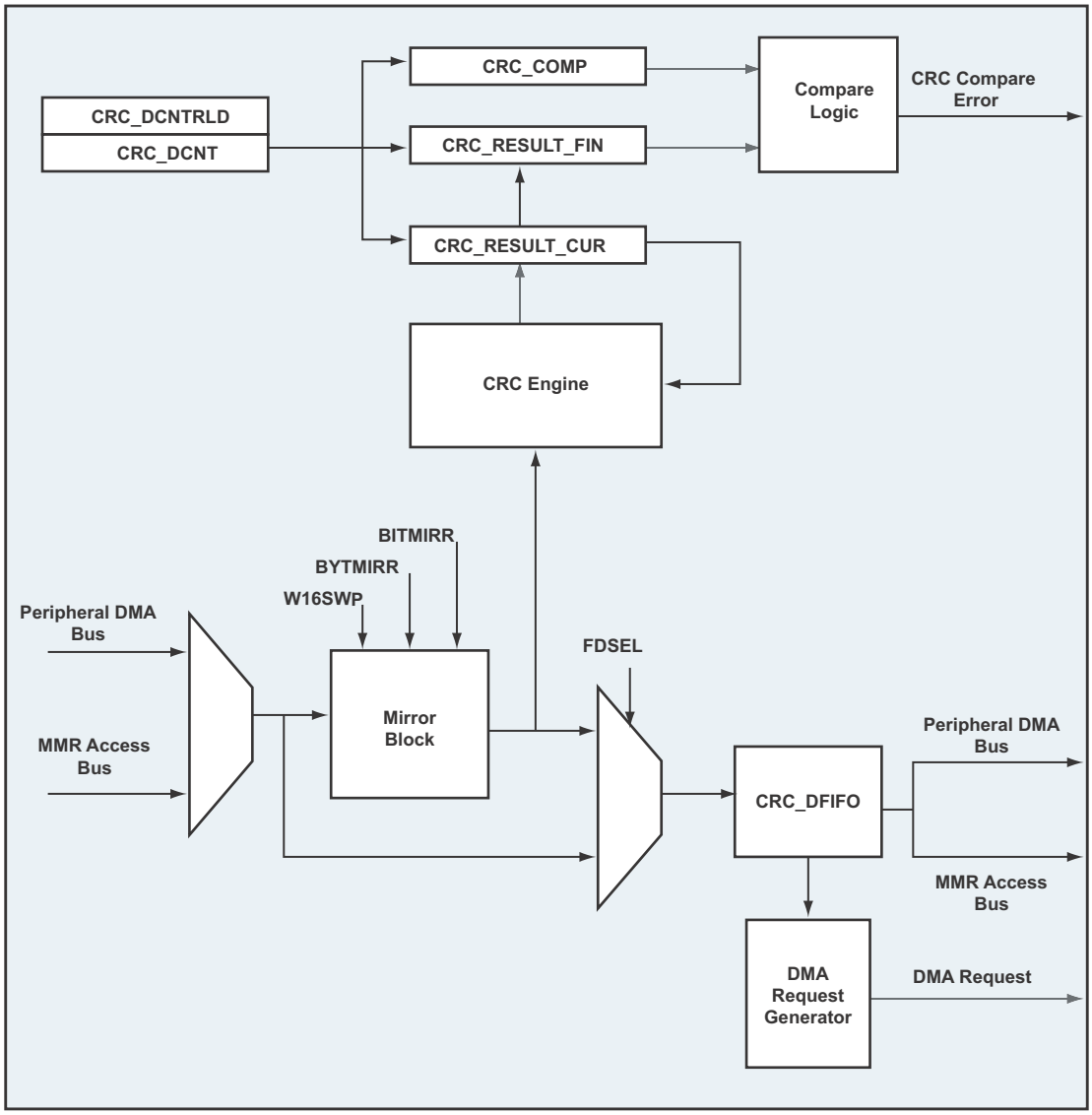


Figure 12-2: CRC Block Diagram

Peripheral DMA Bus

The CRC peripheral provides both an incoming and outgoing data path to the Peripheral DMA bus. The MDMA source channel is interfaced to the incoming data path providing data to the CRC peripheral. For memory transfer and data fill modes, the MDMA destination channel is used to either output the data from the CRC FIFO or the data to be used for the fill operation.

MMR Access Bus

The MMR access bus is used by the core to access all the memory-mapped registers of the peripheral for configuration, status and debug purposes. The core may also use the MMR access bus to feed data to the

CRC peripheral or read data from the FIFO of the CRC peripheral as an alternative to the operation being performed by the DMA channels.

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. Endian and reflection of processed data can be controlled by bit mirroring, byte mirroring, word swapping and any combination of these operations.

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 Lookup scheme. The CRC engine provides support for a user-programmable 32-bit polynomial that is used to load the lookup table parameters required for the CRC calculation. The CRC engine is a 2-cycle implementation operating on 16 bits of data per cycle.

Compare Logic

The compare logic takes the final CRC signature and compares this to the expected CRC signature, generating a CRC compare error if the signatures do not match. A compare error can flag a system fault.

CRC Architectural Concepts

The CRC peripheral includes a 32-bit CRC engine that implements the reduced table lookup scheme operating on 16 bits of data per cycle, resulting in a 2-cycle implementation for each 32 bits of data written to the peripheral. The upper 16 bits of the data are processed in the first cycle, followed by the lower 16 bits.

A 32-bit polynomial is required before calculation of the CRC signature can occur. The polynomial is used to generate the contents of an internal lookup table that is required by the reduced table lookup implemen-

tation. The lookup table that is automatically generated when the polynomial is written must be initialized prior to any operation that requires the use of the CRC engine.

The data presented to the CRC engine may be manipulated by the mirror block logic before being used 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 to be applied to the incoming data. For memory transfer compute and compare operations, programs may configure the peripheral to output the data in the same form in which it was received, or 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 is capable of receiving data, when data is available to be read from the FIFO and when the result of the CRC_RESULT_CUR register has been updated, indicating that the current CRC calculation has completed and the result is available.

Lookup Table

The lookup table consists of a set of 16 32-bit registers that are automatically populated by hardware when a write access takes place to the CRC_POLYregister. 16 clock cycles are required to generate all 16 look up table entries. The status of the lookup table generation process is reflected in CRC_STAT.LUTDONE allowing for software to poll on the completion of the event or for generation of an interrupt.

NOTE: The lookup table must be populated before any operation requiring the use of the CRC peripheral can take place, even if the operation does not require the use of the CRC engine. The peripheral will not issue any data requests until the table generation process has completed. 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 lookup table generation process.

Data Mirroring

The data mirror block may be configured to manipulate the incoming data before the data is passed on to the CRC engine and, optionally, to the FIFO. This 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 following table details how these options affect the incoming data and the output that is generated by the mirror block.

Table 12-3: 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]

Table 12-3: Data Mirroring Options (Continued)

W16SWP	BYTMIRR	BITMIRR	Output Data
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 may be written to the FIFO. This feature is controlled via the CRC_CTL.FDSELfield.

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 via the CRC_CTL.RSLTMIRR field. When mirroring is enabled, the values to be written to these registers are fully bit-reversed before being written.
- CRC_POLY, controlled via the CRC_CTL.POLYMIRR field. When mirroring is enabled, the 32-bit polynomial is fully bit-reversed before being written to the register.
- CRC_COMP, controlled via the CRC_CTL.CMPMIRR field. When mirroring is enabled, the contents to be loaded to this register are fully bit-reversed before being written.

FIFO Status and Data Requests

The CRC peripheral provides input and output buffer status indication via CRC_STAT.IBR and CRC_STAT.OBR respectively. For core-based operations, software is required to monitor these status fields prior to writing to or reading from the CRC FIFO. No write to the CRC FIFO should occur while CRC_STAT.IBR indicates that the buffer is not ready to accept data. Similarly, the CRC FIFO should not 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, whereas the memory transfer compute and compare mode is required to use both input and output buffer status. If at any point 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, as indicated by CRC_STAT.IRR.

NOTE: The memory transfer fill mode of operation requires the use of a DMA channel. Core reads from the CRC FIFO for this mode of operations are not supported.

Memory transfer compute and compare mode makes use of burst transactions in order 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, and the peripheral generates data requests to the source DMA channel (if DMA is used). As long as 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 until the CRC FIFO becomes full.

Once full, the CRC_STAT.IBR and CRC_STAT.OBR bits are updated accordingly, and then outgoing data requests are issued. 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 remaining to be processed is less than the number of words required to fill the FIFO, the burst mode of operation is disabled and incoming data is accepted as long as the FIFO is not full and 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 may be accepted as long as the FIFO is not full, and that outgoing data is available as long the FIFO is not empty.

The way in which data requests and the status bits are generated is additionally influenced by the CRC_CTL.OBRSTALL and CRC_CTL.IRRSTALL bit configurations described below.

- The CRC_CTL.OBRSTALL bit may be configured such that the CRC peripheral stalls as soon as there is output data available in the FIFO. This mode of operation should only be used in memory transfer compute and compare mode. This results in the processing of a single 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 may be configured so that the CRC peripheral stalls all further incoming data requests until the CRC_RESULT_CUR register is read after being updated. This mode of operation is only used for modes that result in CRC signature generation. It is not applicable to memory transfer data fill or memory scan data verify modes 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 if 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 if the data is found to be different.

Both of these modes of operation require, at the very most, a single DMA channel to read the data from memory into the peripheral. No data is forwarded to data output or destination DMA. Core-driven transfers may also be used 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. A CRC signature is generated on the data presented to the peripheral and compared with a predetermined and pre-loaded result. An error is generated if 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 either in the form of DMA channels, core driven write/read operations to/from the FIFO or 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

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 via the CRC_DCNT register. The accumulated result of the CRC operation is contained in the CRC_RESULT_CUR register. Upon each 32-bit word being processed by the CRC engine 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_COMP register is used to store the expected result of the CRC 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. CRC_STAT.CMPERR is required to be cleared before the next CRC operation is performed.

The CRC peripheral also contains CRC_DCNTRLD register. This register is used to reload CRC_DCNT upon completion of the CRC operation in preparation for the next transfer.

The initial seed of the CRC computation may be configured via CRC_CTL.AUTOCLRZ and CRC_CTL.AUTOCLRF. This 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 may 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 that are to be compared. The CRC_DCNT register is decremented upon receiving a new 32-bit word from the data stream. If at any point the compare operation should fail the CRC_STAT.CMPERR bit updated accordingly and the contents of CRC_DCNT are captured in the CRC_DCNTCAP register. This may be used in order to identify the location in the data stream where the error occurred. The CRC_STAT.CMPERR field should be cleared in order to re-enable 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 may be configured to allow for the compare error and data expiration events to generate an interrupt.

Memory Transfer Compute and Compare

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 via the CRC_DCNT register. The accumulated result of the CRC operation is contained in the CRC_RESULT_CUR register. Upon each 32-bit word being processed by the CRC engine 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_COMP register is used to store the expected result of the CRC 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. CRC_STAT.CMPERR is required to be cleared before the next CRC operation is performed.

The CRC peripheral also contains CRC_DCNTRLD register. This register is used to reload CRC_DCNTupon completion of the CRC operation in preparation for the next transfer.

The initial seed of the CRC computation may be configured via CRC_CTL.AUTOCLRZ and CRC_CTL.AUTOCLRF. This 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 may 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. This value is used to initialize a block memory via 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 that are to be written.

Once CRC_DCNTdecrements to zero, CRC_STAT.DCNTEXP is updated accordingly to signal the end of the operation. The peripheral may be configured to allow for the this 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 is capable of generating two interrupts that may optionally be enabled within the System Event Controller. One is a CRC status interrupt and the other a CRC error interrupt.

The CRC_STAT.CMPERR status bit may be configured as an interrupt and is signalled via the CRC error interrupt signal. The CRC_STAT.CMPERR status field is set whenever a compare operation performed by the CRC peripheral fails. This may be as 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 may be as a result of the CRC signature calculated for a memory region not matching the expected pre-programmed result for a memory scan or memory transfer compute compare operation.

The CRC_STAT.DCNTEXP status bit is set when the CRC_DCNT register has decremented to zero indicating that the CRC peripheral has now processed all the data that was requested for the current CRC operation. This signal may also be used to generate an interrupt. The interrupt is signalled on the CRC status interrupt signal.

Both these status bits may be configured to generate and interrupt via the CRC_INEN register. The CRC_INEN register also has bit set, CRC_INEN_SET, and bit clear CRC_INEN_CLR equivalent registers that may be used 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 via CRC_CTL.BLKEN does not result in the interrupt sources being cleared. The interrupt sources must be cleared via a W1C operation to CRC_STAT.

CRC Programming Model

It is important to note the following restrictions when using the CRC peripheral in conjunction with the DMA channels:

1. When enabling the CRC peripheral and the DMA channels, the CRC peripheral should be enabled prior to enabling the DMA channels.
2. When disabling the CRC peripheral and the DMA channels, the DMA channels should be disabled 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 Compare Mode*
- *DMA Driven Memory Scan Compute Compare Mode*
- *Core Driven Memory Scan Data Verify Mode*
- *DMA Driven Memory Scan Data Verify Mode*
- *Core Driven Memory Transfer Compute Compare Mode*
- *DMA Driven Memory Transfer Compute 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 `CRC_POLY`

   ADDITIONAL INFORMATION: This operation results in the CRC peripheral starting the LUT initialization process. `CRC_STAT.LUTDONE` is updated to reflect the operation is in progress.

2. Poll `CRC_STAT.LUTDONE` until the status bit indicates that the operation is completed.

RESULT:

The CRC peripheral has completed initialization of all the LUT registers and is now ready for data operations. The `CRC_STAT.LUTDONE` field remains in the current state until `CRC_POLY` is written again, or the peripheral or processor are reset.

Core Driven Memory Scan Compute 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 the burst mode of operation due to the stalling options configured via CRC_CTL being disabled.

PREREQUISITE:

The task assumes that the polynomial has been loaded and the look-up table is fully initialized, that all CRC interrupts have been serviced (none pending), and the CRC block is currently disabled as per CRC_CTL.BLKEN.

1. Initialize CRC_DCNT.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the CRC signature is to be calculated and verified.

2. Initialize CRC_DCNTRLD.

   ADDITIONAL INFORMATION: This is the value that is used to reload the CRC_DCNT register upon completion of current CRC operation. If no further operation is required then this register may be initialized to zero.

3. Initialize CRC_RESULT_CUR.

   ADDITIONAL INFORMATION: This register may be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize CRC_COMP.

   ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that is used in the final compare operation.

5. Initialize CRC_INEN.

   ADDITIONAL INFORMATION: This register is used to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts as required. If enabled ensure the corresponding interrupt handlers are also configured.

6. Initialize CRC_CTLwith CRC_CTL.OPMODE set to Memory Scan Compute Compare Mode and CRC_CTL.BLKEN configured to enable the CRC peripheral.

   - CRC_CTL.OBRSTALL and CRC_CTL.IRRSTALL options must be disabled for this task example.
   - All mirroring and bit reversal options should also be configured.
   - CRC auto clear options should also be configured.

   STEP RESULT: The CRC peripheral is now enabled and ready for data to be written by the core or DMA channel.

7. Write memory region data to the CRC peripheral.

   a. While CRC_STAT.IBR indicates input buffer is ready, write the CRC_DFIFO with 32-bit data.

   ADDITIONAL INFORMATION: This step is repeated until all required data has been written.

8. Poll CRC_STAT.DCNTEXP if the interrupt was disabled.

   ADDITIONAL INFORMATION: This step is required only if counter expired interrupt is disabled. Polling is required to ensure all the data has been processed.

9. Poll CRC_STAT.CMPERR if the interrupt was disabled to check for a compare error.

   ADDITIONAL INFORMATION: This step is only required if the compare error interrupt is not enabled.

10. Write CRC_STAT to clear both CRC_STAT.DCNTEXP and CRC_STAT.CMPERR.

    ADDITIONAL INFORMATION: If interrupts were enabled then the clearing of these status bits should be performed within the interrupt handlers for the respective interrupts.

    STEP RESULT: The CRC compute compare operation is now complete and the CRC peripheral is ready to be configured for the next CRC operation.

RESULT:

The integrity check of the memory via the expected CRC signature has completed and the final result is indicated via CRC_STAT.CMPERR and the corresponding interrupt if it was enabled.

AFTER COMPLETING THIS TASK:

Ensure any W1C CRC status bits are cleared before performing additional CRC operations.

DMA Driven Memory Scan Compute 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 via CRC_CTL being disabled.

PREREQUISITE:

The task assumes that the polynomial has been loaded and the look-up table is fully initialized, all CRC interrupts have been serviced (none pending), and the CRC block is currently disabled as per CRC_CTL.BLKEN.

1. Initialize CRC_DCNT.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the CRC signature is to be calculated and verified.

2. Initialize CRC_DCNTRLD.

   ADDITIONAL INFORMATION: This is the value that is used to reload the CRC_DCNT register upon completion of current CRC operation. If no further operation is required then this register may be initialized to zero.

3. Initialize CRC_RESULT_CUR.

   ADDITIONAL INFORMATION: This register may be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize CRC_COMP.

   ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that is used in the final compare operation.

5. Initialize CRC_INEN.

   ADDITIONAL INFORMATION: This register is used to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts as required. If enabled ensure the corresponding interrupt handlers are also configured.

6. Initialize CRC_CTL with CRC_CTL.OPMODE set to memory scan compute compare mode and CRC_CTL.BLKEN configured to enable the CRC peripheral.

   - CRC_CTL.OBRSTALL and CRC_CTL.IRRSTALL options must be disabled for this task example.
   - All mirroring and bit reversal options should also be configured.
   - CRC auto clear options should also be configured.

   STEP RESULT: The CRC peripheral is now enabled and ready for data to be written by the core or DMA channel.

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 CRC_STAT.DCNTEXP if the interrupt was disabled.

   ADDITIONAL INFORMATION: This step is required only if counter expired interrupt is disabled. Polling is required to ensure all the data has been processed.

9. Poll CRC_STAT.CMPERR if the interrupt was disabled to check for a compare error.

   ADDITIONAL INFORMATION: This step is only required if the compare error interrupt is not enabled.

10. Write CRC_STAT to clear both CRC_STAT.DCNTEXP and CRC_STAT.CMPERR.

    ADDITIONAL INFORMATION: If interrupts were enabled then the clearing of these status bits should be performed within the interrupt handlers for the respective interrupts.

    STEP RESULT: The CRC compute compare operation is now complete and the CRC peripheral is ready to be configured for the next CRC operation.

RESULT:

The integrity check of the memory via the expected CRC signature has completed and the final result indicated is via CRC_STAT.CMPERR and the corresponding interrupt if it were enabled.

AFTER COMPLETING THIS TASK:

Ensure any W1C CRC status bits are cleared before performing a further CRC operation. Any W1C status bits of the memory-to-memory source DMA channel should also be cleared 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.

PREREQUISITE:

The task assumes that the polynomial has been loaded and the look-up table is fully initialized, all CRC interrupts have been serviced (none pending), and the CRC block is currently disabled as per CRC_CTL.BLKEN. The interrupt service routine for the compare error interrupt should read and store the contents of CRC_DCNTCAP to a buffer before clearing the compare error interrupt.

1. Initialize CRC_DCNT.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the CRC signature is to be calculated and verified.

2. Initialize CRC_DCNTRLD.

   ADDITIONAL INFORMATION: This is the value that is used to reload the CRC_DCNT register upon completion of current CRC operation. If no further operation is required then this register may be initialized to zero.

3. Initialize CRC_COMP.

   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 will be compared with this value.

4. Initialize CRC_INEN.

   ADDITIONAL INFORMATION: This register is used to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts as required. If enabled ensure the corresponding interrupt handlers are also configured.

5. Initialize CRC_CTL with CRC_CTL.OPMODE set to memory scan data verify mode and CRC_CTL.BLKEN configured to enable the CRC peripheral.

   STEP RESULT: The CRC peripheral is now enabled and ready for data to be written by the core or DMA channel.

6. Write memory region data to the CRC peripheral.

   a. Poll CRC_STAT.IBR until input buffer is ready.

   b. Write CRC_DFIFO with 32-bit data.

   ADDITIONAL INFORMATION: These two steps are repeated until the entire memory region has been written to the CRC peripheral.

7. Poll CRC_INEN_SET.DCNTEXP if the interrupt was disabled.

   ADDITIONAL INFORMATION: This step is required only if counter expired interrupt is disabled. Polling is required to ensure 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 CRC_STAT to clear both CRC_INEN_SET.DCNTEXP and CRC_INEN.CMPERR.

   ADDITIONAL INFORMATION: If interrupts were enabled then the clearing of these status bits should be performed within the interrupt handlers for the respective interrupts.

   STEP RESULT: The CRC memory scan verify operation is now complete and the CRC peripheral is ready to be configured for the next CRC operation.

RESULT:

The result of the integrity check of the memory with the 32-bit constant is indicated via CRC_INEN.CMPERR and the corresponding interrupt if it were enabled. Each comparison error is traceable due to the logging of CRC_DCNTCAP from within the compare error interrupt handler.

AFTER COMPLETING THIS TASK:

Ensure any W1C CRC status bits are cleared before performing a further CRC operation.

DMA Driven 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.

PREREQUISITE:

The task assumes that the polynomial has been loaded and the look-up table is fully initialized, all CRC interrupts have been serviced (none pending), and the CRC block is currently disabled as per `CRC_CTL.BLKEN`. The interrupt service routine for the compare error interrupt should read and store the contents of `CRC_DCNTCAP` to a buffer before clearing the compare error interrupt.

1. Initialize `CRC_DCNT`.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the CRC signature is to be calculated and verified.

2. Initialize `CRC_DCNTRLD`.

   ADDITIONAL INFORMATION: This is the value that is used to reload the `CRC_DCNT` register upon completion of current CRC operation. If no further operation is required then this register may be initialized to zero.

3. Initialize `CRC_COMP`.

   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 will be compared with this value.

4. Initialize `CRC_INEN`.

   ADDITIONAL INFORMATION: This register is used to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts as required. If enabled ensure the corresponding interrupt handlers are also configured.

5. Initialize `CRC_CTL` with `CRC_CTL.OPMODE` set to memory scan data verify mode and `CRC_CTL.BLKEN` configured to enable the CRC peripheral.

   STEP RESULT: The CRC peripheral is now enabled and ready for data to be written by the core or DMA channel.

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 `CRC_STAT.DCNTEXP` if the interrupt was disabled.

   ADDITIONAL INFORMATION: This step is required only if counter expired interrupt is disabled. Polling is required to ensure 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 CRC_STAT to clear both CRC_STAT.DCNTEXP and CRC_STAT.CMPERR.

   ADDITIONAL INFORMATION: If interrupts were enabled then the clearing of these status bits should be performed within the interrupt handlers for the respective interrupts.

   STEP RESULT: The CRC memory scan verify operation is now complete and the CRC peripheral is ready to be configured for the next CRC operation.

RESULT:

The result of the integrity check of the memory with the 32-bit constant is indicated via CRC_STAT.CMPERR and the corresponding interrupt if it were enabled. Each comparison error is traceable due to the logging of CRC_DCNTCAP from within the compare error interrupt handler.

AFTER COMPLETING THIS TASK:

Ensure any W1C CRC status bits and DMA status bits are cleared before performing a further CRC operation.

Core Driven Memory Transfer Compute 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 via CRC_CTL being disabled.

PREREQUISITE:

The task assumes that the polynomial has been loaded and the look-up table is fully initialized, all CRC interrupts have been serviced (none pending), and the CRC block is currently disabled as per CRC_CTL.BLKEN.

1. Initialize CRC_DCNT.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the CRC signature is to be calculated and verified.

2. Initialize CRC_DCNTRLD.

   ADDITIONAL INFORMATION: This is the value that is used to reload the CRC_DCNT register upon completion of current CRC operation. If no further operation is required then this register may be initialized to zero.

3. Initialize CRC_RESULT_CUR.

   ADDITIONAL INFORMATION: This register may be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize CRC_COMP.

   ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that is used in the final compare operation.

5. Initialize CRC_INEN.

   ADDITIONAL INFORMATION: This register is used to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts as required. If enabled ensure the corresponding interrupt handlers are also configured.

6. Initialize CRC_CTL with CRC_CTL.OPMODE set to memory scan compute compare mode and CRC_CTL.BLKEN configured to enable the CRC peripheral.

   - CRC_CTL.OBRSTALL and CRC_CTL.IRRSTALL options must be disabled for this task example.
   - All mirroring and bit reversal options should also be configured.
   - CRC auto clear options should also be configured.

   STEP RESULT: The CRC peripheral is now enabled and ready for data to be written by the core or DMA channel.

7. Write memory region data to the CRC peripheral and read it back to the new destination.

   a. While CRC_STAT.IBR indicates input buffer is ready, write CRC_DFIFO with 32-bit data.

   b. While CRC_STAT.OBR indicates output buffer is ready, read CRC_DFIFO and store data to new destination.

   ADDITIONAL INFORMATION: These two steps are repeated until all required data has been processed through the CRC peripheral and copied to the new destination.

8. Poll CRC_STAT.DCNTEXP if the interrupt was disabled.

   ADDITIONAL INFORMATION: This step is required only if the counter expired interrupt is disabled. Polling is required to ensure all the data has been processed.

9. Poll CRC_STAT.CMPERR if the interrupt was disabled to check for a compare error.

   ADDITIONAL INFORMATION: This step is only required if the compare error interrupt is not enabled.

10. Write CRC_STAT to clear both CRC_STAT.DCNTEXP and CRC_STAT.CMPERR.

    ADDITIONAL INFORMATION: If interrupts were enabled then the clearing of these status bits should be performed within the interrupt handlers for the respective interrupts.

    STEP RESULT: The CRC compute and compare operation is now complete and the CRC peripheral is ready to be configured for the next CRC operation. The memory region has also been copied to its new destination.

RESULT:

The memory region has been copied to a new location and an integrity check of the memory via the expected CRC signature has also completed and the final result is indicated via CRC_STAT.CMPERR and the corresponding interrupt if it were enabled.

AFTER COMPLETING THIS TASK:

Ensure any W1C CRC status bits are cleared before performing a further CRC operation.

DMA Driven Memory Transfer Compute 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 via the use of Memory-to-Memory DMA transfers. The CRC peripheral is configured such that it operates in the burst mode of operation due to the stalling options configured via CRC_CTL being disabled.

PREREQUISITE:

The task assumes that the polynomial has been loaded and the look-up table is fully initialized, all CRC interrupts have been serviced (none pending), and the CRC block is currently disabled as per CRC_CTL.BLKEN.

1. Initialize CRC_DCNT.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the CRC signature is to be calculated and verified.

2. Initialize CRC_DCNTRLD.

   ADDITIONAL INFORMATION: This is the value that is used to reload the CRC_DCNT register upon completion of current CRC operation. If no further operation is required then this register may be initialized to zero.

3. Initialize CRC_RESULT_CUR.

   ADDITIONAL INFORMATION: This register may be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize CRC_COMP.

   ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that is used in the final compare operation.

5. Initialize CRC_INEN.

   ADDITIONAL INFORMATION: This register is used to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts as required. If enabled ensure the corresponding interrupt handlers are also configured.

6. Initialize CRC_CTL with CRC_CTL.OPMODE set to memory scan compute compare mode and CRC_CTL.BLKEN configured to enable the CRC peripheral.

   - CRC_CTL.OBRSTALL and CRC_CTL.IRRSTALL options must be disabled for this task example.
   - All mirroring and bit reversal options should also be configured.
   - CRC auto clear options should also be configured.

   STEP RESULT: The CRC peripheral is now enabled and ready for data to be written by the core or DMA channel.

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 via the memory-to-memory DMA channels and the CRC peripheral.

8. Poll CRC_STAT.DCNTEXP if the interrupt was disabled.

   ADDITIONAL INFORMATION: This step is required only if counter expired interrupt is disabled. Polling is required to ensure all the data has been processed.

9. Poll CRC_STAT.CMPERR if the interrupt was disabled to check for a compare error.

   ADDITIONAL INFORMATION: This step is only required if the compare error interrupt is not enabled.

10. Write CRC_STAT to clear both CRC_STAT.DCNTEXP and CRC_STAT.CMPERR.

    ADDITIONAL INFORMATION: If interrupts were enabled then the clearing of these status bits should be performed within the interrupt handlers for the respective interrupts.

    STEP RESULT: The CRC compute and compare operation is now complete and the CRC peripheral is ready to be configured for the next CRC operation. The memory region has also been copied to its new destination.

RESULT:

The integrity check of the memory via the expected CRC signature has completed and the final result is indicated via CRC_STAT.CMPERR and the corresponding interrupt if it were enabled. The memory region has also been copied to its final destination.

AFTER COMPLETING THIS TASK:

Ensure any W1C CRC status bits are cleared before performing a further CRC operation. Any W1C status bits of the memory-to-memory source and destination DMA channels should also be cleared before the next CRC operation.

DMA Driven Memory Transfer Data Fill Mode

Initializes a region of memory to a constant 32-bit value using DMA transactions.

PREREQUISITE:

The task assumes that the polynomial has been loaded and the look-up table is fully initialized, all CRC interrupts have been serviced (none pending), and the CRC block is currently disabled as per CRC_CTL.BLKEN.

1. Initialize CRC_DCNT.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the CRC signature is to be calculated and verified.

2. Initialize CRC_DCNTRLD.

   ADDITIONAL INFORMATION: This is the value that is used to reload the CRC_DCNT register upon completion of current CRC operation. If no further operation is required then this register may be initialized to zero.

3. Initialize CRC_FILLVAL.

   ADDITIONAL INFORMATION: This register contains the 32-bit constant that is used to fill the memory region.

4. Initialize CRC_INEN.

   ADDITIONAL INFORMATION: This register is used to enable the generation of the CRC interrupts for notification of block completion. Configure these interrupts as required. If enabled ensure the corresponding interrupt handlers are also configured.

5. Initialize CRC_CTL with CRC_CTL.OPMODE set to memory transfer fill mode and CRC_CTL.BLKEN configured to enable the CRC peripheral.

   STEP RESULT: The CRC peripheral is now enabled and is ready for data to be written by the DMA channel

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 `CRC_STAT.DCNTEXP` if the interrupt was disabled.

   ADDITIONAL INFORMATION: This step is required only if counter expired interrupt is disabled. Polling is required to ensure all the data has been processed.

8. Write `CRC_STAT` to clear `CRC_STAT.DCNTEXP`.

   ADDITIONAL INFORMATION: If interrupts were enabled then the clearing of this status bit should be performed within the interrupt handlers for the respective interrupts.

   STEP RESULT: The CRC memory transfer fill operation is now complete and the CRC peripheral is ready to be configured for the next CRC operation.

RESULT:

The memory region is now filled with the constant data and the CRC peripheral is ready to be configured for a new operation.

AFTER COMPLETING THIS TASK:

Ensure any W1C CRC status bits and DMA status bits are cleared before performing a further CRC operation.

CRC Peripheral and DMA Channel List

Table 12-4: CRC DMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/Monitor Support
DMA21	CRC0 Receive	128	Yes
DMA22	CRC0 Transmit	64	Yes
DMA23	CRC1 Receive	64	Yes
DMA24	CRC1 Transmit	64	Yes

Table 12-5: CRC DMA Channels (Continued)

DMA Channel	Memory Bus Width	Peripheral Bus Width	Max Outstanding Reads	Max Outstanding Writes
DMA21	32-bit	32-bit	8	7
DMA22	32-bit	32-bit	8	4

Table 12-5: CRC DMA Channels (Continued) (Continued)

DMA Channel	Memory Bus Width	Peripheral Bus Width	Max Outstanding Reads	Max Outstanding Writes
DMA23	32-bit	32-bit	8	4
DMA24	32-bit	32-bit	8	4

ADSP-BF60x CRC Register Descriptions

Cyclic Redundancy Check Unit (CRC) contains the following registers.

Table 12-6: ADSP-BF60x CRC Register List

Name	Description
CRC_CTL	Control Register
CRC_DCNT	Data Word Count Register
CRC_DCNTRLD	Data Word Count Reload Register
CRC_COMP	Data Compare Register
CRC_FILLVAL	Fill Value Register
CRC_DFIFO	Data FIFO Register
CRC_INEN	Interrupt Enable Register
CRC_INEN_SET	Interrupt Enable Set Register
CRC_INEN_CLR	Interrupt Enable Clear Register
CRC_POLY	Polynomial Register
CRC_STAT	Status Register
CRC_DCNTCAP	Data Count Capture Register
CRC_RESULT_FIN	CRC Final Result Register
CRC_RESULT_CUR	CRC Current Result Register

Control Register

The CRC_CTL configures the operation modes and settings for the CRC.

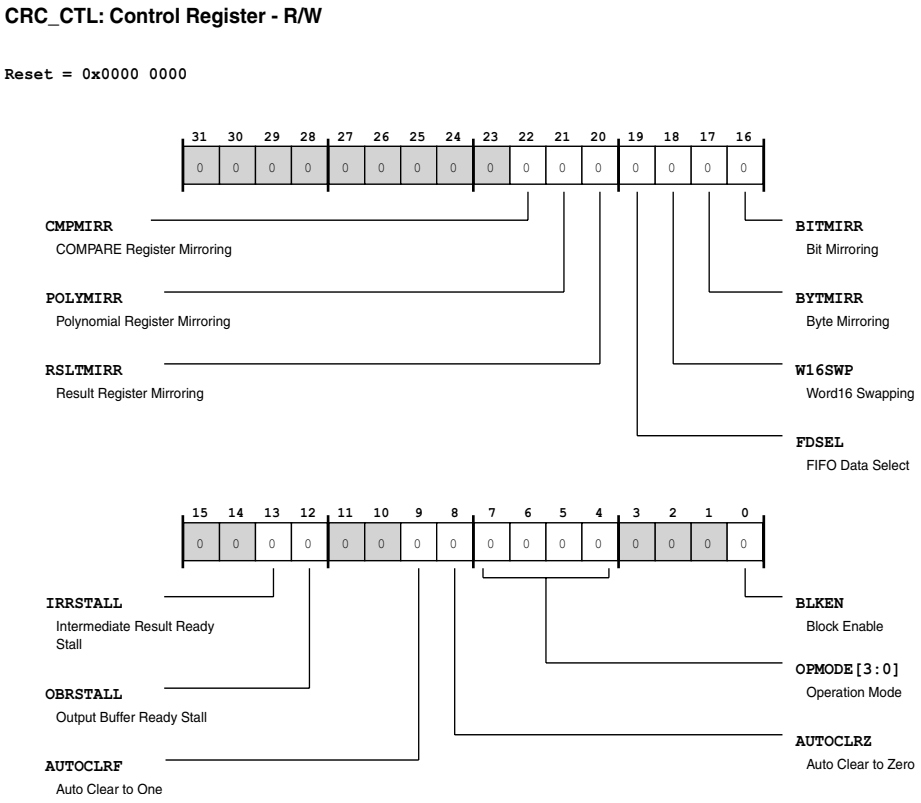


Figure 12-3: CRC_CTL Register Diagram

Table 12-7: CRC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/W)	CMPMIRR	COMPARE Register Mirroring. The CRC_CTL.CMPMIRR enables data mirroring for the CRC_COMP 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 CRC_RESULT_FIN register.	
		0	Disable compare mirroring
		1	Enable compare mirroring

Table 12-7: CRC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	POLYMIRR	Polynomial Register Mirroring. The CRC_CTL.POLYMIRR enables data mirroring for the CRC_POLY 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
20 (R/W)	RSLTMIRR	Result Register Mirroring. The CRC_CTL.RSLTMIRR enables data mirroring for the CRC_RESULT_CUR and CRC_RESULT_FIN result registers. When enabled, the 32-bit values is these registers are fully bit mirrored (reversed).	
		0	Disable result mirroring
		1	Enable result mirroring
19 (R/W)	FDSEL	FIFO Data Select. The CRC_CTL.FDSEL 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 (CRC_CTL.BITMIRR, CRC_CTL.BYTMIRR, and CRC_CTL.W16SWP) 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 CRC_CTL.W16SWP 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 CRC_CTL.BYTMIRR 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

Table 12-7: CRC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	BITMIRR	Bit Mirroring. The CRC_CTL.BITMIRR 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 CRC_CTL.IRRSTALL enables stalling the state machine for input data when there is a valid intermediate result to be read in CRC_RESULT_CUR. This feature should be used only in CRC computation modes (for example, CRC_CTL.OPMODE =1 or =3).	
		0	Do not stall
		1	Stall on IRR
12 (R/W)	OBRSTALL	Output Buffer Ready Stall. The CRC_CTL.OBRSTALL enables stalling the state machine for input data when there is a 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 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 CRC_CTL.AUTOCLRZ must be disabled, or the CRC_CTL.AUTOCLRF has no effect.	
		0	No auto clear
		1	Auto clear
8 (R/W)	AUTOCLRZ	Auto Clear to Zero. The CRC_CTL.AUTOCLRZ 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

Table 12-7: CRC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7:4 (R/W)	OPMODE	Operation Mode. The CRC_CTL.OPMODE 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 enables/disables CRC operation.	
		0	Disable
		1	Enable

Data Word Count Register

The CRC_DCNT 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 CRC_DCNT is automatically loaded from the CRC_DCNTRLD for the next CRC operation. Note that the initial value programmed into CRC_DCNT may be different from what is programmed in the CRC_DCNTRLD.

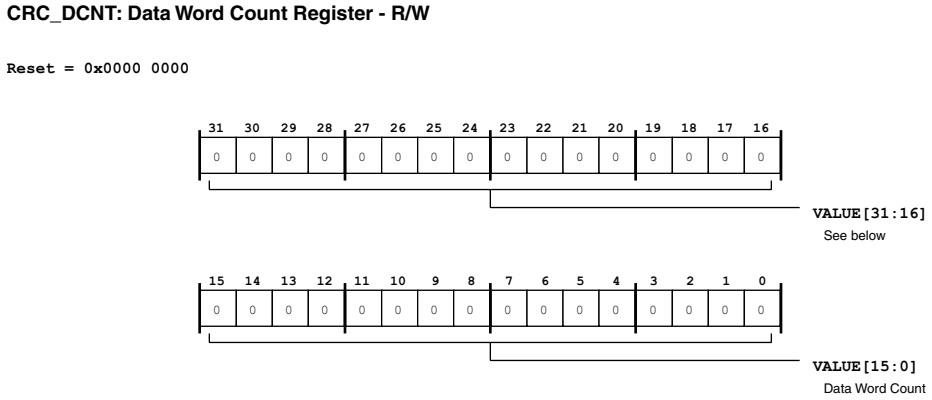


Figure 12-4: CRC_DCNT Register Diagram

Table 12-8: CRC_DCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Data Word Count.

Data Word Count Reload Register

The CRC_DCNTRLD 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 CRC_DCNTRLD 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.

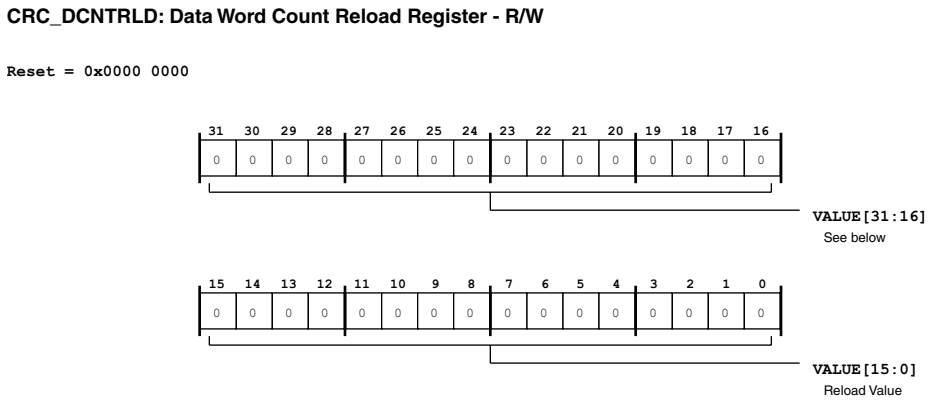


Figure 12-5: CRC_DCNTRLD Register Diagram

Table 12-9: CRC_DCNTRLD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Reload Value.

Data Compare Register

The CRC_COMP 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.

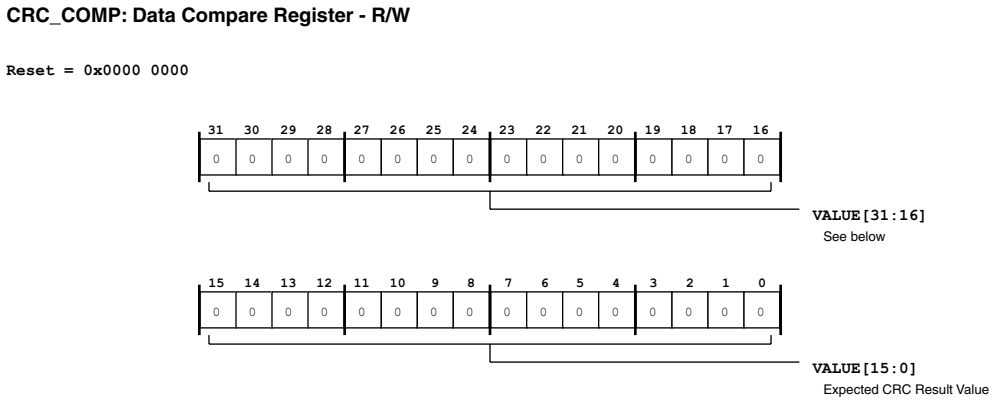


Figure 12-6: CRC_COMP Register Diagram

Table 12-10: CRC_COMP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Expected CRC Result Value.

Fill Value Register

The CRC_FILLVAL 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.

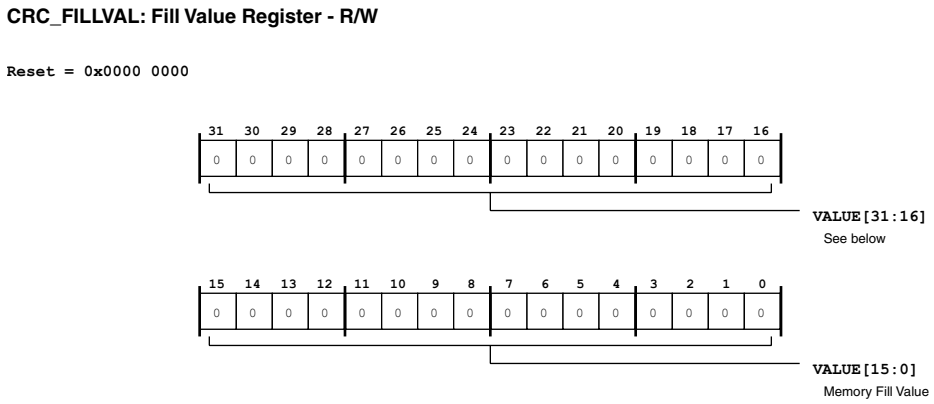


Figure 12-7: CRC_FILLVAL Register Diagram

Table 12-11: CRC_FILLVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Memory Fill Value.

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 actually does not get written into this FIFO. So, this register should not be read in these modes.

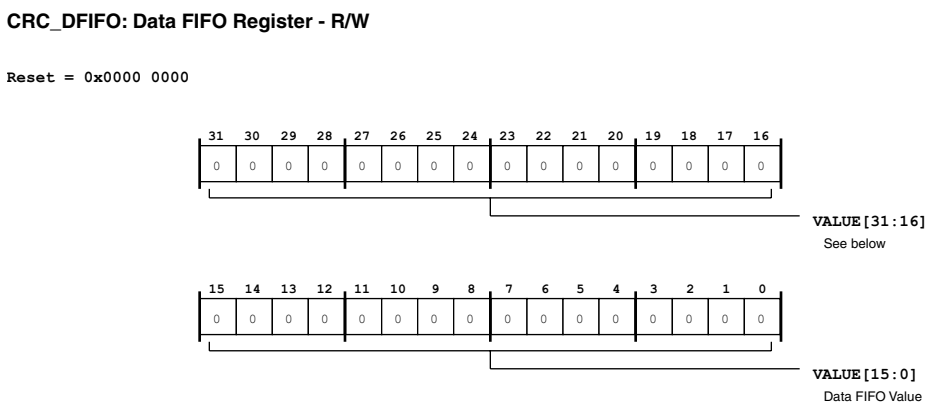


Figure 12-8: CRC_DFIFO Register Diagram

Table 12-12: CRC_DFIFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Data FIFO Value.

Interrupt Enable Register

The CRC_INEN unmasks (enables) or masks (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).

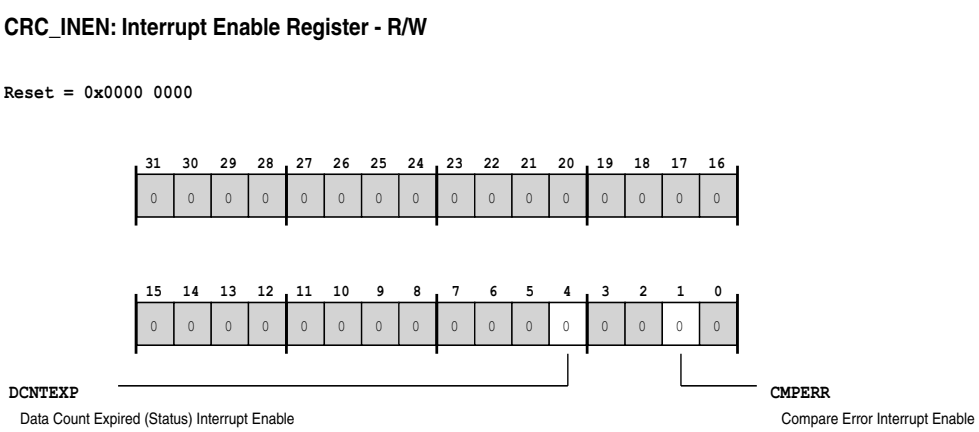


Figure 12-9: CRC_INEN Register Diagram

Table 12-13: CRC_INEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	DCNTEXP	Data Count Expired (Status) Interrupt Enable. The CRC_INEN.DCNTEXP 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 CRC_INEN.CMPERR enables (unmasks) the data compare interrupt, which is generated when CRC data comparison fails.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt

Interrupt Enable Set Register

The CRC_INEN_SET permits setting individual bits in the CRC_INEN register without affecting other bits in the register.

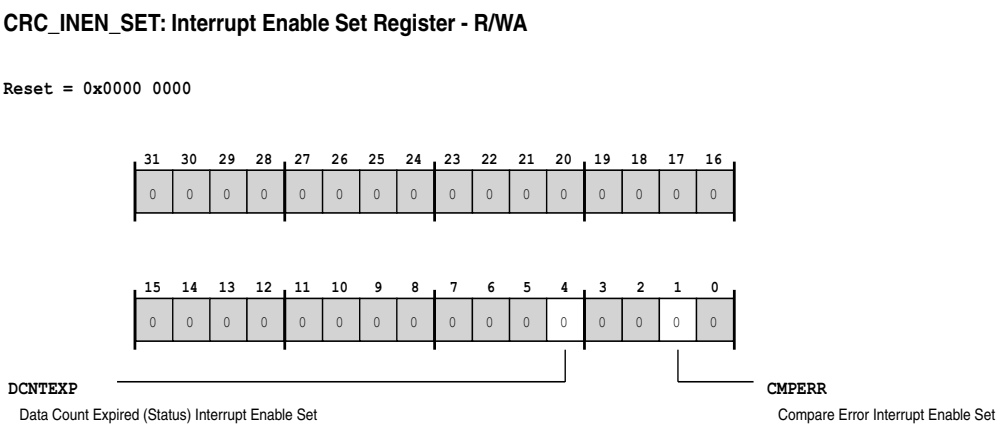


Figure 12-10: CRC_INEN_SET Register Diagram

Table 12-14: CRC_INEN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R0/WS)	DCNTEXP	Data Count Expired (Status) Interrupt Enable Set.	
		0	No Effect
		1	Set Bit
1 (R0/WS)	CMPERR	Compare Error Interrupt Enable Set.	
		0	No Effect
		1	Set Bit

Interrupt Enable Clear Register

The CRC_INEN_CLR permits clearing individual bits in the CRC_INEN register without affecting other bits in the register.

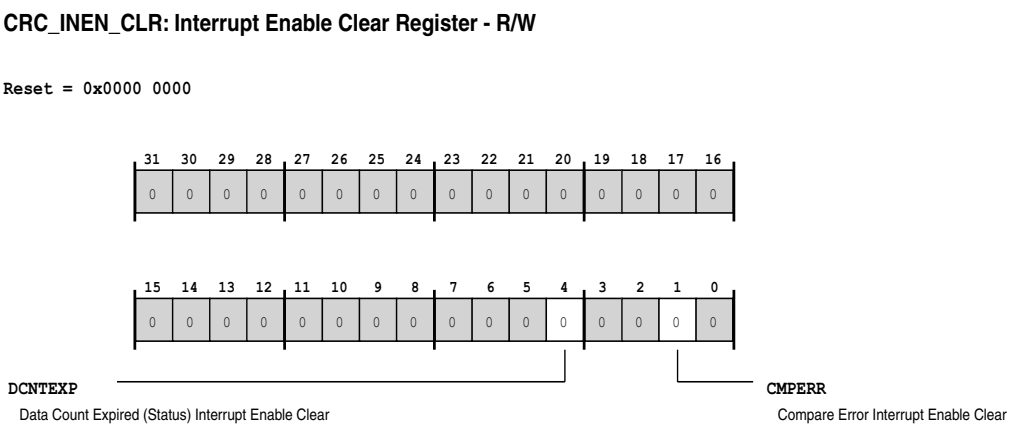


Figure 12-11: CRC_INEN_CLR Register Diagram

Table 12-15: CRC_INEN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R0/WC)	DCNTEXP	Data Count Expired (Status) Interrupt Enable Clear.	
		0	No Effect
		1	Clear Bit
1 (R0/WC)	CMPERR	Compare Error Interrupt Enable Clear.	
		0	No Effect
		1	Clear Bit

Polynomial Register

The CRC_POLY holds a 32-bit polynomial for CRC operations. Bit 31 corresponds to coefficient of x31 of the CRC polynomial, bit 30 corresponds to coefficient of x30, and so on through to bit 0. Coefficient of x32 is assumed to be "1" for any polynomial that is selected. Based on the polynomial in CRC_POLY, the CRC generates a look-up table (LUT), which is used to compute the CRC of the incoming data stream.

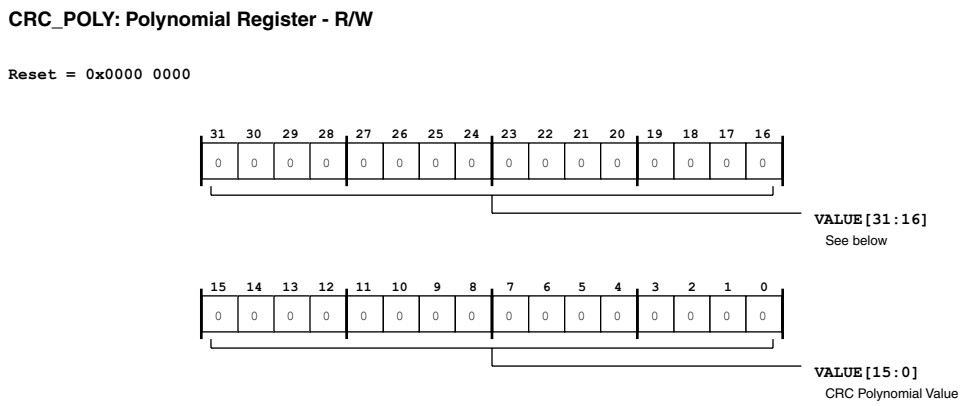


Figure 12-12: CRC_POLY Register Diagram

Table 12-16: CRC_POLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	CRC Polynomial Value.

Status Register

The CRC_STAT indicates status for CRC operations and interrupt generation.

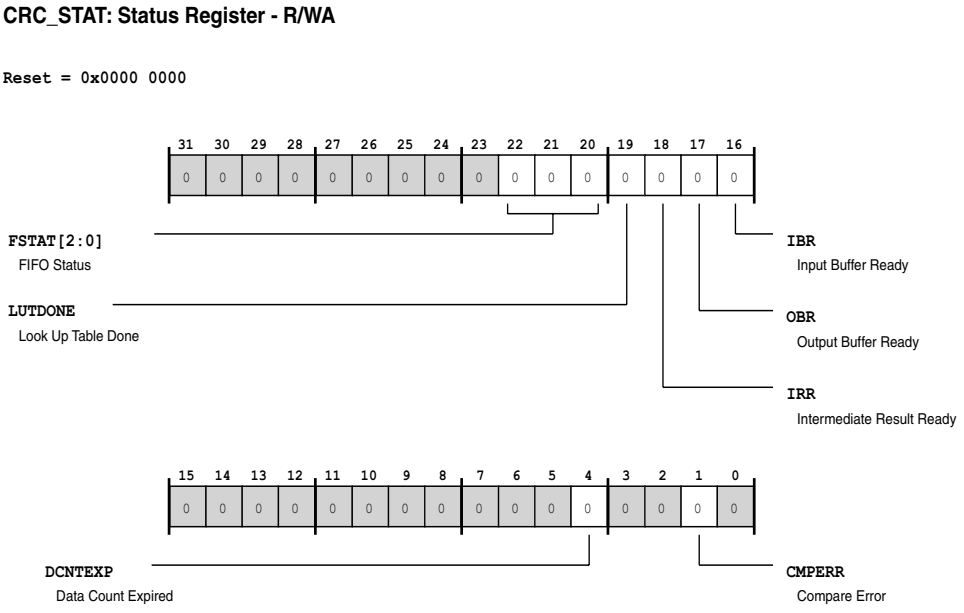


Figure 12-13: CRC_STAT Register Diagram

Table 12-17: CRC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
22:20 (R/NW)	FSTAT	FIFO Status. The CRC_STAT.FSTAT 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
		4	FIFO has 4 data (Full)
19 (R/NW)	LUTDONE	Look Up Table Done. The CRC_STAT.LUTDONE 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 CRC_POLY is written.	
		0	No Status
		1	LUT Generation Done
18 (R/NW)	IRR	Intermediate Result Ready. The CRC_STAT.IRR 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 CRC_CTL.IRRSTALL is enabled.	
		0	No Status
		1	Intermediate Results Ready
17 (R/NW)	OBR	Output Buffer Ready. The CRC_STAT.OBR 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

Table 12-17: CRC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/NW)	IBR	Input Buffer Ready. The CRC_STAT.IBR 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 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 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 is disabled from capturing the data count values.	
		0	No Status
		1	Compare Error

Data Count Capture Register

The CRC_DCNTCAP captures the CRC_DCNT value when a compare operation fails in data verify mode. This capture can be used to track the position of error in the data stream. Capture operation is enabled only if the CRC_STAT.CMPERR indicates no compare error. After an error occurs and data count is captured, no further errors are logged until the CRC_STAT.CMPERR bit is cleared. To obtain the position of error in the data stream, subtract the CRC_DCNTCAP value from the initial CRC_DCNT.

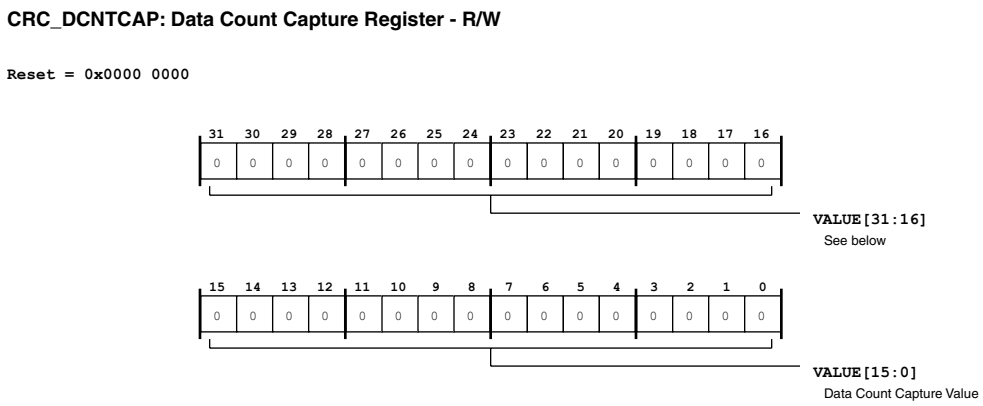


Figure 12-14: CRC_DCNTCAP Register Diagram

Table 12-18: CRC_DCNTCAP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Data Count Capture Value.

CRC Final Result Register

The CRC_RESULT_FIN 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 datastream, the CRC loads CRC_RESULT_FIN with the value from CRC_RESULT_CUR.

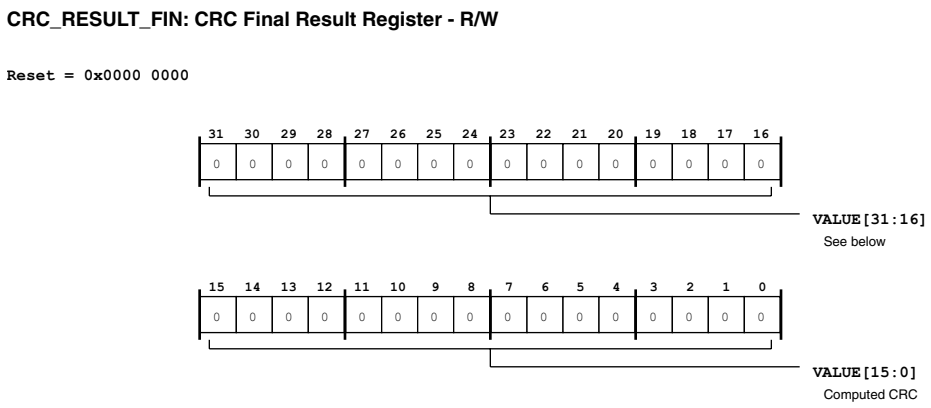


Figure 12-15: CRC_RESULT_FIN Register Diagram

Table 12-19: CRC_RESULT_FIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Computed CRC.

CRC Current Result Register

The CRC_RESULT_CUR holds the current or intermediate CRC result and 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. The CRC_RESULT_CUR 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 should be programmed to the desired value. Note that this register can be read by the processor core at any time.

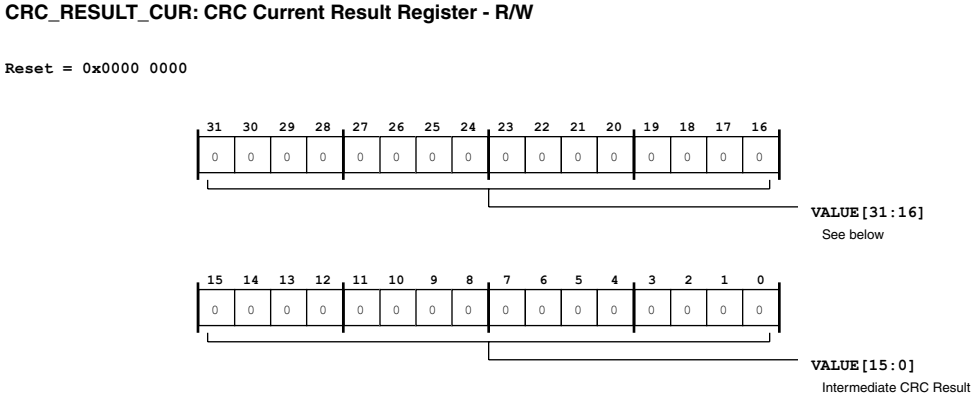


Figure 12-16: CRC_RESULT_CUR Register Diagram

Table 12-20: CRC_RESULT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Intermediate CRC Result.

13 Direct Memory Access (DMA)

The DMA channels are dispersed throughout the infrastructure and may be clustered together via system crossbars (SCB) so as to share 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. Two DMA channels are required for memory to memory DMA transfers (MDMA). 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 can be descriptor-based or register-based. Register-based DMA allows the processor to directly program DMA controller registers to initiate a DMA transfer. On completion, the controller registers may 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. Descriptor-based transfers allow the chaining together of multiple DMA sequences. In Descriptor-based DMA operations, a DMA channel can be programmed to 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 is passed through an external memory interface port. Any kind of device that is supported by the external memory interface can also be accessed by DMA operations. This is typically flash memory, SRAM, DDR SDRAM, FIFOs, or memory-mapped peripheral devices.

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
  - 1-D and 2-D transfers supported per descriptor set
  - 3-D support provided by chained descriptor sets
- Configurable memory and peripheral transfer word sizes
  - Memory interface supports 8, 16, 32, 64, 128 and 256-bit transfers
  - Peripheral interface supports for 8, 16, 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

DMA Channel Functional Description

This section provides a functional description of the DMA channel interface.

ADSP-BF60x DMA Register List

The DMA channel (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 DMAs. A set of registers govern DMA operations. For more information on DMA functionality, see the DMA register descriptions.

Table 13-1: ADSP-BF60x DMA Register List

Name	Description
DMA_DSCPTR_NXT	Pointer to Next Initial Descriptor
DMA_ADDRSTART	Start Address of Current Buffer
DMA_CFG	Configuration Register
DMA_XCNT	Inner Loop Count Start Value
DMA_XMOD	Inner Loop Address Increment
DMA_YCNT	Outer Loop Count Start Value (2D only)
DMA_YMOD	Outer Loop Address Increment (2D only)
DMA_DSCPTR_CUR	Current Descriptor Pointer
DMA_DSCPTR_PRV	Previous Initial Descriptor Pointer
DMA_ADDR_CUR	Current Address
DMA_STAT	Status Register
DMA_XCNT_CUR	Current Count(1D) or intra-row XCNT (2D)
DMA_YCNT_CUR	Current Row Count (2D only)
DMA_BWLCNT	Bandwidth Limit Count
DMA_BWLCNT_CUR	Bandwidth Limit Count Current
DMA_BWMCNT	Bandwidth Monitor Count
DMA_BWMCNT_CUR	Bandwidth Monitor Count Current

DMA Definitions

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

Descriptor

An individual configuration fetched from memory that maps to a single register within a DMA channel.

Descriptor Fetch

The action of retrieving descriptors from memory through memory read operations and loading then into the DMA channel registers upon their read return.

Descriptor Set

A group of descriptors associated with a single work unit.

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 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 will skip 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. In the case of 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 or work units separated by a stop or disabled state. The work units in the chain are programmed to another descriptor flow. The last work unit in the chain is programmed to a flow of STOP or AUTO. STOP stops the state at the end of that work unit. AUTO is required to be disabled by disabling the DMA channel. A work unit chain is also known as a descriptor chain.

Block Diagram

The figure shows the functional blocks within the DMA interface.

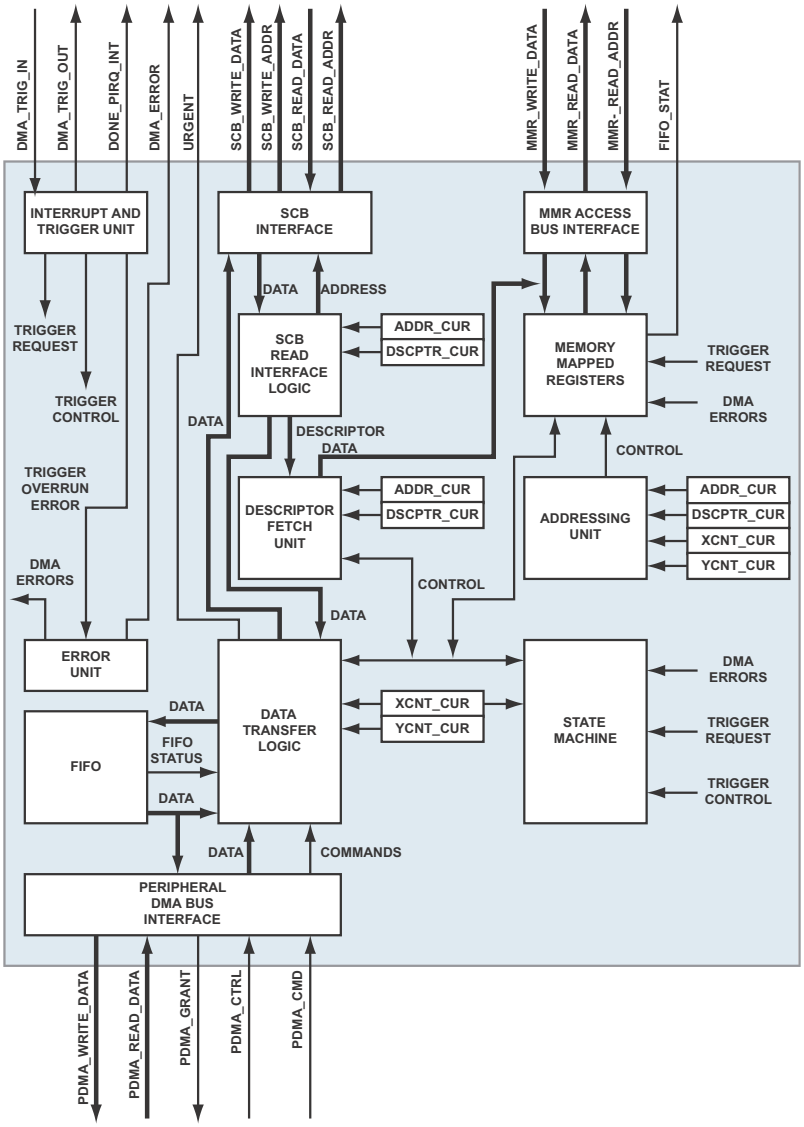


Figure 13-1: DMA Channel Block Diagram

For more information on the interfaces shown in the block diagram, see:

- *DMA Channel Peripheral DMA Bus*
- *DMA Channel MMR Access Bus*
- *DMA Channel Event Control*
- *DMA Channel SCB Interface*
- *DMA Channel List for ADSP-BF60x*

SCB Interface Signals

The DMA channel operates at SCLK frequency as does the SCB interface. The SCB crossbar handles the internal arbitration of the transfer requests of all the masters interfaced to the SCB crossbar instance (see the following table.

Table 13-2: SCB Interface Signals

Signal	Width (bits)	Description
SCB_WRITE_DATA	16/32/64/128	Data bus used for write operations. The width of the bus can be determined from DMA_STAT.MBWID
SCB_WRITE_ADDRESS	32	Write address bus. Provides the address of the first transfer in a burst transaction
SCB_READ_DATA	16/32/64/128	Data bus used for read operations. The width of the bus can be determined from DMA_STAT.MBWID
SCB_READ_ADDRESS	32	Read address bus. Provides the address of the first transfer in a burst transaction

DMA Channel Peripheral DMA Bus

The peripheral DMA bus connects the DMA channel to a peripheral or another DMA channel.

The DMA channel connects to peripherals or other DMA channels via the peripheral DMA bus. This 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 may vary and are not configurable. The assigned bus width can be determined by reading the DMA_STAT.PBWID field.

The DMA channel operates at SCLK frequency as does the peripheral DMA bus. The following table provides descriptions of the peripheral DMA bus signals.

Table 13-3: Peripheral DMA Bus Signals

Signal	Width (bits)	Description
PDMA_WRITE_DATA	8/16/32/64	Data bus used for write operations. The width of the bus can be determined from DMA_STAT.PBWID
PDMA_READ_DATA	8/16/32/64	Data bus used for read operations. The width of the bus can be determined from DMA_STAT.PBWID
PDMA_DMA_GRANT		Control signals to indicate that data is valid for DMA channel read operations (peripheral transmit) and that the DMA channel is ready to receive data for write operations (peripheral receive)

Table 13-3: Peripheral DMA Bus Signals (Continued)

Signal	Width (bits)	Description
PDMA_CMD	3	Used by the peripheral for issuing DMA channel control commands
PDMA_CTRL		The control signals used by the peripheral to send various commands to the DMA channel and control the direction of flow

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 following table provides descriptions of the MMR access bus signals.

Table 13-4: 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 that is used to select the MMR to access

Event Signals

The following table provides descriptions of DMA channel events.

Table 13-5: 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 `DMA_STAT.ERRC` bit.
DONE_PIRQ_INT	1	Signal used to indicate DMA completions events, PIRQ events and also for forwarding PDR events based on configuration. The source of the event may be determined by reading the corresponding fields in `DMA_STAT`.
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.

Architectural Concepts

The DMA channel provides a means to transfer data between memory spaces or between memory and a peripheral using a number of system interfaces. The DMA channel provides an efficient means 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 that 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 so that the DMA channel can perform work unit data transfers with memories such as L1, internal L2 and external L3. In addition to work unit data transfers, the SCB interface is also used for fetching descriptor sets for all the descriptor based transfer modes.

The DMA channel is capable of supporting data bus widths of 16, 32, 64 or 128-bits. The data bus widths for a given DMA channel on a specific processor may vary and are not configurable. The assigned bus widths can be determined by reading the `DMA_STAT.MBWID` field.

SCB Interface Signals

The DMA channel operates at `SCLK` frequency as does the SCB interface. The SCB crossbar handles the internal arbitration of the transfer requests of all the masters interfaced to the SCB crossbar instance (see the following table.

Table 13-6: SCB Interface Signals

Signal	Width (bits)	Description
SCB_WRITE_DATA	16/32/64/128	Data bus used for write operations. The width of the bus can be determined from `DMA_STAT.MBWID`
SCB_WRITE_ADDRESS	32	Write address bus. Provides the address of the first transfer in a burst transaction
SCB_READ_DATA	16/32/64/128	Data bus used for read operations. The width of the bus can be determined from `DMA_STAT.MBWID`
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 DMA channel's configurable memory size for the work unit and the fixed bus width of the DMA channel's SCB data bus.

- If the DMA channel is configured such that the memory transfer size is less than or equal to the DMA channels bus width, then the burst length is always 1.
- If the configured memory size is greater then the SCB interface bus width, then the burst length is sufficient to transfer a transaction as specified by the configured memory size.

Table 13-7: DMA Channel SCB Burst Lengths

Configured Memory Size	Burst Length			
	16-bit Bus	*32-bit Bus*	*64-Bit Bus*	*128-bit Bus*
1 Bytes	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

In order to prevent addressing errors and maximize bandwidth of the SCB interface to the DMA channel, data addresses must be aligned to be a multiple of the programmable memory size of the DMA channels configuration as shown in *Descriptor Set Address Alignment*.

There are situations in which entire work units cannot be transferred at the maximum configurable memory size. In this case the entire work unit may be fulfilled by reducing the configured memory size at the expense of bus bandwidth. Through the use of descriptor sets:

- 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 may then be used to transfer a bulk of the data in the work unit.
- Finally a third descriptor set may be used with a smaller memory size in order to complete any final data transfers that may not meet the alignment requirements of the previous descriptor set configuration.

Table 13-8: DMA Channel Address Alignment Requirements

Configured Memory Size	Address Restriction
1 Byte	No restriction
2 Bytes	ADDR[0] == 0
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 be aligned to a 32-bit address. The memory size of the DMA channel's configuration is ignored for descriptor set fetches, which avoids the need to align descriptor sets based on the previous descriptor set's memory width configuration.

For descriptor sets containing only a single descriptor the transfer takes place as a single 32-bit transfer. For descriptor sets containing multiple descriptors, each 32-bit descriptor is fetched individually and treated as multiple 32-bit transfers.

DMA Channel Peripheral DMA Bus

The peripheral DMA bus connects the DMA channel to a peripheral or another DMA channel.

The DMA channel connects to peripherals or other DMA channels via the peripheral DMA bus. This 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 may vary and are not configurable. The assigned bus width can be determined by reading the `DMA_STAT.PBWID` field.

The DMA channel operates at `SCLK` frequency as does the peripheral DMA bus. The following table provides descriptions of the peripheral DMA bus signals.

Table 13-9: Peripheral DMA Bus Signals

Signal	Width (bits)	Description
PDMA_WRITE_DATA	8/16/32/64	Data bus used for write operations. The width of the bus can be determined from `DMA_STAT.PBWID`
PDMA_READ_DATA	8/16/32/64	Data bus used for read operations. The width of the bus can be determined from `DMA_STAT.PBWID`
PDMA_DMA_GRANT		Control signals to indicate that data is valid for DMA channel read operations (peripheral transmit) and that the DMA channel is ready to receive data for write operations (peripheral receive)

Table 13-9: Peripheral DMA Bus Signals (Continued)

Signal	Width (bits)	Description
PDMA_CMD	3	Used by the peripheral for issuing DMA channel control commands
PDMA_CTRL		The control signals used by the peripheral to send various commands to the DMA channel and control the direction of flow

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 to provide greater control over the DMA channel operation. This control improves real-time performance and relieves control and interrupt demands on the core. Peripherals may 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.

While these DMA control commands (see the following table) are not visible to or controlled by the program, their use by a peripheral has implications for the structure of the DMA transfers which that peripheral can support. It is important that application software be written to comply with certain restrictions regarding work units and descriptor chains so that the peripheral operates properly whenever it issues DMA control commands.

The following table describes the commands that are given by the DMA controller. These commands are described in more detail in the following sections.

Table 13-10: 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
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

This command is driven by the DMA channel when the peripheral is enabled and no data requests are required.

Restart Command

This command causes the current work unit to interrupt processing and star again, using the addresses and count values from the DMA_ADDRSTART, DMA_XCNT and DMA_YCNT registers. No interrupt is signalled when the work unit terminates.

If a channel programmed to transmit (memory read) receives a restart command, the channel momentarily pauses while any pending memory reads initiated prior to the Restart command are completed. During this period of time, the channel does not grant DMA requests. Once all pending reads have been flushed from the channel's pipelines, the channel resets its counters and FIFO, and starts pre fetch reads from memory. DMA data requests from the peripheral are granted as soon as new pre fetched data is available in the DMA FIFO. In this case the peripheral can use the Restart command to reattempt a failed transmission of a work unit.

If a channel programmed to receive (memory write) receives a restart command, the channel stops writing to memory, discards any data held in its DMA FIFO, and resets its counters and FIFO. As soon as this initialization is complete, the channel again grants DMA write requests from the peripheral. In this case the peripheral can use the restart command to abort the transfer of received data into a work unit, and reuse the memory buffer for a later data transfer.

The restart control command request is not granted/acknowledged. The request is always accepted by the DMA controller.

Finish Command

The finish command causes the current work unit to terminate processing and move on to the next work unit. An interrupt/trigger event is signalled as usual, (if enabled within the DMA_CFG register). The peripheral can then use the finish command to partition the DMA stream into work units on its own, perhaps as a result of parsing the data currently passing though its supported communication channel, without direct real-time control by the processor.

If a channel is programmed to transmit (memory read) operation and receives a finish command, the channel momentarily pauses while any pending memory reads initiated prior to the finish command are completed. During this period of time, the channel does not grant DMA requests. Once all pending reads have been flushed from the channel's pipelines, the channel signals an interrupt/trigger (if enabled), and begins fetching the next descriptor (if any). DMA data requests from the peripheral are granted as soon as new pre fetched data is available in the DMA FIFO.

If a channel programmed to receive (memory write) receives a finish command, the channel stops granting new DMA requests while it drains its FIFO. Any DMA data received by the DMA channel prior to the finish command is written to memory. When the FIFO reaches an empty state, the channel signals an interrupt/trigger (if enabled) and begins fetching the next descriptor (if any). Once the next descriptor has

been fetched, the channel initializes its FIFO, and then resumes granting DMA requests from the peripheral.

The finish command request is not granted/acknowledged. The request is always accepted by the DMA channel.

Interrupt Command

The interrupt command causes the DMA channel to generate an interrupt. When programming the channel to support this command, the DMA_CFG.INT bit field must be configured to PIRQ mode so that the channel does not generate interrupts based on work unit state, but instead generates interrupts only when it receives the interrupt command from the peripheral. When the interrupt command is received, the event is indicated in the DMA_STAT.PIRQ bit if all of the following conditions are satisfied.

- The DMA channel is enabled as per the DMA_CFG.EN bit.
- The DMA channel is in the stop state.
- The interrupt in DMA_CFG.INT is configured for PIRQ mode.

The peripheral only issues the interrupt command in response to the last grant command being received from the DMA channel which indicates that the transfer is the last transfer in the work unit.

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/acknowledged by the DMA channel.

Request Data Urgent Command

The request data urgent command behaves identically to the request data command, except that while it is asserted the DMA channel performs its memory accesses with urgent priority. This includes both data and descriptor fetch memory accesses. A DMA management capable peripheral might use this control command if, for example, an internal FIFO is approaching a critical condition.

The request is held by the peripheral until granted/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. These restrictions are described in the following sections.

Transmit Restart or Finish

No restart or finish control command may be issued by a peripheral to a channel configured for memory read unless all the following conditions are met.

- The peripheral has already performed at least one DMA transfer in the current work unit.
- The current work unit has (FIFO_SIZE/DMA_CFG.MSIZE) + 1 memory transfers remaining.

The first item ensures that the work unit has started. The second item ensures that the work unit has not completed. The second item is sufficiently large that it is always at least five more than the maximum data count prior to any restart or finish command. This implies that any work unit which might be managed by restart or finish commands must have DMA_XCNT_CUR and DMA_YCNT_CUR register values representing at least five data items.

The second item can be satisfied by ensuring that the number of memory transfers described by the descriptor is (FIFO_SIZE/DMA_CFG.MSIZE) + 1 larger than the maximum number of memory transfers expected.

Receive Restart or Finish

No restart or finish control command may be issued by a peripheral to a channel configured for memory write unless either of the following conditions is met.

- The number of peripheral transfers completed is less than (DMA_CFG.MSIZE/DMA_CFG.PSIZE) × (transfers described by descriptor)

In addition to either of the above two conditions, one of the following two conditions must also be met.

- The previous work unit was terminated by a finish command AND the peripheral has done at least one transfer in the current work unit.
- The peripheral has done (FIFO_SIZE/DMA_CFG.PSIZE) + 1 transfers in the current work unit.

The first set of conditions ensures that the descriptor is still active while the second set ensures that data from the previous descriptor has left the FIFO and that the current descriptor has started.

Finish Only

The peripheral has completed exactly (DMA_CFG.MSIZE/DMA_CFG.PSIZE) × (transfers described by descriptor) and gives the restart/finish command immediately in the next cycle following the last data transfer.

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.

Memory DMA (MDMA) channels are implemented by interfacing two DMA channels via the peripheral DMA bus interface. One DMA channel is used for memory read operations and the second is used for

memory writes. Depending on the processor, a memory DMA channel may have an additional peripheral, such as a CRC peripheral, inserted into the peripheral DMA bus that may optionally be enabled.

MDMA channel configurations that do not involve an additional peripheral impose no restrictions on which of the DMA channels is to be used for the read operation and which is to be used for the write operation so long as both are not configured for the same transfer direction. For MDMA channel configurations that enable a peripheral between the read and write channels, restrictions may be imposed on which channel may be used for a given transfer direction.

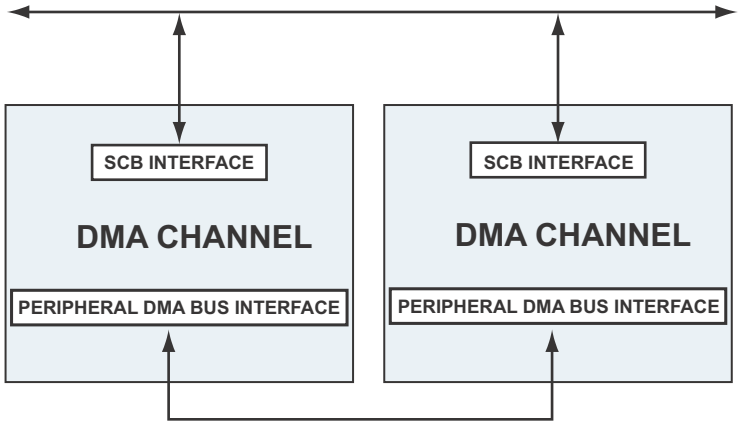


Figure 13-2: MDMA Channel Dedicated Pair

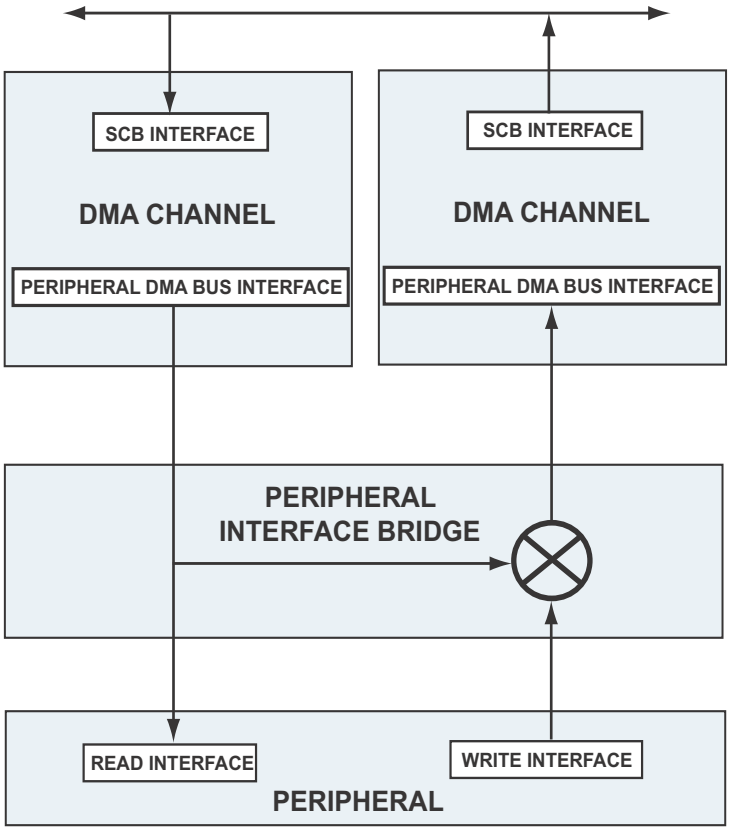


Figure 13-3: MDMA Channel Pair with Peripheral

A memory-to-memory transfer always requires the source and destination channels to be enabled. Because the channels are interfaced through the peripheral DMA bus, and because the channel may have an additional peripheral inserted into the peripheral DMA bus, programs must ensure that the `DMA_CFG.PSIZE` of both the source and destination channels are set to the same values.

The memory DMA channels support the full range of `DMA_CFG.MSIZE` options for the DMA transfers to and from the memories.

As 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 allows for more efficient bursting of both read and write transactions in order to make use of the available bandwidth. While the `DMA_CFG.PSIZE` configuration must be identical for both source and destination DMA channels, this restriction is not imposed for the `DMA_CFG.MSIZE` configuration.

The independent source and destination DMA channels also have their own dedicated interrupt and trigger events, and while it is normal practice to only have event generation performed at destination DMA completion, programs are not restricted to this means of interrupt generation.

Configuration of an MDMA transfer is done in a similar manner to peripheral DMA transfers with the exception of writing two DMA channel registers instead of one.

To control the pace of data transfers, triggers may be used on either the memory read or the memory write channel pair used in an MDMA operation. Enabling `DMA_CFG.TWAIT` in the memory read channel will prevent both channels from transferring data before the system is ready. However, only configuring the memory write channel to wait for a trigger will allow for data to be fetched from the memory in anticipation of the memory write operation.

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 following table provides descriptions of the MMR access bus signals.

Table 13-11: 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 that is used to select the MMR to access

DMA Channel Operation Flow

The flow of operation of the DMA channel is described in the following topics:

- *Startup*
- *Refresh*
- *DMA Operating Modes*
- *Stop Mode*
- *DMA Channel Errors*

Startup

In order to enable a DMA operation on a given channel, some or all of the DMA parameter registers must first be written directly. The minimum set of register required to be initialized is dependent upon the desired mode of operation as described in the following sections.

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 DMA channels desired operating mode.

- For descriptor array based flow modes – At a minimum the DMA_DSCPTR_CUR register must be written 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 a minimum the DMA_DSCPTR_NXT register must be written 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 – The DMA_ADDRSTART, DMA_XCNT and DMA_XMOD registers must be written prior to the DMA_CFG register.

Programs can write other registers that might remain static throughout the course of the DMA activity. The DMA operation begins once the DMA_CFG register is written.

ATTENTION: When the DMA_CFG register is written directly by software, the DMA controller recognizes this as the special startup condition that occurs when starting DMA for the first time on this channel or after the DMA channel is stopped. It is possible for a DMA error condition to be flagged regardless of the DMA_CFG.EN bit setting.

Startup Operation

When the DMA_CFG register is written directly by software, the DMA channel recognizes this as the special startup condition that occurs when starting DMA for the first time on this 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 that was read into the DMA_CFG register assumes control. Before this point, the direct write to the DMA_CFG register had control.

At startup, the selected flow mode and the and 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, while the DMA_CFG.NDSIZE field details how many descriptor elements to fetch before starting the DMA. DMA registers not included in the descriptor are not modified from their prior values.

For descriptor list flow modes, the DMA_DSCPTR_NXT register is copied into the DMA_DSCPTR_CUR register. Then, fetches of new descriptor elements from memory are performed, indexed by the DMA_DSCPTR_CUR register, which is incremented 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, then the DMA_DSCPTR_NXT register is not copied 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, then 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, then the value programmed by the MMR access controls only the loading of the first descriptor fetched from

memory. The subsequent DMA work units are controlled by the configuration of the DMA_CFG register of the fetched descriptor set.

Once the descriptor fetch is complete, or if the flow was originally configured for one of the register based flow modes, then the DMA operation begins. The DMA channel immediately fills its FIFO. For a memory write operation the DMA channel begins accepting data from its peripheral. For a memory read operation the DMA channel begins memory reads when the DMA channel is granted access to the SCB bus.

When the DMA channel performs its first data memory access, its address and count computations take their input operands from the start registers (DMA_ADDRSTART, DMA_XCNT and DMA_YCNT if required), and writes results back to the current registers (DMA_ADDR_CUR, DMA_XCNT_CUR and DMA_YCNT_CUR). Note also that the current registers are not valid until the first memory access is performed, which may be some time after the channel is started by the write to the DMA_CFG register. The current registers are loaded automatically from the appropriate descriptor elements, overwriting their previous contents, as follows:

- DMA_ADDRSTART is copied to DMA_ADDR_CUR
- DMA_XCNT is copied to DMA_XCNT_CUR
- DMA_YCNT is copied to DMA_YCNT_CUR

Refresh

When a work unit has been processed (is complete), the DMA channel performs the following operations:

- Completes the transfer of all data between memory and the DMA channel.
- If the DMA channel is configured for a memory read operation with the DMA_CFG.SYNC bit enabled, then a synchronized transition takes place. The DMA channel transfers all data to the peripheral before continuing.
- If interrupts/triggers are enabled, then the signals are forwarded from the DMA channel and the DMA_STAT register is updated to indicate the interrupt/trigger events.
- If the flow was set to stop mode, the DMA operation stops by setting the DMA_STAT.RUN bit field to indicate the channel is no longer running. Any remaining data in the DMA channel's FIFO is transferred to the peripheral.
- For descriptor array mode – Loads a new descriptor from memory into the DMA registers by way of the contents of the DMA_DSCPTR_CUR register, while incrementing the DMA_DSCPTR_CUR register. The descriptor size is taken from the DMA_CFG.NDSIZE value prior to the fetch.
- For descriptor list mode – Copies the DMA_DSCPTR_NXT register into the DMA_DSCPTR_CUR register. Next, the DMA channel fetches the descriptor from the new contents of the DMA_DSCPTR_CURregister and places these contents into the DMA registers while incrementing the DMA_DSCPTR_CUR register.

- For descriptor on demand array mode – Checks to see if an incoming trigger event has been detected.

  If a trigger event has been detected, then 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 descriptor size is taken from the DMA_CFG.NDSIZE value prior to the fetch.

  If a trigger event was not detected then the DMA channel begins the next work unit by reloading the current registers as described below.

- For descriptor on demand list mode – Checks to see if an incoming trigger event has been detected.
  - If a trigger event was detected, then the DMA channel copies the DMA_DSCPTR_NXT register into the DMA_DSCPTR_CUR register. Next, the DMA channel fetches the descriptor memory 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.
  - If a trigger event was not detected then the DMA channel begins the next work unit by reloading the current registers as described in the step below.
- If flow was configured for anything other than stop mode then the DMA channel begins the next work unit 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 Transitions

Transitions from one work unit to the next are controlled by DMA_CFG.SYNC bit for a given 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 DMA FIFO pipeline is handled while the next descriptor is fetched.

In continuous transitions where synchronization is disabled, the DMA FIFO pipeline continues to transfer data to and from the peripheral or destination memory during the descriptor fetch and/or when the DMA channel is paused between descriptor chains. On the other hand, 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, at the cost of higher latency in the transition. In synchronized transitions, the DMA FIFO pipeline is drained to the destination or flushed (received data discarded) between work units.

NOTE: Work unit transitions for MDMA streams are controlled by the DMA_CFG.SYNC bit of the MDMA source channel. The DMA_CFG.SYNC bit of the MDMA destination channel is reserved and must be set to disabled state. In transmit (memory read) channels, the DMA_CFG.SYNC bit of the last descriptor prior to 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.

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 and the handling of the DMA FIFO between the current and the next work unit.

If the DMA_CFG.SYNC bit is configured to disable synchronization, a continuous transition is selected. In a continuous transition, just after the last data item is read from memory, these four operations all start in parallel.

- The interrupt/trigger (if any) is signalled.
- The DMA_STAT register is updated to indicate DMA status is completed.
- The next descriptor begins to be fetched.
- The final data items are delivered from the DMA FIFO to the destination memory or peripheral.

This allows the DMA channel to provide data from the FIFO to the peripheral continuously during the descriptor fetch latency period.

When synchronization is disabled, the final interrupt/trigger (if enabled) occurs when the last data is read from memory. This event occurs at the earliest time that the output memory buffer may safely be modified without affecting the previous data transmission. There may be a number of data items still remaining in the FIFO and not yet at the peripheral. This number is dependent on the FIFO depth of the DMA channel. Therefore, in this configuration, the DMA interrupt should not be used as the sole means of synchronizing the shutdown or re configuration of the peripheral following a transmission.

NOTE: If continuous transition is selected on a transmit (memory read) descriptor, the next descriptor is required to have the same peripheral transfer size (DMA_CFG.PSIZE), read/write direction, and source memory (internal versus external) as the current descriptor.

Disabling synchronization, to select continuous transition on a work unit that is configured for stop flow mode with interrupts/triggers enabled, can result in the event service routine being executed while the final data is still draining from the FIFO to the peripheral. This is indicated by the DMA channels DMA_STAT.RUN bits—if the channel is still running then the FIFO is not yet empty. Do not start a new work unit with different peripheral transfer size or direction while the channel is still running. Further, if the channel is disabled via the DMA_CFG.EN bit, the data in the FIFO is lost.

A synchronized transition allows the DMA FIFO to first be drained to the destination memory or peripheral before any interrupt is signalled, and before any subsequent descriptor or data is fetched. This incurs greater latency, but provides direct synchronization between the DMA interrupt and the state of the data at the peripheral.

For example, if synchronization is enabled and interrupts are enabled on the last descriptor in a work unit, the interrupt occurs when the final data is transferred to the peripheral. This allows the service routine to properly switch to non-DMA transmit operation. When the interrupt service routine is invoked, the DMA channel FIFO is empty and the DMA channel is not running as 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 may have any DMA_CFG.PSIZE configuration or read/write direction supported by the peripheral and may come from either memory space (internal as opposed to external). This 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 (memory write) channels, the DMA_CFG.SYNC bit controls the handling of the DMA FIFO between descriptor chains (not individual descriptor sets), when the DMA channel is paused. The DMA channel pauses after descriptor sets configured with stop flow mode complete, and the channel may be restarted (for example, after an interrupt) by writing the channel's DMA_CFG register with a value that enables the DMA channel. If the synchronization is disabled in the new work unit's DMA_CFG value, a continuous transition is selected. In this mode, any data items received into the DMA FIFO while the channel was paused are retained, 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 that no data items are dropped during a DMA pause, at the cost of certain restrictions on the DMA descriptors.

NOTE: If the DMA_CFG.SYNC bit is configured to disable synchronization on the first descriptor of a descriptor chain after a DMA pause, the DMA_CFG.PSIZE field of the new chain must not change from the configuration of the previous descriptor chain that was active before the pause, unless the DMA channel is reset between chains by disabling and then re-enabling the DMA channel.

A synchronized transition is selected if the DMA_CFG.SYNC bit is configured to enable synchronization. In this mode, only the data received from the peripheral by the DMA channel after the write to the DMA_CFG register is delivered to memory. Any prior data items transferred from the peripheral to the DMA FIFO before this register write are discarded. This provides direct synchronization between the data stream received from the peripheral and the timing of the channel restart (when the DMA_CFG register is written).

For receive DMA operations, the synchronization has no effect in transitions between work units in the same descriptor chain (that is, when the previous descriptor's flow mode was not stop, so that 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 to 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 (memory write) channel, there is no restriction on changes of peripheral transfer size (internal versus external) between descriptors or descriptor chains.

Transfer Termination and Shutdown

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 a list or array of descriptors is used to control DMA transfers, and if every descriptor contains a DMA_CFG descriptor element, then the final DMA_CFG descriptor element should have the flow configured to stop mode setting to gracefully stop the channel. Upon completion the DMA channel remains in the stop state. This state

should not be confused with the disabled state which occurs either due to a DMA error or by configuring the DMA_CFG.EN bit so as to disable the DMA channel.

Disabling the DMA channel via a write to the DMA_CFG.EN bit is intended to shut down the DMA channel and enter the disabled state. All memory and peripheral data transfers cease and only peripheral interrupts are passed through the DMA channels interrupt signals. However, the DMA channel maintains the DMA_STAT.RUN bits. Therefore, in the case of a memory write operation, the outstanding memory transaction counter keeps track of returning memory write acknowledgements and updates as required.

In the case of memory read operations, the outstanding memory transaction counter also keeps track of returning memory reads. However, the memory reads are not written into the FIFO. The counter is updated to reflect the completion of the transaction, but the data is ignored. The DMA_STAT.RUN bits remain in the *waiting for write ACK/FIFO drain to peripheral* state and do not change to *stop/idle state* until all outstanding transactions have returned.

When the DMA channel is enabled again via the DMA_CFG.EN bit, a full reset is performed and all counters are cleared. If an outstanding memory transaction returns an acknowledgement or read data after this event, then a memory transaction error has occurred and an error is generated. Programs must ensure that all outstanding memory transactions have been completed before re configuring the DMA channel. One method programs may use is to poll the DMA_STAT.RUN bits to return to the stop/idle state before proceeding.

Autobuffer Flow Mode

In the case of Autobuffer flow modes, the only way to cease operations is to disable the DMA channel via the DMA_CFG.EN bit. Therefore, one method of changing to a new work unit would be to disable the DMA channel, set up all the registers (and descriptors in memory, if used) except for DMA_CFG, then poll DMA_STAT.RUN to wait for the status to reflect stop/idle state, and finally write DMA_CFG to the new configuration to begin the next work unit.

In autobuffer flow mode, or if a list or array of descriptor sets without DMA_CFG descriptors, then the DMA transfer process must be terminated by an MMR write to the DMA_CFG register with a value whose DMA_CFG.EN bit is configured to disable the DMA channel.

CAUTION: Interrupt logic based on work unit transitions are disabled when the DMA channel is disabled. Programmers should be aware of their environment and current actions so that additional interrupts are not required from the DMA channel.

CAUTION: The DMA channel completes any transactions that have begun and avoids generating bus errors if disabled through DMA_CFG.EN in the middle of a transaction. However, the action of re-enabling the DMA is considered a hard reset for all internal DMA channel components. Therefore, programmers must pay special attention to that particular action in order to avoid unexpected results.

DMA Channel Errors

When an error occurs, the DMA channel maintains all the state and register values which allows 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.

It is the responsibility of the programmer to take special care to ensure the root cause of the error is addressed, whether the problem originated in the DMA channel or not. If not properly resolved, the error could result in an additional error shortly after operations resume. The problem may have caused other errors elsewhere in the DMA channel or associated modules and circuitry, therefore care must be taken to address those potential problems also. Finally, the programmer must ensure that all outstanding memory reads and writes are complete, or cleared, before resuming DMA channel operation.

Once all issues have been addressed and all side effects of any error are neutralized, the programmer may 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 error types are described in the following sections.

Status and Debug

DMA channel error conditions can cause the DMA process to end abnormally. DMA error detection is provided as a tool for system development and debug, as a way to detect DMA related programming errors. When the DMA channel detects and error, the channel is immediately stopped and any memory read transactions that are returned are discarded. The DMA channels DMA_STAT.RUN field is set to indicate idle state, once all outstanding memory transactions are acknowledged. In addition, an error interrupt is asserted and theDMA_STAT.IRQERR is updated to reflect this. Also the error cause of the first detected error is updated in the DMA_STAT.ERRC field. Unless the error occurs at the exact moment that register values are being modified, the registers will contain their values.

It is possible for error interrupt signals to be combined. Combined error signals requires that the DMA_STAT register of each DMA channel associated with a combined error interrupt be read to determine the DMA channel responsible for the generation of the interrupt.

The DMA channel error interrupt handler is required to perform the following actions:

- Read each DMA channel's DMA_STAT register to look for a channel with the DMA_STAT.IRQERR set to indicate an error.
- Read the DMA channel's DMA_STAT.ERRC field to determine 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 via a write 1 to clear operation to the DMA_STAT.IRQERR bit.

If any error other than a bandwidth monitor error is already flagged and is not cleared, no other error is reported. If a bandwidth monitor error was reported and not cleared, any newly detected error would be in the updated DMA_STAT.ERRC field.

DMA Configuration Register Errors

These errors are only flagged when the DMA channel is enabled via the DMA_CFG.EN bit.

- Reserved setting was used
- DMA_CFG.TWAIT enabled in Descriptor On Demand Flow mode
- Illegal DMA_CFG.NDSIZE
- Illegal DMA_CFG.MSIZE
- DMA_CFG.MSIZE exceeds the DMA channel's FIFO size
- Illegal DMA_CFG.PSIZE
- DMA_CFG.PSIZE exceeds the FIFO size
- DMA_CFG.PSIZE exceeds the bus width
- Memory read (transmit operation), cannot change to receive unless properly synced in the previous work unit, or if first work unit in a new chain
- Memory read (transmit operation), cannot change DMA_CFG.PSIZE unless properly synced in previous work unit, or if first work unit in a new chain
- Memory write (receive operation), cannot change to transmit during a descriptor chain. Can only change from receive to transmit if new transmit is synced and first work unit
- Memory write (receive operation), cannot change DMA_CFG.PSIZE unless first work unit with DMA_CFG.SYNC enabled

Illegal Register Write During Run

Writes to writable registers when the DMA channel is enabled and running are blocked and generate an error. The DMA_STAT, DMA_BWLCNT and DMA_BWMCNT registers are exempt from this behavior.

Address Alignment Error

An address alignment error is generated when a descriptor address is not aligned to a 32-bit boundary or a transfer address is not aligned for the current DMA_CFG.MSIZE configuration.

Memory Access Error

A memory access error is generated when an attempt was made to access an address not populated, defined as cache, or there was a security violation. This error is triggered by an error returned from the memory.

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

This error is generated when the bandwidth monitor count expired. This is not a fatal error and the DMA channel continues operation.

Control Interface Error

Control interface errors are reported as bus errors to the bus master. This can be as a result of:

- An address 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 may include the descriptor responsible for configuring the flow of the work unit and there is no restriction that the flow must be configured the same for the entire descriptor chain. If the descriptor chain is not endless then the last descriptor set configures the flow to stop mode which results in termination of the descriptor chain after work unit completion. Another example for mixing flow modes is to create an endless descriptor array. The last descriptor set in the array is configured for list mode and the next descriptor pointer of this descriptor set points to the first descriptor in the array.

Register Based Flow Modes

Register-based DMA operations require configuration by directly writing to the DMA channel's memory-mapped registers.

Register-based DMA is the traditional method of DMA operation. Software writes all of the DMA channel's configuration into the memory-mapped registers. This includes information such as the source or destination address and length of the data to be transferred. 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 for the flow mode of the next descriptor set to be altered allowing for the transition from descriptor array mode to descriptor list mode, in addition to configuring the flow to stop or auto-buffer mode.

Stop Mode

In stop mode, the DMA operation is executed 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. Interrupts and triggers may also be generated for each row/work unit completion, depending on the desired operation.

Autobuffer Mode

In autobuffer mode, the DMA operates repeatedly in a circular manner. If all data words have been transferred, the address pointer (`DMA_ADDR_CUR`) is reloaded automatically with the `DMA_ADDRSTART` value. An interrupt may also be generated.

Autobuffer mode is enabled via the `DMA_CFG.FLOW` field. The `DMA_CFG.NDSIZE` field must be configured 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 is not required 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 descriptor sets can be fetched from any memory space that supports DMA read operations.
- The descriptor set describes the next operation to be performed by the DMA controller.
- The descriptor set may include information such as the DMA configuration word as well as data source/destination address, transfer count, and address modify values.

A descriptor set describes a single work unit. However some values from one descriptor set may be reused in the next work unit if they are not overwritten in the subsequent descriptor set fetches and 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 the flow mode of the next descriptor set to be altered. 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. The initial descriptor pointer value is written by software and points to an array of descriptors. The individual descriptors are assumed to reside next to each other and, therefore, their address is known.

The following table illustrates how a descriptor set must be structured in memory. Note that descriptor sets must reside in a contiguous block or memory in the format shown in the table. That is to say that the first descriptor of the next descriptor set must be located in the memory location immediately following the last descriptor of the current descriptor set. The values have the same order as the corresponding MMR offset addresses.

Table 13-12: Descriptor Array Mode Parameter and Descriptor Offsets

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. All of the current registers are reloaded between the descriptor set fetch and the start of the DMA operation for the work unit.

NOTE: At a minimum the `DMA_DSCPTR_CUR` register must be written 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 descriptor set is fetched, this pointer value is loaded into the DMA channels next descriptor pointer register.

Descriptor Sets

The *Descriptor List Mode Parameter and Descriptor Offsets* illustrates how a descriptor set must be structured in memory. Note that while the descriptor sets can be dispersed throughout memory and reside in different memory blocks, 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 MMR offset addresses.

Table 13-13: Descriptor List Mode Parameter and Descriptor Offsets

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. All of the register's current values are reloaded between the descriptor set fetch and the start of the DMA operation for the work unit.

Minimum Startup Requirements

At a minimum the `DMA_DSCPTR_NXT` register must be written 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 current work unit is repeated. If the DMA channel receives an incoming trigger before completion of the work unit, a new descriptor set is fetched.

The following tables illustrate how each descriptor set must be structured in memory.

Table 13-14: Descriptor Array Mode Parameter and Descriptor Offsets

Descriptor Offset	Parameter Register
0x00	DMA_ADDRSTART
0x04	DMA_CFG
0x08	DMA_XCNT
0x0C	DMA_XMOD
0x10	DMA_YCNT
0x14	DMA_YMOD

Table 13-15: Descriptor List Mode Parameter and Descriptor Offsets

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 the DMA_DSCPTR_NXT register must be written prior to write to the DMA_CFG register, which is the special action required to start the DMA channel.

NOTE: For descriptor array mode, at a minimum the DMA_DSCPTR_CUR register must be written prior to writing 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 and descriptor-based DMA flow modes support two-dimensional data transfers.

In two-dimensional (2D) mode the X directional count and modifier (DMA_XCNT and DMA_XMOD) registers are accompanied by the Y directional count and modifier (DMA_YCNT and DMA_YMOD) registers, supporting arbitrary row and column sizes. Furthermore, modify values can be negative, allowing implementation of interleaved data streams. DMA_XCNT and DMA_YCNT specify the row and column sizes respectively, where the DMA_XCNT must be 2 or greater.

The DMA start address (DMA_ADDRSTART) register, along with DMA_XMOD and DMA_YMOD registers, are all specified in bytes, and they must be aligned to a multiple of the DMA transfer word size as configured by the DMA_CFG.MSIZE bit. Misalignment results in a DMA channel error.

The DMA_XMOD register value is the byte-address increment that is applied after each transfer that decrements the DMA_XCNT register. The DMA_XCNT register is not applied when the inner loop count is ended by the DMA_XCNT_CUR register decrementing to 0 from 1, except that it is applied 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 is applied after each decrement of the value in the DMA_YCNT_CUR register. However, the DMA_YMOD value is not applied 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. Note that if the DMA channel is programmed to refresh automatically such as in autobuffer mode, then both the DMA_XCNT_CUR and DMA_YCNT_CUR registers and the DMA current address (DMA_ADDR_CUR) are reloaded for the first data transfer of the next work unit.

Interrupt notification is configurable for end of row or end of work unit completion.

DMA Channel Event Control

The DMA channel supports a number of events that provide notification of work unit state, peripheral data request, peripheral data 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. DMA channel errors are reported on a dedicated interrupt signal while all other interrupt sources share the same 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.

DMA channel events can be signaled to the processor using status information and optional interrupt requests. These events may be used for data transfer progress updates and to request intervention from the processor core. A majority of DMA channel interrupts are configured using bits in the DMA_CFG register. Dedicated bits in the DMA_STAT register are used to report the occurrence of various events. Interrupt requests are cleared by write-one-to-clear (W1C) operations to the status register.

NOTE: Hardware does not clear the interrupt status bits automatically even if the DMA channel is disabled then re-enabled. In this situation the interrupt signal from the DMA channel is de-asserted once the DMA channel is disabled, but the status bit remains set until the DMA channel is enabled again or cleared by software.

The DMA channel supports the following categories of events on the interrupt signals.

- Work unit state events are used to generate interrupts on row or on work unit DMA completion.
- Peripheral interrupt request (PIRQ) events are signaled by the peripheral when it has completed the transfer of all data.
- Peripheral data request (PDR) events for when the DMA channel is disabled or idle and the peripheral is requesting data from the DMA channel.
- Error events due to a failure in the work unit.

ATTENTION: The DMA channel does not generate an interrupt to the processor for a work unit state event, PIRQ event or forward a PDR event while in an error state.

Event Signals

The following table provides descriptions of DMA channel events.

Table 13-16: 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 DMA_STAT.ERRC bit.
DONE_PIRQ_INT	1	Signal used to indicate DMA completions events, PIRQ events and also for forwarding PDR events based on configuration. The source of the event may be determined by reading the corresponding fields in DMA_STAT.
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

Work unit state events are generated as a result of a row or a work unit completion. In order for either of these events to result in the generation of an interrupt, the interrupt of the DMA channel must be configured for one of the available work unit completion modes.

- Current X count reaching 0 for row completion or 1-D DMA work unit completion.
- Current Y count reaching 0 for work unit completion of 2-D DMA.

NOTE: For 1-D DMA, configuring the interrupt to be generated on the current Y counter reaching 0 results in a DMA channel configuration error.

The DMA channel issues the last memory read or write transaction for the row or work unit and then pauses until the read or write acknowledge is returned. Once the transfer has been acknowledged successfully, the DMA channel issues the interrupt and continues to process the next row or work unit.

Waiting for the memory access to be acknowledged 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 may be paused waiting for the memory transfer to be acknowledged, the DMA channel is still capable of fetching the next descriptor set in order to be ready to process the next work unit as soon as the memory access completes.

The interrupt timing is also affected by the synchronization feature of the DMA channel's configuration. For memory read operations with synchronization enabled, the interrupt is delayed further until the last transfer from the DMA channel's FIFO to the peripheral completes. The interrupt timing for memory write operations is not affected by the synchronization feature.

Peripheral Interrupt Request Events

Peripheral interrupt request (PIRQ) events may be used by a peripheral connected to the DMA channel to indicate, in the case of a peripheral transmit operation, that data has not only left the FIFO of the DMA channel, but that the peripheral has also completed the transfer.

In order to support PIRQ interrupts the DMA channel's interrupt must be configured correctly. This disables the generation of interrupts based on work unit state and instead results in generating an interrupt when the DMA channel receives the command from the peripheral.

The interrupt is only generated if the following conditions are satisfied.

- The DMA channel is enabled.
- The DMA channel is in the stop state.
- The DMA channel's interrupt is configured for 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 is either disabled or enabled and in the stop state.

When the DMA channel is disabled and a peripheral sends a command to the DMA channel to request data, the DMA channel generates an interrupt to the System Event Controller (SEC). There is no status information reported about this event in the DMA channel's status register. Instead, the PDR event is identified by the fact that the DMA channel generated an interrupt when it was disabled. Further confirmation can be obtained by verifying the status of the peripheral interfaced to the DMA channel.

In addition to requests for data being forwarded as interrupts when the DMA channel is in the disabled state, the DMA channel is also able to forward PDR events as an interrupt when the DMA channel is in the stop state after 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. Channel triggers can be used in combination with each other in order to create ping-pong buffers or when combined with interrupts to notify the processor that a particular milestone has been reached and that

service is required. Triggers may also be used 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 in the case of a 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 be used to control the pace of the data transfers performed by the DMA channel. The DMA channel can be configured 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 that may be configured 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, and then pauses until the transfer acknowledge is returned. Once the transfer has been acknowledged, the DMA channel issues the trigger before processing the next row or work unit.

Issuing Triggers

The DMA channel can be configured 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, and then pauses until the transfer acknowledge is returned. Once the transfer has been acknowledged the DMA channel issues the trigger before processing the next row or work unit.

NOTE: While the DMA channel may be paused while waiting for the memory transfer to be acknowledged, the DMA channel is still capable of fetching the next descriptor set in order to be ready to process the next work unit as soon as the memory access completes.

Waiting For Triggers

Triggering may be used to control the pace of data transfers performed by the DMA channel. If the `DMA_CFG.TWAIT` bit is enabled and a trigger has been received since the last time the DMA channel left the wait state or since transition from disabled to enabled, then the DMA channel enters a wait state before beginning the next work unit. In this state the DMA channel also does not perform a descriptor fetch. Once a trigger is received, 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 the DMA channel is programmed through a memory mapped register write operation with stop flow mode enabled, the `DMA_CFG.TWAIT` bit enabled, and no trigger having already been received, then the DMA channel enters a wait state before performing the data transfer. Upon 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 the DMA channel is programmed through a memory-mapped register write operation with the flow mode configured to one of the descriptor based modes, then the DMA channel enters the wait state before performing the descriptor fetch. Once the descriptor fetch is complete, the DMA channel immediately proceeds to the data transfer, regardless of the value of the `DMA_CFG.TWAIT` bit. If the descriptor fetch is

followed by another descriptor fetch, then the DMA channel enters a wait state before fetching the next descriptor.

If the descriptor fetch returns a descriptor with stop flow mode then the DMA_CFG.TWAIT value for that descriptor does not affect the DMA as the DMA channel enters the stop state once the data transfer is completed. 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, then DMA_CFG.TWAIT for that descriptor does not affect the DMA for the first work unit of the autobuffer transfer. Once the first work unit is completed and another trigger has not been received, then the DMA channel enters the wait state before re-initializing its counters and address registers (if not configured for current addressing). The next work unit is performed once the trigger is received.

The incoming trigger does not have to be issued after the DMA channel has entered the wait state, and can be issued while the DMA channel is executing a work unit, descriptor fetch or even when in the stop state. The trigger is held internally, and once 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 used the next time DMA_CFG.TWAIT is enabled. This allows programs to enable the DMA_CFG.TWAIT functionality several work units apart and not be concerned with losing a trigger. The DMA channels trigger overrun enable functionality may be enabled in all work units to ensure 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. Such software needs to be able to 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 for each peripheral DMA and memory DMA stream to be managed by a separate task or to be managed together with any other stream. Each DMA channel has independent, orthogonal control registers, resources, and interrupts, so that 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

Configures a peripheral's DMA channel to read data from internal memory and send it to the peripheral for transmission.

PREREQUISITE:

The peripheral is assumed to be 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 in order to configure a DMA channel to read data from internal memory and send it 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_ADDRSTART` register.

   ADDITIONAL INFORMATION: The address may be used to calculate the most optimum `DMA_CFG.MSIZE` possible.

2. Calculate most 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 based on the calculated `DMA_CFG.MSIZE`.

   ADDITIONAL INFORMATION: The `DMA_XCNT` value is the number of `DMA_CFG.MSIZE` transfers required to make up the entire work unit.

4. Write the `DMA_XMOD` register.

   ADDITIONAL INFORMATION: For a completely linear buffer transfer, `DMA_XMOD` is determined by the selected `DMA_CFG.MSIZE`. This register is always specified in the number of bytes.

5. Write the `DMA_CFG` register with `DMA_CFG.EN` configured to enable the DMA channel.

   ADDITIONAL INFORMATION: `DMA_CFG.FLOW` must be set for STOP mode, `DMA_CFG.WNR` must be configured for memory read operation and `DMA_CFG.PSIZE` must be configured to a value no larger than the supported bus width of the peripheral DMA bus.

   - The `DMA_CFG.SYNC` bit may be configured to control DMA completion notification timing.
   - Interrupts and triggers may also be configured at this step depending on requirements.

RESULT:

The DMA channel is now enabled and the buffer is transferred. The DMA channel enters the IDLE state upon completion of the work unit.

Register Based Autobuffer Flow Mode

Configures a peripheral's DMA channel to read data from internal memory and send it to the peripheral for transmission. The transmission of the buffer is repeated endlessly.

PREREQUISITE:

The peripheral is assumed to be 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 in order to configure a DMA channel to read data from internal memory and send it to a peripheral connected to the peripheral DMA bus. On DMA completion the DMA channel starts the DMA operation over again creating an endless circular buffer transfer.

1. Write the `DMA_ADDRSTART` register.

   ADDITIONAL INFORMATION: The address may be used to calculate the most optimum `DMA_CFG.MSIZE` possible.

2. Calculate most 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 based on calculated `DMA_CFG.MSIZE`.

   ADDITIONAL INFORMATION: `DMA_XCNT` is the number of `DMA_CFG.MSIZE` transfers required to make up the entire work unit.

4. Write the `DMA_XMOD` register.

   ADDITIONAL INFORMATION: For a completely linear buffer transfer, `DMA_XMOD` is determined by the selected `DMA_CFG.MSIZE`. This register is always specified in the number of bytes.

5. Write the `DMA_CFG` register with `DMA_CFG.EN` configured to enable the DMA channel.

   ADDITIONAL INFORMATION: `DMA_CFG.FLOW` must be set for AUTOBUFFER mode, `DMA_CFG.WNR` must be configured for memory read operation and `DMA_CFG.PSIZE` must be configured to a value no larger than the supported bus width of the peripheral DMA bus.

   - `DMA_CFG.SYNC` may be configured to control DMA completion notification timing.
   - Interrupts and triggers may also be configured at this step depending on requirements.

RESULT:

The DMA channel is now enabled and the buffer is transferred until the DMA channel is disabled.

Descriptor Array Flow Mode

Configures a peripheral's DMA channel to read data from memory as described by the descriptor sets in the array and send it to the peripheral for transmission. Descriptor sets are read from an array in memory to configure the individual work units.

PREREQUISITE:

The peripheral is assumed to be in a state where it is ready to transmit data received from the DMA channel. The array of descriptors is assumed 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 in order to configure a DMA channel to read the first descriptor set from the array in memory that is responsible for the configuring the DMA channel to retrieve and send the data to a peripheral connected to the peripheral DMA bus. Upon 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 `DMA_CFG.EN` configured to enable the DMA channel.

   ADDITIONAL INFORMATION: `DMA_CFG.FLOW` must be set for DESCRIPTOR ARRAY mode, `DMA_CFG.NDSIZE` must be configured to describe the number of descriptor elements contained within the first descriptor set. `DMA_CFG.WNR` must be configured for memory read operation and `DMA_CFG.PSIZE` must be configured to a value no larger than the supported bus width of the peripheral DMA bus.

   - `DMA_CFG.SYNC` configuration is controlled by the descriptor set that is to be fetched as are interrupt and trigger configurations

   STEP RESULT: The first descriptor set is fetched from memory location provided by `DMA_DSCPTR_CUR` and loaded to the DMA channel's MMR registers.

RESULT:

The DMA channel is now processing all the work units provided in the descriptor array. The DMA channel enters the IDLE state upon completion of the final work unit that was configured for STOP flow mode.

Descriptor List Flow Mode

Configures a peripheral's DMA channel to read data from memory as described by the descriptor sets in the list and send it to the peripheral for transmission. Descriptor sets are read from a list of descriptors in which each descriptor set has a descriptor that points to the next descriptor set location in memory.

PREREQUISITE:

The peripheral is assumed to be in a state where it is ready to transmit data received from the DMA channel. The list of descriptors is assumed to 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 in order to configure a DMA channel to read the first descriptor set from the list in memory that is responsible for the configuring the DMA channel to retrieve and send the data to a peripheral connected to the peripheral DMA bus. Upon 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: `DMA_CFG.FLOW` must be set for DESCRIPTOR LIST mode, and the `DMA_CFG.NDSIZE` bit must be configured to describe the number of descriptor elements contained within the first descriptor set. The `DMA_CFG.WNR` bit must be configured for memory read operation and the `DMA_CFG.PSIZE` bit must be configured to a value no larger than the supported bus width of the peripheral DMA bus.

   - `DMA_CFG.SYNC` configuration is controlled by the descriptor set that is to be fetched as are interrupt and trigger configurations.

   STEP RESULT: The first descriptor set is fetched from the memory location provided by `DMA_DSCPTR_NXT` and loaded to the DMA channel's MMR registers.

RESULT:

The DMA channel is now 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

Configures a memory DMA channel pair in STOP flow mode. One DMA channel is configured for memory read operations while the other is configured for memory write.

PREREQUISITE:

The task involves writing to a number of DMA channels on two DMA channels that create a memory DMA pair. Upon DMA completion the DMA channel enters the idle state until either disabled or reconfigured for a new transfer.

1. Write the `DMA_ADDRSTART` register of the source DMA channel.

   ADDITIONAL INFORMATION: The address may be used to calculate the most optimum `DMA_CFG.MSIZE` possible.

2. Calculate most 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 required to make up the entire work unit.

4. Write the `DMA_XMOD` register of the source DMA channel.

   ADDITIONAL INFORMATION: For a completely linear buffer transfer, `DMA_XMOD` is determined by 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 may be used to calculate the most optimum `DMA_CFG.MSIZE` possible.

6. Calculate most 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 calculated `DMA_CFG.MSIZE`.

   ADDITIONAL INFORMATION: `DMA_XCNT` is the number of `DMA_CFG.MSIZE` transfers required to make up the entire work unit.

8. Write the DMA_XMOD register of the destination DMA channel.

   ADDITIONAL INFORMATION: For a completely linear buffer transfer, DMA_XMOD is determined by 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: DMA_CFG.FLOW must be set for STOP mode, DMA_CFG.WNR must be configured for memory read operation and DMA_CFG.PSIZE must be configured to a value no larger than the supported bus width of the peripheral DMA bus.

   - DMA_CFG.SYNC may be configured to control DMA completion notification timing.
   - Interrupts and triggers may also be configured at this step depending on requirements, generally however they would be enabled within the destination DMA channel configuration.

   STEP RESULT: 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: DMA_CFG.FLOW must be set for STOP mode, DMA_CFG.WNR must be configured for memory write operation and DMA_CFG.PSIZE must be configured to a value no larger than the supported bus width of the peripheral DMA bus. This must also match the value written for the source DMA channel configuration.

    - Interrupts and triggers may also be configured at this step depending on requirements.

    STEP RESULT: The memory write DMA transfer begins.

RESULT:

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 synchronization of DMA work unit completion with the software. This can best be achieved using DMA channel interrupt and trigger events, or through polling of these event's status bits within the DMA channel registers, or a combination of both. Polling for address or count can only provide synchronization within loose tolerances comparable to pipeline lengths.

Interrupt and Trigger Event Based Synchronization

Interrupt and trigger based synchronization methods must avoid interrupt/trigger overrun, or the failure to invoke a DMA channel's event handler for every event due to excessive latency in processing of events. Generally, the system design must either ensure that only one event per channel is scheduled (for example, at the end of a descriptor list), or that generated events are spaced sufficiently far apart in time that system processing budgets can guarantee every event is serviced.

When the DMA channel issues an interrupt/trigger event or changes event status bits in the DMA_STATregister, it guarantees that the last memory operation of the work unit is complete. For memory read DMA transactions, this means that the final memory read data has been safely received in the DMA channel's FIFO. For memory write DMA transactions, this means that the DMA channel has received an acknowledge that the last write transfer of the work unit is complete.

Register Polling Based Synchronization

Polling of DMA channel registers such as the DMA_ADDR_CUR, DMA_DSCPTR_CUR, or the DMA_XCNT_CUR/DMA_YCNT_CUR registers is not recommended as a method of precisely synchronizing DMA with data processing due to the DMA channel FIFOs and DMA/memory pipelining. The current address, pointer, and count registers change several cycles in advance of the completion of the corresponding memory operation, as measured by 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 a DMA channel write operation is initiated by DMA channel A. This causes the DDR SDRAM to perform a page open operation which takes many system clock cycles. Meanwhile, another DMA channel (channel B) which does not in itself incur latency, initiates a transfer that is stalled behind the slow operation of channel A. Software monitoring channel B could not safely conclude whether the memory location pointed to by channel B's DMA_ADDR_CUR register has or has not been written, based solely on this register's contents.

Polling of the current address, pointer, and count registers can permit loose synchronization of DMA with software if allowances are made for the lengths of the DMA/memory pipeline. Further, the length of the DMA FIFO for a particular peripheral needs to be taken into consideration. The DMA channel does not advance current address/pointer/count registers if these FIFOs are filled with incomplete work (including reads that have been started but not yet finished).

Additionally, the length of the pipelines to the destination memory needs to be taken into consideration. If the DMA FIFO length and the DMA channel's memory pipeline length are added, an estimate can be made of the maximum number of incomplete memory operations in progress at one time.

NOTE: The estimate would be a maximum, as the DMA/memory pipeline may include traffic from other DMA channels.

Descriptor Queues

A system designer might 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 might contain. The software could manage these transfers using a circular linked list

of DMA descriptors, where each descriptor sets the DMA_DSCPTR_NXT descriptor which points to the next descriptor set, and the last descriptor set points to the first.

The code that writes into this descriptor list could use the processor's circular addressing modes, so that it 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 could even be written once at startup, and skipped over as each descriptor's new contents are written.

The recommended method for synchronization of a descriptor queue is through the use of an interrupt or trigger. The descriptor queue is structured so 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 or trigger on every descriptor set. This method should only be used 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, while 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 if the DMA channel is running or not. If the DMA channel is paused (counts equal), the software increments its count and then starts the DMA channel by writing the new descriptor set's DMA_CFG descriptor.

If the counts are unequal, the software instead modifies the next-to-last descriptor set's DMA_CFG descriptor so that it now describes the newly-queued descriptor set. This operation does not disrupt the DMA channel provided the rest of the descriptor set's descriptors are initialized in advance. It is necessary, however, to synchronize the software to the DMA to correctly determine whether the new or the old DMA_CFG value was read by the DMA channel.

The synchronization operation should be performed in the event handler. First, when an event is detected, the handler should read the channel's DMA_STAT register. If the DMA_STAT.RUN bit indicates the DMA channel is running, then the channel has moved on to processing another descriptor, and the event handler may increment its count and exit. If the DMA_STAT.RUN bit indicates the channel is not running, then the channel is paused, either because there are no more descriptor sets to process, or because the last descriptor set was queued too late (that is, the modification of the next-to-last descriptor set's DMA_CFG descriptor occurred after that descriptor was read into the DMA channel). In this case, the event handler writes the

DMA_CFG value appropriate for the last descriptor set to the DMA channel's DMA_CFG register, increments the completed descriptor count, and exits.

Again, this system can fail if the system's event latencies are large enough to cause any of the channel's DMA events to be dropped. An event handler capable of safely synchronizing multiple descriptor set interrupts 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 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 a new descriptor set's contents and adds it to the waiting portion of the queue. The descriptor set's DMA_CFG descriptor should 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, forming a waiting portion of the queue that is disconnected from the active portion of the queue being processed by the DMA channel. 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 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 containing the desired DMA_CFG value to use to start the first waiting descriptor set in the waiting queue (or 0 to indicate no descriptors are waiting).

It is critical that the software 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 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 channel's DMA_CFG register. This single register write 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.

If, on the other hand, the event handler reads its mailbox and finds a DMA_CFG value of zero, indicating there is no more work to perform, then it 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 needs to synchronize the activation of a new work unit with the interrupt handler. If the queue has stopped (that is, if the mailbox from the event handler is zero), the non-interrupt software is responsible for starting the queue (writing the first descriptor sets DMA_CFG value to the channel's DMA_CFG register). If the queue is not stopped, the non-interrupt software must not write the DMA_CFG register (which would cause a DMA error), but instead it should queue the descriptor onto the waiting queue and update its mailbox directed to the event handler.

ADSP-BF60x DMA Register Descriptions

DMA Channel (DMA) contains the following registers.

Table 13-17: ADSP-BF60x DMA Register List

Name	Description
DMA_DSCPTR_NXT	Pointer to Next Initial Descriptor
DMA_ADDRSTART	Start Address of Current Buffer
DMA_CFG	Configuration Register
DMA_XCNT	Inner Loop Count Start Value
DMA_XMOD	Inner Loop Address Increment
DMA_YCNT	Outer Loop Count Start Value (2D only)
DMA_YMOD	Outer Loop Address Increment (2D only)
DMA_DSCPTR_CUR	Current Descriptor Pointer
DMA_DSCPTR_PRV	Previous Initial Descriptor Pointer
DMA_ADDR_CUR	Current Address
DMA_STAT	Status Register
DMA_XCNT_CUR	Current Count(1D) or intra-row XCNT (2D)
DMA_YCNT_CUR	Current Row Count (2D only)
DMA_BWLCNT	Bandwidth Limit Count
DMA_BWLCNT_CUR	Bandwidth Limit Count Current

Table 13-17: ADSP-BF60x DMA Register List (Continued)

Name	Description
DMA_BWMCNT	Bandwidth Monitor Count
DMA_BWMCNT_CUR	Bandwidth Monitor Count Current

Pointer to Next Initial Descriptor

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.

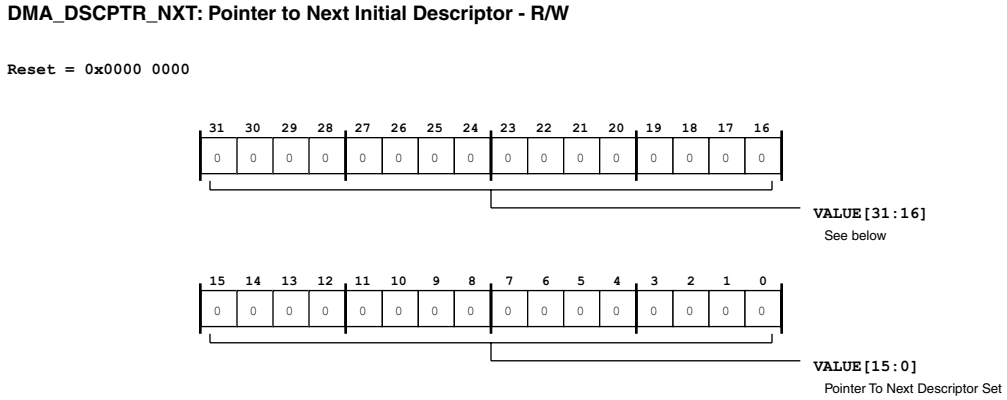


Figure 13-4: DMA_DSCPTR_NXT Register Diagram

Table 13-18: DMA_DSCPTR_NXT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Pointer To Next Descriptor Set.

Start Address of Current Buffer

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.

DMA_ADDRSTART: Start Address of Current Buffer - R/W

Reset = 0x0000 0000

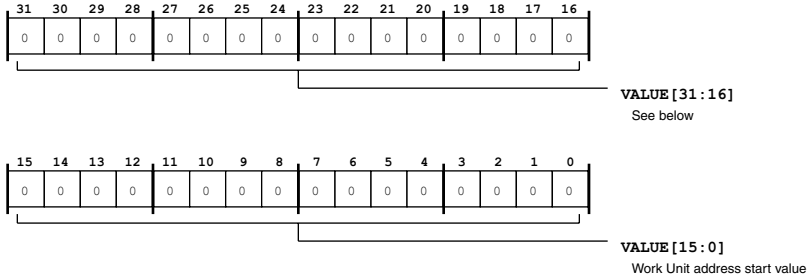


Figure 13-5: DMA_ADDRSTART Register Diagram

Table 13-19: DMA_ADDRSTART Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit address start value.

Configuration Register

The DMA_CFG sets up DMA parameters and operation modes. Other than clearing the DMA_CFG.EN bit, writing to the DMA_CFG register while a DMA process is already running cause a DMA error.

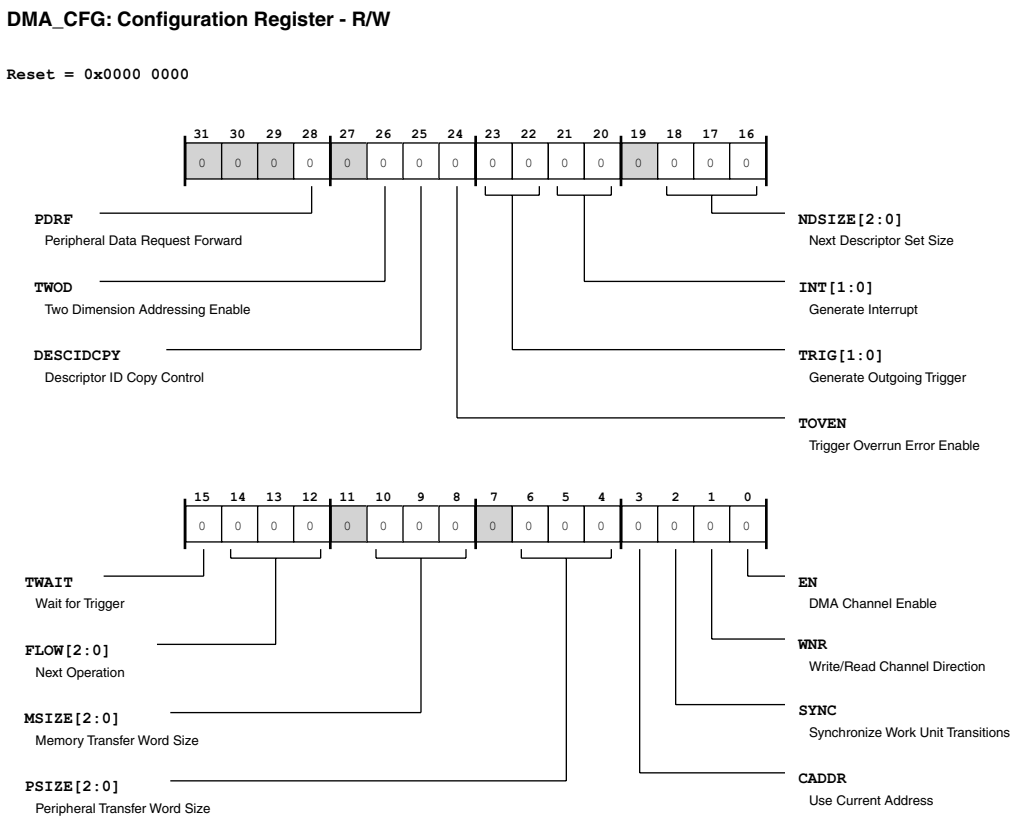


Figure 13-6: DMA_CFG Register Diagram

Table 13-20: DMA_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
28 (R/W)	PDRF	Peripheral Data Request Forward. The DMA_CFG.PDRF 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. Note that the peripheral data request forward selection applies only to DMA_CFG.FLOW bits set for stop and DMA_CFG.WNR bits set for memory read.	
		0	Peripheral Data Request Not Forwarded
		1	Peripheral Data Request Forwarded

Table 13-20: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/W)	TWOD	Two Dimension Addressing Enable. The DMA_CFG.TWOD 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 specifies when to copy the initial descriptor pointer to the DMA_DSCPTR_PRV register. Note that a bus write to DMA_CFG to clear the DMA_CFG.EN bit cause the DMA to use the new value of DMA_CFG.DESCIDCPY immediately. To preserve consistency (if required by application), the new value of DMA_CFG.DESCIDCPY should match 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 13-20: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
23:22 (R/W)	TRIG	Generate Outgoing Trigger. 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 DMA_CFG.TRIG to trigger when DMA_YCNT_CUR reaches 0 (or to reserved) cause the DMA to flag a configuration error. If in two-dimensional addressing mode, the options are to trigger at the end of each row of the inner loop (when DMA_XCNT_CUR reaches 0), to trigger only after completing the whole work unit (when DMA_YCNT_CURreaches 0), or not to trigger at all. 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. 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. For more information about trigger generation timing, see the trigger section of the DMA functional description.	
		0	Never assert Trigger
		1	Trigger when XCNTCUR reaches 0
		2	Trigger when YCNTCUR reaches 0
		3	Reserved

Table 13-20: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	INT	Generate Interrupt. The DMA_CFG.INT selects whether an interrupt is sent to the core based on work unit status or peripheral interrupt request. For one-dimensional mode, setting DMA_CFG.INT for interrupt when DMA_YCNT_CUR reaches 0 causes the DMA to flag a configuration error. The peripheral interrupt setting enables the DMA to generate the last grant indication and to accept/forward the peripheral interrupt command. Note that the peripheral interrupt selection applies only to DMA_CFG.FLOW bits set for stop and DMA_CFG.WNR bits set for memory read. If DMA_CFG.INT is set for interrupt on count completion (DMA_XCNT_CUR or DMA_YCNT_CUR reach 0) and the peripheral issues a finish command, the DMA issues an interrupt 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
18:16 (R/W)	NDSIZE	Next Descriptor Set Size. The DMA_CFG.NDSIZE 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 may or may not have the next descriptor pointer, depending on whether it is a descriptor list or 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 13-20: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	TWAIT	Wait for Trigger. The DMA_CFG.TWAIT controls whether the DMA waits for a incoming trigger from another channel or user. If DMA_CFG.TWAIT is set, the DMA enters the wait state before starting the next work unit, including descriptor fetch if in descriptor mode. Using the wait for trigger control is not allowed for descriptor-on-demand mode, and using this control in that mode 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)

Table 13-20: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14:12 (R/W)	FLOW	Next Operation. The DMA_CFG.FLOW selects descriptor handling options.	
		0	STOP - Stop When the current work unit completes, the DMA channel stops automatically, after signaling an interrupt (if selected). The DMA_STAT.RUN status bit changes to idle, while DMA_CFG.EN bit is unchanged. In this state, the channel is stopped. Peripheral interrupts are still filtered out by the DMA. The channel may be restarted simply by another write (with the DMA_CFG.EN set) to the DMA_CFG register specifying the next work unit.
		1	AUTO - Autobuffer In this mode, no descriptors in memory are used. Instead, DMA is performed in a continuous circular buffer fashion based on user programmed DMA MMR settings. On completion of the work unit, the parameter registers are reloaded into the current registers, and DMA resumes immediately with zero overhead. This mode is considered to be a succession of automatically restarted work units. Autobuffer mode is stopped by a user write of 0 to the DMA_CFG.EN bit.
		2	Reserved
		3	Reserved
		4	DSCL - Descriptor List This mode fetches a descriptor Set from memory that includes DMA_DSCPTR_NXT, allowing maximum flexibility in locating descriptors in memory.

Table 13-20: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		5	DSCA - Descriptor Array This mode fetches a descriptor set from memory that does not include the DMA_DSCPTR_NXT element. Because the descriptor set does not contain a next descriptor pointer entry, the DMA defaults to using the DMA_DSCPTR_CUR register to step through descriptors, allowing a group of descriptors sets to follow one another in memory as an array.
		6	Descriptor On Demand List This mode fetches a descriptor set from memory that includes DMA_DSCPTR_NXT. At the end of the work unit, if the channel has not been triggered, the work unit is repeated. But, if the channel has been triggered before the end of the work unit, the DMA fetches a new descriptor set.
		7	Descriptor On Demand Array This mode fetches a descriptor set from memory that does not include DMA_DSCPTR_NXT. At the end of the work unit, if the channel has not been triggered, the work unit is repeated. But, if the channel has been triggered before the end of the work unit, the DMA fetches a new descriptor set is fetched. Because the descriptor set does not contain a next descriptor pointer entry, the DMA defaults to using the DMA_DSCPTR_CUR register to step through descriptors, allowing a group of descriptors sets to follow one another in memory as an array.

Table 13-20: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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. Note that 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.	
		0	1 Byte
		1	2 Bytes
		2	4 Bytes
		3	8 Bytes
		4	16 Bytes
		5	32 Bytes
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/grant results in a single peripheral access. There is no bursting 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. Note that the processor's peripheral bus is dedicated to DMA and peripheral accesses.	
		0	1 Byte
		1	2 Bytes
		2	4 Bytes
		3	8 Bytes
3 (R/W)	CADDR	Use Current Address. If the DMA_CFG.CADDR bit is cleared, the DMA loads the DMA_ADDRSTART register on the first access of the work unit. If 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. If this mode is used at the end of a descriptor list or array, the DMA ignores the start address value that is fetched as part of the descriptor set.	
		0	Load Starting Address
		1	Use Current Address

Table 13-20: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	SYNC	Synchronize Work Unit Transitions. Setting the DMA_CFG.SYNC bit clears the DMA FIFO and pointers before starting the first Work Unit of a Work Unit Chain. When the transfer direction is memory read/transmit (DMA_CFG.WNR =0), the DMA waits until all data has been transmitted to peripheral before moving on to next Work Unit, clearing the FIFO and pointers. When the transfer direction is memory write/receive (DMA_CFG.WNR =1), the DMA ignores the DMA_CFG.SYNC bit value after processing the first Work Unit of a Work Unit Chain. Because the channel is allowed to receive data when turned on but idling, there could be data in the FIFO that was put in by the peripheral before the channel was programmed. With DMA_CFG.SYNC set at the beginning of a work unit chain (during the first work unit), the DMA clears the FIFO, erasing the data put in to the FIFO while the channel was idling. Syncing lets you change the DMA_CFG.PSIZE between individual work units and (in some cases) work unit chains. The sync resets the pointers in the FIFO, preventing misaligned FIFO access. The DMA_CFG.MSIZE may be changed between consecutive work units, independent of the DMA_CFG.SYNC bit setting. Syncing also permits changes to transfer direction. And, 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.	
		0	No Synchronization
		1	Synchronize Channel
1 (R/W)	WNR	Write/Read Channel Direction. The DMA_CFG.WNR selects receive (write to memory) or transmit (read from memory) channel direction.	
		0	Transmit (Read from memory)
		1	Receive (Write to memory)

Table 13-20: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	DMA Channel Enable. The DMA_CFG.EN enables the selected DMA Channel. 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. To avoid unexpected results, take care to enable the DMA channel before enabling the peripheral, and to disable the peripheral before disabling the DMA channel. A transition of DMA_CFG.EN from 0 to 1 creates a hard reset of all internal counters and state, including the DMA_STAT register. (All other register values remain unaffected.) A transition from 1 to 0 maintains all counters and registers for the user to read and analyze. If a descriptor is loaded (see DMA_CFG.FLOW field) with DMA_CFG.EN cleared, the DMA goes to off/idle state after the descriptor load is complete.	
		0	Disable
		1	Enable

Inner Loop Count Start Value

For 2D DMA, the DMA_XCNT contains the inner loop count. This value selects the number of DMA_CFG.MSIZE size data transfers to make up the length of a row. For 1D DMA, DMA_XCNT 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 DMA_XCNT is 0x0 when a work unit begins.

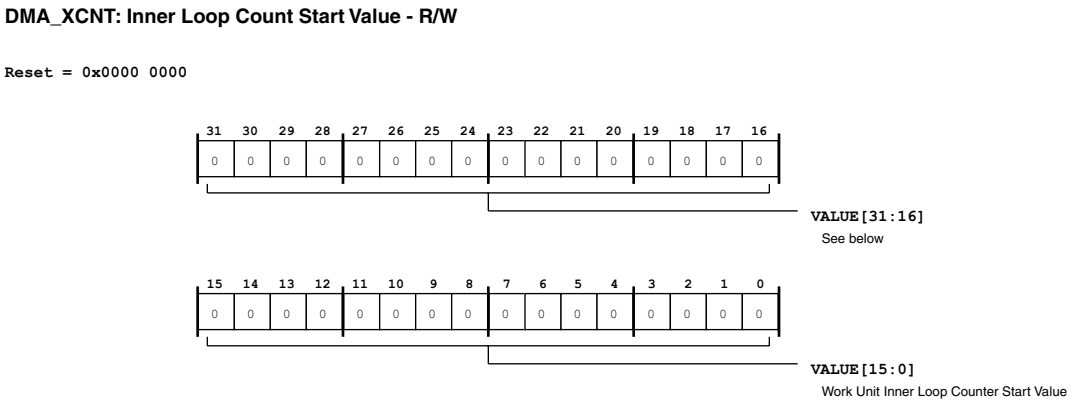


Figure 13-7: DMA_XCNT Register Diagram

Table 13-21: DMA_XCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Inner Loop Counter Start Value.

Inner Loop Address Increment

The DMA_XMOD 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 channel.

The DMA_XMOD 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 may 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.

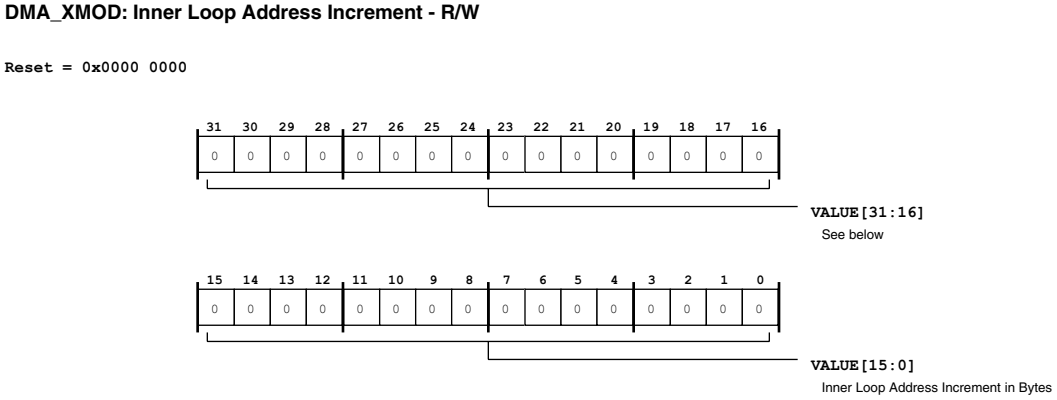


Figure 13-8: DMA_XMOD Register Diagram

Table 13-22: DMA_XMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Inner Loop Address Increment in Bytes.

Outer Loop Count Start Value (2D only)

For 2D DMA, the DMA_YCNT 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 DMA_YCNT is 0x0 when a work unit begins.

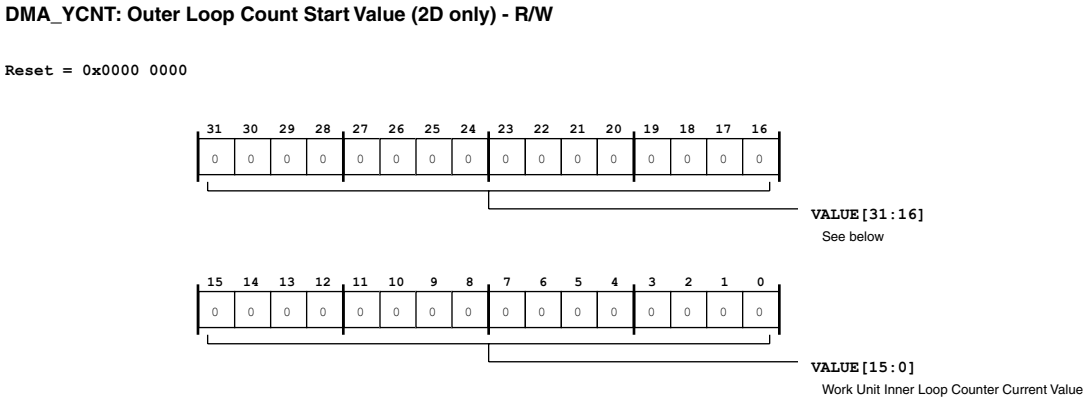


Figure 13-9: DMA_YCNT Register Diagram

Table 13-23: DMA_YCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Inner Loop Counter Current Value.

Outer Loop Address Increment (2D only)

The DMA_YMOD 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 channel.

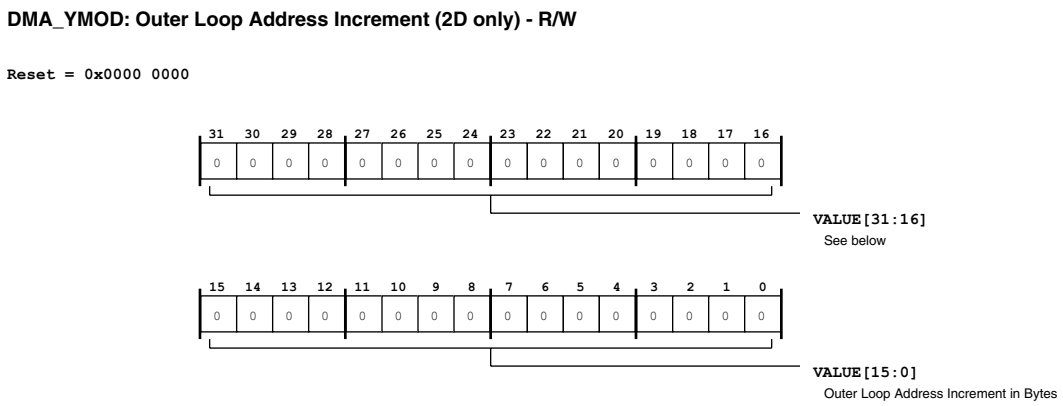


Figure 13-10: DMA_YMOD Register Diagram

Table 13-24: DMA_YMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Outer Loop Address Increment in Bytes.

Current Descriptor Pointer

The DMA_DSCPTR_CUR 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 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 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 in.

When the entire descriptor set has been read, the DMA_DSCPTR_CUR register contains this value:

DMA_DSCPTR_CUR = Descriptor Start Address + Descriptor Size (# of elements)

For descriptor array mode, the DMA_DSCPTR_CUR register, and not the DMA_DSCPTR_NXT register, must be programmed by MMR access before starting DMA operation.

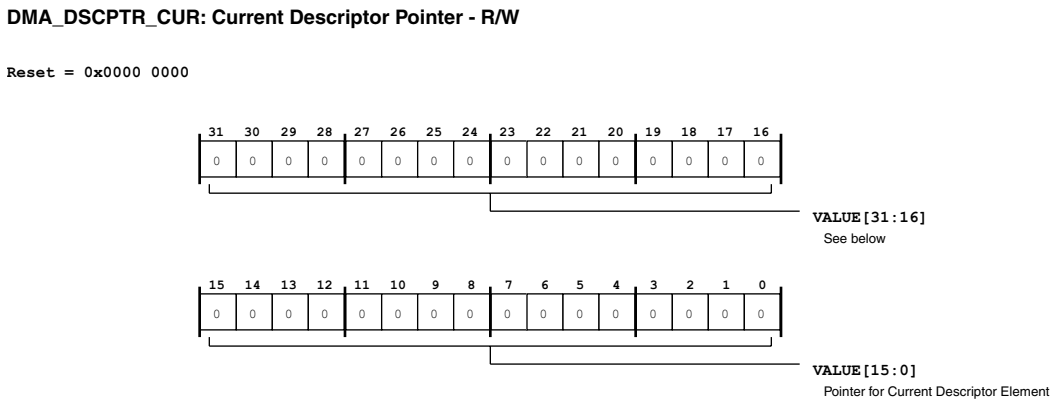


Figure 13-11: DMA_DSCPTR_CUR Register Diagram

Table 13-25: DMA_DSCPTR_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Pointer for Current Descriptor Element.

Previous Initial Descriptor Pointer

The DMA_DSCPTR_PRV 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 DMA_DSCPTR_PRV 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 would occur when used for descriptor fetches. As a result, bit 1 and 0 of DMA_DSCPTR_PRV may be used for status. For more information, see the section on descriptor pointer capture in the DMA functional description.

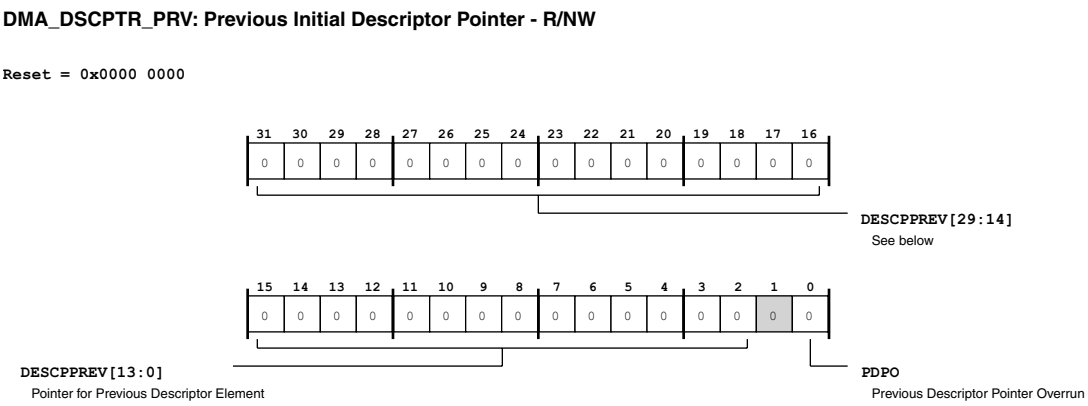


Figure 13-12: DMA_DSCPTR_PRV Register Diagram

Table 13-26: DMA_DSCPTR_PRV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:2 (R/NW)	DESCPPREV	Pointer for Previous Descriptor Element.
0 (R/NW)	PDPO	Previous Descriptor Pointer Overrun.

Current Address

The DMA_ADDR_CUR contains the present memory transfer address for a given work unit. At the start of a work unit, the DMA_ADDR_CUR is loaded from the DMA_ADDRSTART register, and DMA_ADDR_CUR 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 channel.

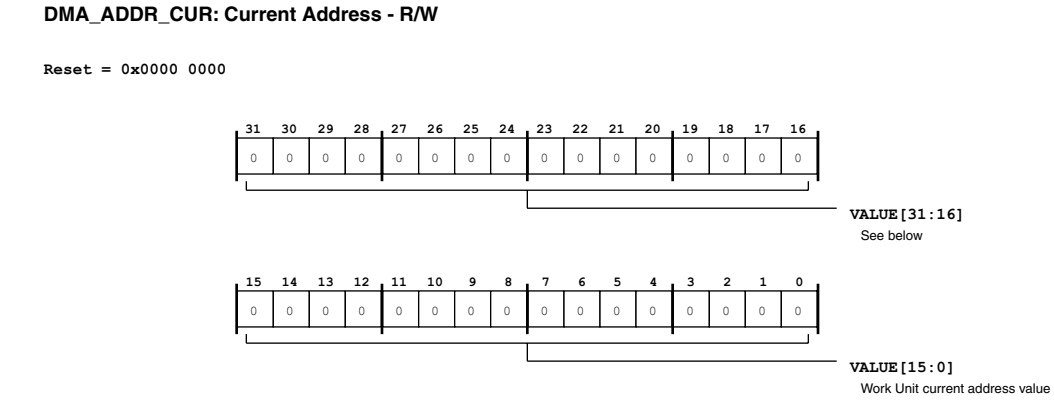


Figure 13-13: DMA_ADDR_CUR Register Diagram

Table 13-27: DMA_ADDR_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit current address value.

Status Register

The DMA_STAT indicates status of DMA work units, FIFO, errors, interrupts, and triggers.

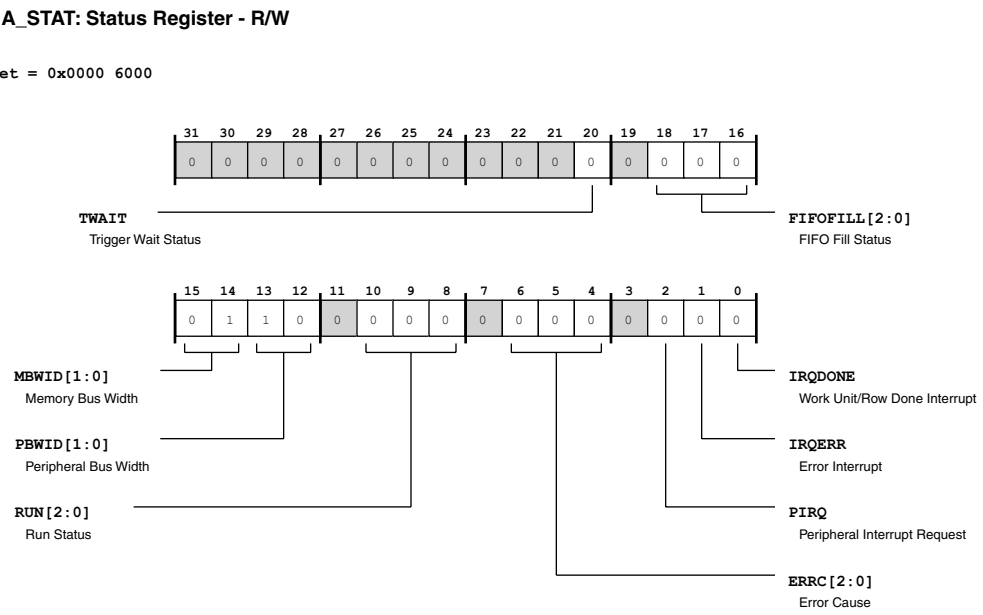


Figure 13-14: DMA_STAT Register Diagram

Table 13-28: DMA_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/NW)	TWAIT	Trigger Wait Status. The DMA_STAT.TWAIT 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 is processed. The DMA does not distinguish between one or more triggers received.	
		0	No trigger received
		1	Trigger received

Table 13-28: DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/NW)	FIFOFILL	FIFO Fill Status. The DMA_STAT.FIFOFILL 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
		6	Reserved
		7	Full
15:14 (R/NW)	MBWID	Memory Bus Width. The DMA_STAT.MBWID 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 indicates the width of the peripheral bus connected to this DMA.	
		0	1 Byte
		1	2 Bytes
		2	4 Bytes
		3	8 Bytes

Table 13-28: DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10:8 (R/NW)	RUN	Run Status. The DMA_STAT.RUN reports the DMA's current operational state. If the DMA is in idle or stop state, the DMA_CFG.EN bit may be either 0 or 1. Note that the DMA_STAT.RUN is not cleared by a transition of the DMA_CFG.EN bit from 0 to 1. The DMA_STAT.RUN is automatically cleared 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 DMA_STAT.ERRC to identify the type of error. For more information, see the errors section of the DMA functional description.	
		0	Configuration Error
		1	Illegal Write Occurred While Channel Running
		2	Address Alignment Error
		3	Memory Access/Fabric Error
		4	Reserved
		5	Trigger Overrun
		6	Bandwidth Monitor Error
		7	Reserved

Table 13-28: DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	PIRQ	Peripheral Interrupt Request. The DMA_STAT.PIRQ indicates an interrupt has been caused by the peripheral. Programmers can use the DMA_STAT.PIRQ status to help determine which DMA asserted the interrupt and to help distinguish between an interrupt caused based on the state of the work unit and an interrupt made by the peripheral.	
		0	No Interrupt
		1	Interrupt Signaled by Peripheral
1 (R/W1C)	IRQERR	Error Interrupt. The DMA_STAT.IRQERR indicates that the DMA has detected a documented rule violations during DMA programming or operation. The DMA cannot, however, flag all possible programming or operation issues to indicate errors. Programmers can use DMA_STAT.IRQERR to help determine which DMA issued the error interrupt. Note that the DMA_STAT.IRQERR is not cleared by a transition of the DMA_CFG.EN bit from 0 to 1. The DMA_STAT.IRQERR must be cleared with a write-1-to-clear operation prior to the DMA_CFG.EN transition for the fields to be reset.	
		0	No Error
		1	Error Occurred
0 (R/W1C)	IRQDONE	Work Unit/Row Done Interrupt. The DMA_STAT.IRQDONE indicates the DMA has detected the completion of a work unit or row (inner loop count) and has issued an interrupt. Programmers can use the DMA_STAT.IRQDONE status to help determine which DMA asserted the interrupt and to help distinguish between an interrupt caused based on the state of the work unit and an interrupt made by the peripheral. For more information, see the interrupts section of the DMA functional description.	
		0	Inactive
		1	Active

Current Count(1D) or intra-row XCNT (2D)

For 1D DMA, the DMA loads the DMA_XCNT_CUR from the DMA_XCNT register at the beginning of each work unit. For 2D DMA, the DMA loads DMA_XCNT_CUR from the DMA_XCNT register after the end of each row. The DMA decrements the value in DMA_XCNT_CUR each time a DMA_CFG.MSIZE size data transfer occurs.

When the count in DMA_XCNT_CUR expires, the work unit is complete. In 2D DMA, the DMA_XCNT_CUR value is 0 only when the entire transfer is complete.

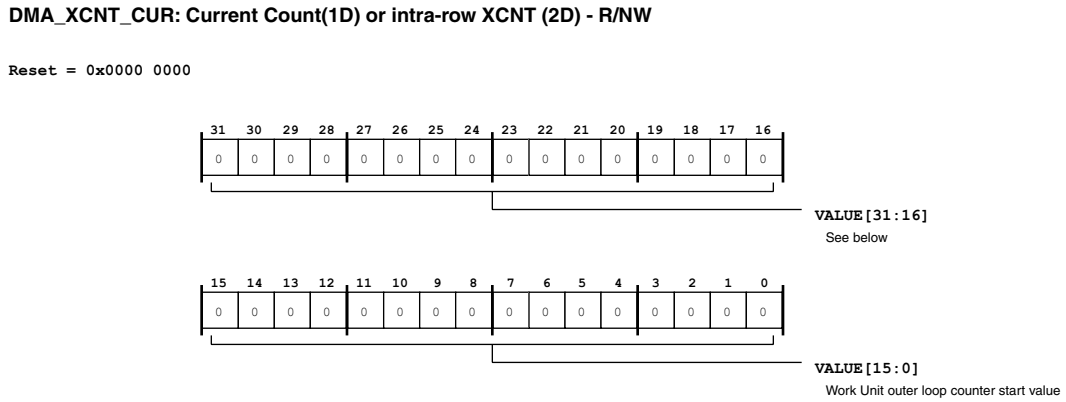


Figure 13-15: DMA_XCNT_CUR Register Diagram

Table 13-29: DMA_XCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Work Unit outer loop counter start value.

Current Row Count (2D only)

For 2D DMA, the DMA loads the DMA_YCNT_CUR from the DMA_YCNT register at the beginning of each 2D DMA session. The DMA_YCNT_CUR is not used for 1D DMA. The DMA decrements DMA_YCNT_CUR each time the DMA_XCNT_CUR expires during 2D DMA operation, signifying completion of an entire row transfer.

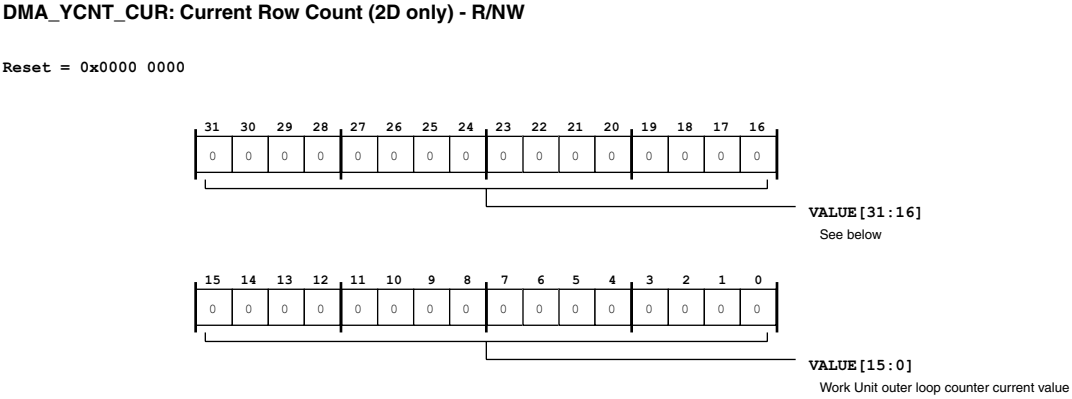


Figure 13-16: DMA_YCNT_CUR Register Diagram

Table 13-30: DMA_YCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Work Unit outer loop counter current value.

Bandwidth Limit Count

The DMA_BWLCNT contains a count that determines how often the DMA issues memory transactions. The DMA loads the value from DMA_BWLCNT into DMA_BWLCNT_CUR 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.

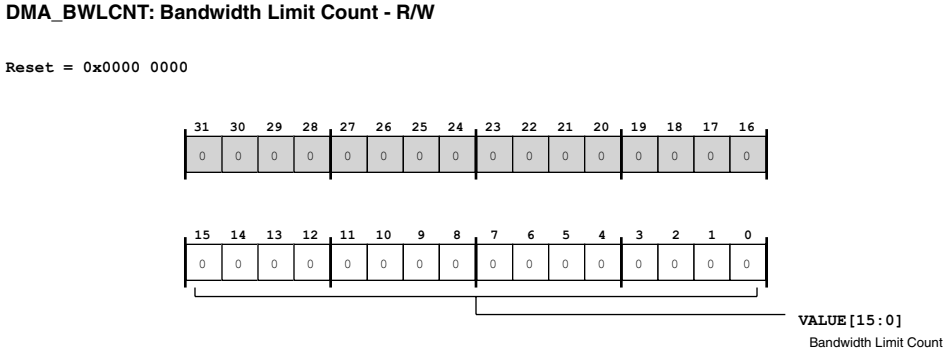


Figure 13-17: DMA_BWLCNT Register Diagram

Table 13-31: DMA_BWLCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Bandwidth Limit Count.

Bandwidth Limit Count Current

The DMA_BWLCNT_CUR contains the number of SCLK count cycles remaining before the next request is issued.

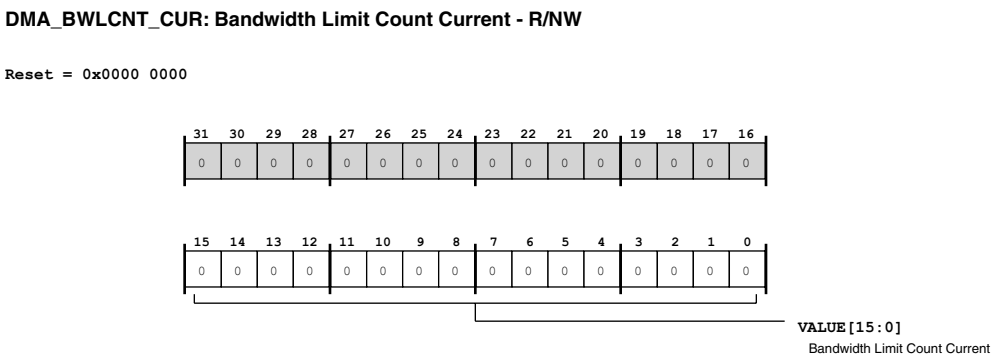


Figure 13-18: DMA_BWLCNT_CUR Register Diagram

Table 13-32: DMA_BWLCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Bandwidth Limit Count Current.

Bandwidth Monitor Count

The DMA_BWMCNT 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 DMA_BWMCNT into DMA_BWMCNT_CUR. The DMA decrements DMA_BWMCNT_CUR every SCLK a work unit is active. If DMA_BWMCNT_CUR reaches 0x0000_0000, the DMA_STAT.IRQERR bit is set, and the DMA_STAT.ERRC is set to 0x6. The DMA_BWMCNT_CUR remains at 0x0000_0000 until it is reloaded when the work unit completes. Unlike other error causes, a bandwidth monitor error does not stop work unit processing. Programming 0x0000_0000 disables bandwidth monitor functionality.

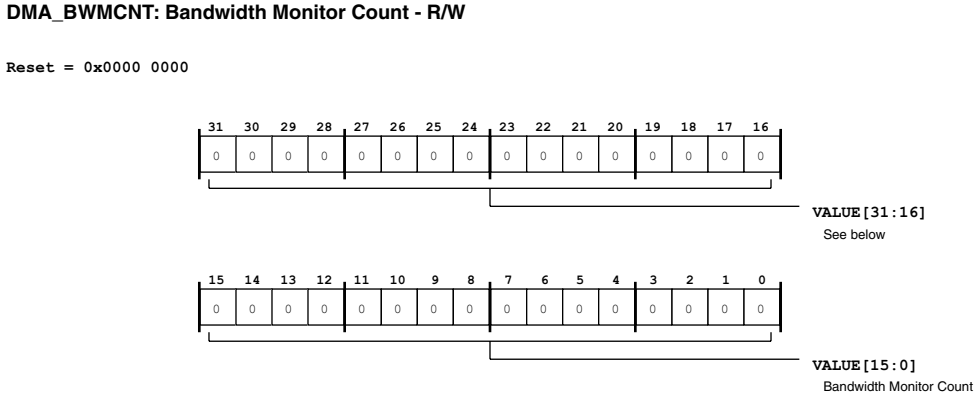


Figure 13-19: DMA_BWMCNT Register Diagram

Table 13-33: DMA_BWMCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Bandwidth Monitor Count.

Bandwidth Monitor Count Current

The DMA_BWMCNT_CUR contains the number of cycles remaining for the current descriptor to complete.

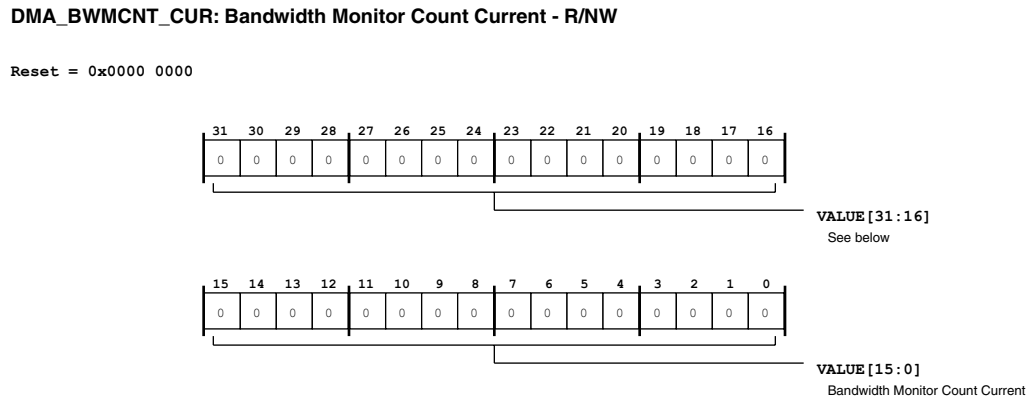


Figure 13-20: DMA_BWMCNT_CUR Register Diagram

Table 13-34: DMA_BWMCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Bandwidth Monitor Count Current.

DMA Channel List for ADSP-BF60x

The following tables provide DMA channel assignment and channel parametric information for the ADSP-BF60x processors.

Table 13-35: SPORT DMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/ Monitor Support	Memory Bus Width (DMA_STAT.MBWID)	Peripheral Bus Width (DMA_STAT.PBWID)	Max Outstanding Reads	Max Outstanding Writes
DMA0	SPORT0 Channel A	64	No	32-bit	32-bit	4	4
DMA1	SPORT0 Channel B	64	No	32-bit	32-bit	4	4
DMA2	SPORT1 Channel A	64	No	32-bit	32-bit	4	4
DMA3	SPORT1 Channel B	64	No	32-bit	32-bit	4	4
DMA4	SPORT2 Channel A	64	No	32-bit	32-bit	4	4
DMA5	SPORT2 Channel B	64	No	32-bit	32-bit	4	4

Table 13-36: SPI DMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/ Monitor Support	Memory Bus Width (DMA_STAT.MSIZE)	Peripheral Bus Width (DMA_STAT.PSIZE)	Max Outstanding Reads	Max Outstanding Writes
DMA6	SPI0 Transmit	64	No	32 bit	32-bit	4	4
DMA7	SPI0 Receive	64	No	32 bit	32-bit	4	4
DMA8	SPI1 Transmit	64	No	32 bit	32-bit	4	4
DMA9	SPI1 Receive	64	No	32 bit	32-bit	4	4

Table 13-37: RSI DMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/ Monitor Support	Memory Bus Width (DMA_STAT.MSIZE)	Peripheral Bus Width (DMA_STAT.PSIZE)	Max Outstanding Reads	Max Outstanding Writes
DMA10	RSI0	64	No	32 bit	32-bit	4	4

Table 13-38: SDU DMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/ Monitor Support	Memory Bus Width (DMA_STAT.MSIZE)	Peripheral Bus Width (DMA_STAT.PSIZE)	Max Outstanding Reads	Max Outstanding Writes
DMA11	SDU0	64	No	32 bit	32-bit	4	4

Table 13-39: Link Port DMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/ Monitor Support	Memory Bus Width (DMA_STAT.MSIZE)	Peripheral Bus Width (DMA_STAT.PSIZE)	Max Outstanding Reads	Max Outstanding Writes
DMA13	Linkport0	64	No	32 bit	32-bit	4	4
DMA14	Linkport1	64	No	32 bit	32-bit	4	4
DMA15	Linkport2	64	No	32 bit	32-bit	4	4
DMA16	Linkport3	64	No	32 bit	32-bit	4	4

Table 13-40: UART DMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/ Monitor Support	Memory Bus Width (DMA_STAT.MSIZE)	Peripheral Bus Width (DMA_STAT.PSIZE)	Max Outstanding Reads	Max Outstanding Writes
DMA17	UART0 Transmit	64	No	32-bit	32-bit	4	4

Table 13-40: UART DMA Channels (Continued)

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/ Monitor Support	Memory Bus Width (DMA_STAT.MSIZE)	Peripheral Bus Width (DMA_STAT.PSIZE)	Max Outstanding Reads	Max Outstanding Writes
DMA18	UART0 Receive	64	No	32-bit	32-bit	4	4
DMA19	UART1 Transmit	64	No	32-bit	32-bit	4	4
DMA20	UART1 Receive	64	No	32-bit	32-bit	4	4

Table 13-41: MDMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/ Monitor Support	Memory Bus Width (DMA_STAT.MSIZE)	Peripheral Bus Width (DMA_STAT.PSIZE)	Max Outstanding Reads	Max Outstanding Writes
DMA21	MDMA0 Source/ Destination	128	Yes	32-bit	32-bit	8	7
DMA22	MDMA0 Destination/ Source	64	Yes	32-bit	32-bit	8	4
DMA23	MDMA1 Source/ Destination	64	Yes	32-bit	32-bit	8	4
DMA24	MDMA1 Destination/ Source	64	Yes	32-bit	32-bit	8	4
DMA25	MDMA2 Source/ Destination	128	Yes	32-bit	32-bit	8	7
DMA26	MDMA2 Destination/ Source	64	Yes	32-bit	32-bit	8	4
DMA27	MDMA3 Source/ Destination	64	Yes	32-bit	32-bit	8	4

Table 13-41: MDMA Channels (Continued)

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/ Monitor Support	Memory Bus Width (DMA_STAT.MSIZE)	Peripheral Bus Width (DMA_STAT.PSIZE)	Max Outstanding Reads	Max Outstanding Writes
DMA28	MDMA3 Destination/ Source	64	Yes	32-bit	32-bit	8	4

Table 13-42: CRC DMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/ Monitor Support	Memory Bus Width (DMA_STAT.MSIZE)	Peripheral Bus Width (DMA_STAT.PSIZE)	Max Outstanding Reads	Max Outstanding Writes
DMA21	CRC0 Receive	128	Yes	32-bit	32-bit	8	7
DMA22	CRC0 Transmit	64	Yes	32-bit	32-bit	8	4
DMA23	CRC1 Receive	64	Yes	32-bit	32-bit	8	4
DMA24	CRC1 Transmit	64	Yes	32-bit	32-bit	8	4

Table 13-43: EPPI DMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/ Monitor Support	Memory Bus Width (DMA_STAT.MSIZE)	Peripheral Bus Width (DMA_STAT.PSIZE)	Max Outstanding Reads	Max Outstanding Writes
DMA29	EPPI0 Luma/ Pixel Pipe Data	128	No	32-bit	32-bit	8	7
DMA30	EPPI0 Chrominance Data	128	No	32-bit	32-bit	8	7
DMA31	EPPI2 Luma/ Pixel Pipe Data	128	No	32-bit	32-bit	8	7

Table 13-43: EPPI DMA Channels (Continued)

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/ Monitor Support	Memory Bus Width (DMA_STAT.MSIZE)	Peripheral Bus Width (DMA_STAT.PSIZE)	Max Outstanding Reads	Max Outstanding Writes
DMA32	EPPI2 Chrominance Data	128	No	32-bit	32-bit	8	7
DMA33	EPPI1 Luma/ Pixel Pipe Data	128	No	32-bit	32-bit	8	7
DMA34	EPPI1 Chrominance Data	128	No	32-bit	32-bit	8	7

Table 13-44: PIXC DMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/ Monitor Support	Memory Bus Width (DMA_STAT.MSIZE)	Peripheral Bus Width (DMA_STAT.PSIZE)	Max Outstanding Reads	Max Outstanding Writes
DMA35	PIXC Input Channel A	128	Yes	32-bit	32-bit	8	7
DMA36	PIXC Input Channel B	128	Yes	32-bit	32-bit	8	7
DMA37	PIXC Output	128	Yes	32-bit	32-bit	8	7

Table 13-45: PVP DMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/ Monitor Support	Memory Bus Width (DMA_STAT.MSIZE)	Peripheral Bus Width (DMA_STAT.PSIZE)	Max Outstanding Reads	Max Outstanding Writes
DMA38	PVP0 Camera Pipe Data Output Port 1	128	No	32-bit	32-bit	8	7

Table 13-45: PVP DMA Channels (Continued)

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/ Monitor Support	Memory Bus Width (DMA_STAT.MSIZE)	Peripheral Bus Width (DMA_STAT.PSIZE)	Max Outstanding Reads	Max Outstanding Writes
DMA39	PVP0 Camera Pipe Data Output Port 2	128	No	32-bit	32-bit	8	7
DMA40	PVP0 Camera Pipe Status Output	64	No	32-bit	32-bit	8	7
DMA41	PVP0 Camera Pipe Configuration Input	64	Yes	32-bit	32-bit	8	7
DMA42	PVP0 Memory Pipe Data Output	128	Yes	32-bit	32-bit	8	7
DMA43	PVP0 Memory Pipe Data Input	128	Yes	32-bit	32-bit	8	7
DMA44	PVP0 Memory Pipe Status Output	64	No	32-bit	32-bit	8	7
DMA45	PVP0 Memory Pipe Configuration Input	64	Yes	32-bit	32-bit	8	7
DMA46	PVP0 Camera Pipe Data Output Port 0	128	No	32-bit	32-bit	8	7

14 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 interrupts

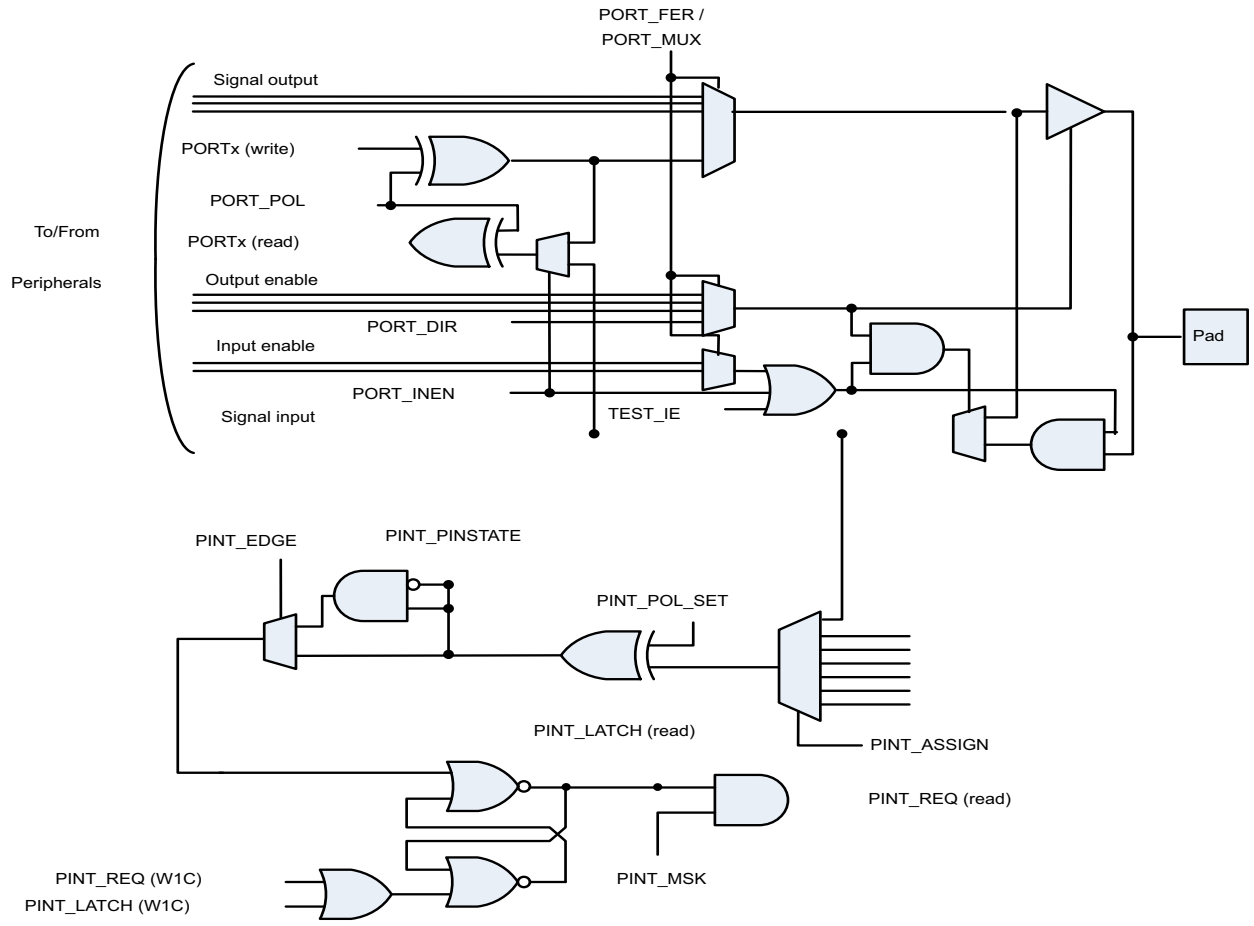


Figure 14-1: Simplified GPIO and Pin Interrupt Signal Flow

PORT Features

The PORTs include the following features:

- Up to 112 general-purpose I/O (GPIO) pins
- Input mode, output mode, and open-drain mode of GPIO operation
- Port multiplexing controlled by individual pin-per-pin base
- No glue hardware required for unused pins
- Six interrupt channels dedicated to pin interrupts
- All port pins provide interrupt functionality
- Byte-wide pin-to-interrupt assignment

PORT Functional Description

Every port pin can operate in GPIO mode. This is the default after reset and is controlled by the port-specific `PORTx_FER` enable register. Every port has a dedicated set of MMR registers that control the GPIO functionality. Every bit in these registers represents a certain GPIO pin of the specific port. The following sections provide functional descriptions for PORT features:

- *ADSP-BF60x PORT Register List*
- *ADSP-BF60x PINT Register List*
- *ADSP-BF60x PINT Interrupt List*
- *ADSP-BF60x PINT Trigger List*
- *ADSP-BF60x PADS Register List*
- *PORT Definitions*
- *PORT Architectural Concepts*

ADSP-BF60x PORT Register List

Every port pin can operate in general-purpose I/O (GPIO) mode. This 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 14-1: ADSP-BF60x PORT Register List

Name	Description
PORT_FER	Port x Function Enable Register
PORT_FER_SET	Port x Function Enable Set Register
PORT_FER_CLR	Port x Function Enable Clear Register
PORT_DATA	Port x GPIO Data Register
PORT_DATA_SET	Port x GPIO Data Set Register
PORT_DATA_CLR	Port x GPIO Data Clear Register
PORT_DIR	Port x GPIO Direction Register
PORT_DIR_SET	Port x GPIO Direction Set Register
PORT_DIR_CLR	Port x GPIO Direction Clear Register
PORT_INEN	Port x GPIO Input Enable Register
PORT_INEN_SET	Port x GPIO Input Enable Set Register
PORT_INEN_CLR	Port x GPIO Input Enable Clear Register
PORT_MUX	Port x Multiplexer Control Register
PORT_DATA_TGL	Port x GPIO Input Enable Toggle Register
PORT_POL	Port x GPIO Polarity Invert Register
PORT_POL_SET	Port x GPIO Polarity Invert Set Register
PORT_POL_CLR	Port x GPIO Polarity Invert Clear Register
PORT_LOCK	Port x GPIO Lock Register

ADSP-BF60x PINT Register List

The pin-interrupt assignment (PINT) module 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.

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 14-2: ADSP-BF60x PINT Register List

Name	Description
PINT_MSK_SET	Pint Mask Set Register
PINT_MSK_CLR	Pint Mask Clear Register
PINT_REQ	Pint Request Register
PINT_ASSIGN	Pint Assign Register
PINT_EDGE_SET	Pint Edge Set Register
PINT_EDGE_CLR	Pint Edge Clear Register
PINT_INV_SET	Pint Invert Set Register
PINT_INV_CLR	Pint Invert Clear Register
PINT_PINSTATE	Pint Pinstate Register
PINT_LATCH	Pint Latch Register

ADSP-BF60x PINT Interrupt List

Table 14-3: ADSP-BF60x PINT Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
PINT0 Pin Interrupt Block	21		LEVEL
PINT1 Pin Interrupt Block	22		LEVEL
PINT2 Pin Interrupt Block	23		LEVEL
PINT3 Pin Interrupt Block	24		LEVEL
PINT4 Pin Interrupt Block	25		LEVEL
PINT5 Pin Interrupt Block	26		LEVEL

ADSP-BF60x PINT Trigger List

Table 14-4: ADSP-BF60x PINT Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
PINT0 Pin Interrupt Block	10	LEVEL
PINT1 Pin Interrupt Block	11	LEVEL
PINT2 Pin Interrupt Block	12	LEVEL
PINT3 Pin Interrupt Block	13	LEVEL
PINT4 Pin Interrupt Block	14	LEVEL
PINT5 Pin Interrupt Block	15	LEVEL

Table 14-5: ADSP-BF60x PINT Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
None		

ADSP-BF60x PADS Register List

The PADS controls signal hysteresis and other system interface signal features for a number of module interfaces.

Table 14-6: ADSP-BF60x PADS Register List

Name	Description
PADS_EMAC_PTP_CLKSEL	EMAC and PTP Clock Select Register
PADS_TWI_VSEL	TWI Voltage Selection
PADS_PORTS_HYST	GPIO Pin Hysteresis Enable Register

PORT Definitions

This section provides definitions relating to the GPIO ports.

x (PORTx)

The naming convention for bits uses a lowercase "x" to represent one of the existing ports alphabetically named beginning with A,B,C,... For example, the name PORTx_REG represents any one or all of PORTA_REG, PORTB_REG, PORTC_REG, and so on. The bit name PxO represents PA0, PB0, and so on.

PORT Architectural Concepts

This section describes architectural concepts relating to the GPIO ports and signals, including the following interfaces and functionality:

- *Internal Interfaces*
- *External Interfaces*
- *GPIO Functionality*
- *Port Multiplexing Control*

Internal Interfaces

All MMR registers of the pin multiplexing, GPIO and pin interrupt control blocks can be accessed through the MMR bus. There is no DMA support. Every one of the pin interrupt modules has its own and dedicated interrupt request output signal that connects directly to the SIC controller.

External Interfaces

The pin multiplexing hardware can be seen as a layer between the on-chip peripherals and the pads of the silicon. All port groups are controlled by this unit.

GPIO Functionality

By default, every GPIO is set to input mode. The input drivers are not enabled, which avoids the need for unnecessary current sinks and the external pulling of resistors on unused or *do not care* pins.

Input Mode

The default mode of every GPIO pin after reset is input mode, but the input drivers are not enabled. To enable any GPIO input drivers, set the corresponding bits in the input enable register PORTx_INEN. When enabled, a read from the PORTx register returns the logical state of the input pin. The input signal does not overwrite the state of the flip-flop used for the output case. That state can only be altered by software. If the input driver is enabled, a write to the PORTxregister 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 corresponding bits in the direction registers. Direction registers are implemented as a pair of write-1-to-set (W1S) and write-1-to-clear (W1C) MMRs, called PORTx_DIR_SETand PORTx_DIR_CLEAR. This way, 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 PORTx_DIR_SETand PORTx_DIR_CLEARregisters return the same value when read and a logical 1 indicates an enabled output. The state of output pins is controlled by the PORTx registers. A logical 0 drives the output low while a logical 1 drives the output high.

While the PORTx register can be written to alter all GPIOs of a specific port at once, there is also a pair of W1S and W1C MMRs, called PORTx_SETand PORTx_CLEARthat enable manipulation of individual GPIO outputs. The state of the outputs can be obtained by reading the PORTx registers. 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 flip-flop 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 PORTxor PORTx_CLEARregister then set the one bit in the PORTx_INENregister. Read from the PORTxregister 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 PORTx_DIR_SETand PORTx_DIR_CLEARregister 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. When a GPIO port is used in open-drain mode, care must be taken not to exceed the V_{IH} operating condition associated with the respective pins.

Port Multiplexing Control

To configure pins properly, it is necessary to determine which bits in the PORT_FER and PORT_MUX register map to the pin of interest, and set them appropriately according to the desired function.

By default, after reset, all port pins are in GPIO input mode with their output and input drivers disabled. As a result, all unused port pins can be left unconnected. Disabled pins appear in high-impedance mode to external circuits and are pulled low to internal logic.

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.

NOTE: In this chapter, the naming convention for registers and bits omits the alphabetic group enumeration to refer to all/any of the port groups. For example PORT_FER represents PORTA_FER, PORTB_FER, and so on. Likewise P1 represents PA1, PB1, and so on.

The Function Enable register specifies whether the pin is being used as a GPIO pin, or another function, but does not specifies what that other function is. Each bit in the 16-bit PORT_FER register represents one

port pin. For example, bit 1 of the PORT_FER register set the PA1 pin to GPIO operations mode when cleared. When set, one of the available peripheral functions becomes active.

Every pair of bits in the PORT_MUX register controls the multiplexing between the peripheral functions available to a pin. This is a 2-bit bit field because some pins provide up to four options. The truth table of the bit field is identical to all family derivatives, regardless all options are available on the specific part.

Refer to the Signal Muxing table in the data sheet for the specific PORT_MUX settings.

ADSP-BF60x Multiplexing Scheme

ADSP-BF60x Blackfin processors feature a rich set of on-chip peripherals. Each set of peripherals has a combination of input and output signals associated with them. In total, there are many more signals than pins available on the processors. Therefore, a powerful pin multiplexing scheme provides best flexibility to external application space.

The **General-Purpose and Special Function Signals** table shows all peripheral signals that are accessible off the chip through the general-purpose ports. The processor does not feature all the listed peripherals at the same time. Note that some signals are optional and are not necessarily required in all operating modes.

Table 14-7: General-Purpose and Special Function Signals

Module	Signals	Ports
SMC0	All	A, B
	Address(23)	A, B
	Bus (2)	B
	Memory Select(3)	B
	Miscellaneous (3)	B
PPI0	All	D, E, F
	Data	D, E, F
	Frame Sync	E
	CLK	E
PPI1	All	B, C, D
	Data	C, D
	Frame Sync	B, D
	CLK	B
PPI2	All	A, B
	Data	A
	Frame Sync	B
	CLK	B

Table 14-7: General-Purpose and Special Function Signals (Continued)

Module	Signals	Ports
LP0	All	A, B
	Data	A
	CLK	B
	ACK	B
LP1	All	B, C
	Data	B
	CLK	C
	ACK	C
LP2	All	E, F
	Data	F
	CLK	E
	ACK	E
LP3	All	E, F
	Data	F
	CLK	E
	ACK	E
TM0	All	B, D, E, G
	Timers	E, G
	CLKs	B, D, G
	Alternate Capture Input (ACI)	B, D, G
SPT0	All	B
SPT1	All	D, E
	Channel A Data	D
	Miscellaneous	E
SPT2	All	E, G
	SPT2_ATDV	E
	Miscellaneous	G
ETH0	All	B, C, D
	Data	C
	ETH0_PHYINT	D
	Miscellaneous	B, C

Table 14-7: General-Purpose and Special Function Signals (Continued)

Module	Signals	Ports
ETH1	All	C, E, G
	Data	E, G
	ETH1_PTPPPS	C
	Miscellaneous	E, G
SPI0	All	C, D
	SEL 4,6,7	C
	Miscellaneous	D
SPI1	All	C, D, E
	SEL7	C
	D2, D3, RDY	E
	Miscellaneous	D
UART0	All	D
UART1	All	G
RSI0	All	E, G
	Data 3-7	E
	Miscellaneous	G
PWM0	All	E, F
	SYNC/TRIP0	E
	Miscellaneous	F
PWM1	All	E, G
	Channel C, D, B	E
	Miscellaneous	G
ACM0	All	F, G, E
	Address	F
	T1	G
	Miscellaneous	E
GPIOS	All	All

PORT Event Control

The following sections describe event generation in the PORT module.

PORT Interrupt Signals

The pin interrupts are completely decoupled from GPIO functionality which has the following advantages.

- Flexible mapping scheme enables pins from up to four different ports to be grouped into one common interrupt scheme.
- Interrupts work on input and output pins regardless of whether in GPIO or functional mode.

The ADSP-BF60x Blackfin processors have a number of interrupt channels dedicated to pin interrupts. These channels are managed by a set of hardware blocks named PINTx. Every PINTx block can sense up to 32 GPIO pins as described in the following list and shown in the figure below.

- PINT0 can sense pins of PORTA and PORTB
- PINT1 can sense pins of PORTB and PORTC
- PINT2 can sense pins from PORTC and PORTD
- PINT3 can sense pins from PORTD and PORTE
- PINT4 can sense pins from PORTE and PORTF
- PINT5 can sense pins from PORTF and PORTG

Both 32-bit and 16-bit peripheral bus accesses to PINTx registers are supported.

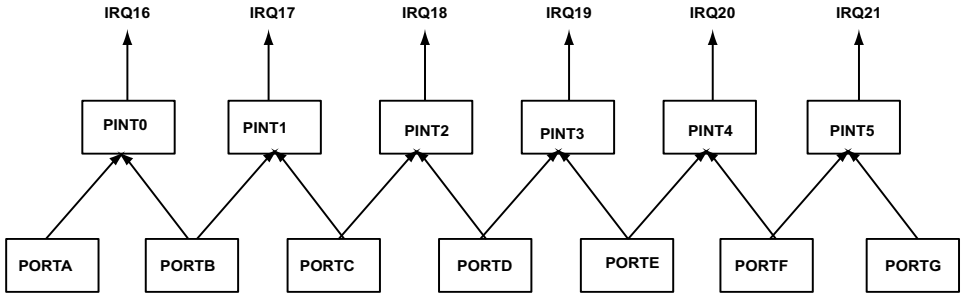


Figure 14-2: GPIO to PINTx Assignment

Pins are connected to the PINTx module and then to the system event controller. Special attention is required with regard to how the pins are assigned to the PINTx modules as shown in the PINTx block diagram below.

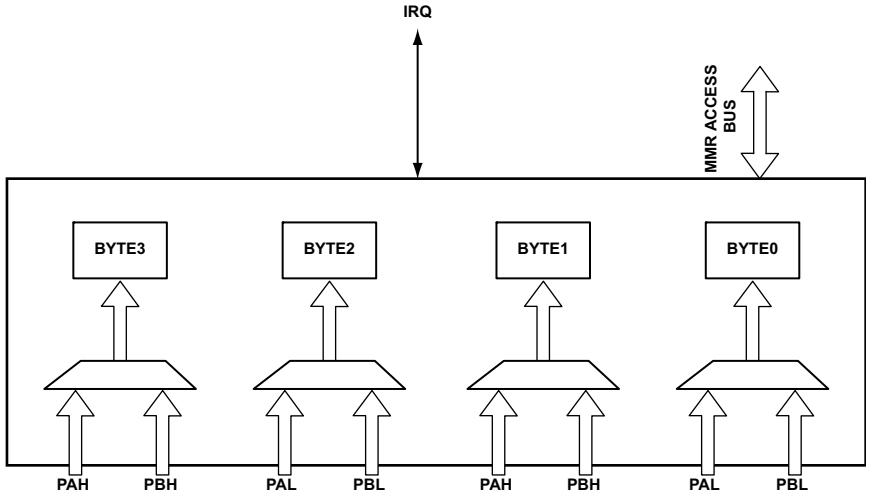


Figure 14-3: PINTx Block Diagram

The ports are subdivided into 8-bit half ports, with lower and upper half 8-bit units. The PINTx_ASSIGN registers control the 8-bit multiplexers shown in the Block Diagram. Lower half units of eight pins can be forwarded to either byte 0 or byte 2 of either associated PINTx block. Upper half units can be forwarded to either byte 1 or byte 3 of the pin interrupts blocks, without further restrictions.

When a half port is assigned to a byte in any PINTx block, the state of the eight pins (regardless of GPIO or function, input or output) can be seen in the PINTx_PINSTATE register. When neither input nor output drivers of the pin are enabled, the pin state is read as zero. The PINTx_PINSTATE register reports the inverted state of the pin if the signal inverter is activated by the PINTx_INVERT_SET register. The inverter can be enabled on a individual bit by bit basis. Every bit in the PINTx_INVERT_SET/ PINTx_INVERT_CLEAR register pair represents a pin signal.

An interrupt can be generated on an active high level of the signal or a rising edge of the signal. The default behavior is level sensitivity. The PINTx_EDGE_SET register can be used to change the behavior to edge sensitivity. By enabling the inverter using the PINTx_INVERT_SET register, the interrupt behavior can be altered to trigger on active-low signals or falling edges.

The PINTx modules also assist if both signals are required to generate interrupts. If two different interrupt requests are required, the PINTx_ASSIGN registers can route a signal to two different PINTx blocks, where one block inverts the signal and the other one does not. If both signal edges can report over the same interrupt, every signal can be routed through to different bit positions within a single PINTx block, where the inverted signal should be enabled for either one. The servicing software routine can then tell from the PINTx_LATCH register whether a falling, a rising, or both edges have occurred.

Regardless of whether using level-sensitive or edge-sensitive mode, an interrupt is always latched by the hardware. Latched signals can be read from the PINTx_LATCH registers. Latches can only be cleared by a software or a hardware reset. To clear, W1C the PINTx_REQUEST or the PINTx_LATCH register. If the pin state does not change by the time the interrupt service routine returns, the interrupt is requested again when in level-sensitive mode.

Because every PINTx block groups up to 32 pin signals, the PINTx_MASK_SET/ PINTx_INVERT_CLEAR register pair can control which of the signals can request an interrupt at the system level. Software may interrogate the PINTx_REQUEST register for signaling pins. The PINTx_REQUEST bits represent a logical AND between the mask and the latch. When any of these bits is set, an interrupt is forwarded to the SIC controller.

All MMR registers in the pin interrupt module are 32 bits wide. Individual bits of the PINTx registers represent the associated pins. Nevertheless, the 32 bits can also be seen as four groups of eight pins. Each group can manage up to eight pins out of either the lower or an upper half of any associated port.

PORT Programming Model

The following sections description of the overall program model of the general purpose ports.

GPIO Programming Model Flow (Part 1), **GPIO Programming Model Flow (Part 2)**, and **GPIO Programming Model Flow (Part 3)** show the programming model of the general-purpose ports. This includes the GPIO input and output operation, open-drain mode, and the pin interrupt PINTx modules.

NOTE: These process flow diagrams connect where callout letters appear. For example, callout "A" on the **GPIO Programming Model Flow (Part 1)** diagram connects to callout "A" on the **GPIO Programming Model Flow (Part 2)** diagram.

The following flow charts describe the processes for setting up pins for different available functionality. Begin the process from the **GPIO Programming Model Flow (Part 1)** chart. The first decision effect the value of the PORT_FER register, shown at "1", for peripheral functions this should be set. For more information on setting up for peripheral functions refer to the *Port Multiplexing Control*

If the pin is to be a GPIO pin, a series of decisions then need to be made. There are several configuration registers that need to be considered: PORT_DATA, PORT_INV, PORT_DIR, and PORT_INEN. Depending on the type of GPIO pin desired, the configurations may or may not be applicable, and can have different meanings. The following paragraphs describe in brief the function of the different settings for each of the pin functions in GPIO mode: Input, Output, and Open-drain. For all registers the SET/CLR versions of the register are recommended to be used. For more detailed descriptions of the configurations, see *ADSP-BF60x PORT Register Descriptions*.

For Output mode, all the pins should always first be made low using PORT_DATAregister. The PORT_DIR register is used to define the direction of each pin (output). In this mode, the other registers aren't of any consequence. This flow can be seen starting at label "2" in **GPIO Programming Model Flow (Part 1)** chart.

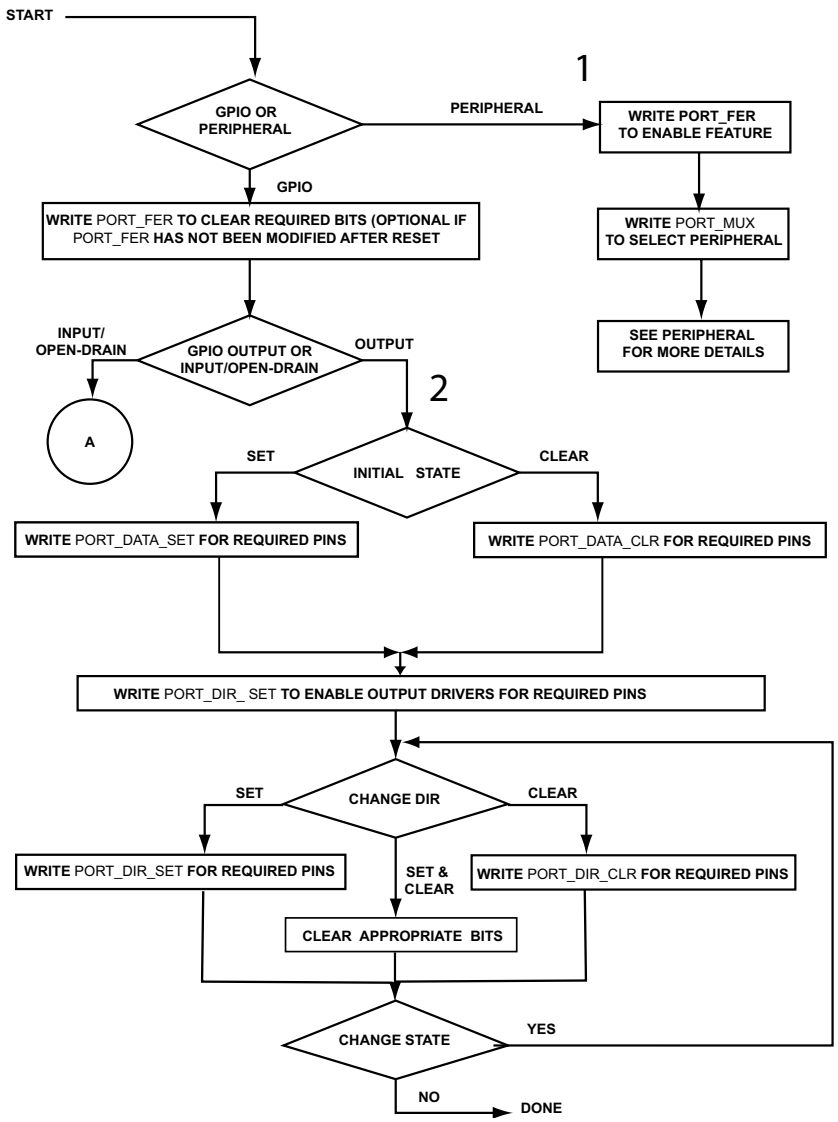


Figure 14-4: GPIO Programming Model Flow (Part 1)

For Input mode, the the polarity must be first decided for each pin using the PINT_INV register. The PORT_DIR register of course must be set to define the appropriate pins for input. If interrupts are desireable a serious of steps must be taken to configure the PINT module according. These steps are shown starting at "B" in the **GPIO Programming Model Flow (Part 3)** chart. Finally, the PORT_INEN register used to enable the associated input drivers. This entire flow can be seen starting at "3" in the **GPIO Programming Model Flow (Part 2)** chart.

For Open Drain mode, all the pins should always be first made low using PORT_DATA. PORT_INENshould then be used to enable the appropriate input drivers. PORT_DIR should be set in this mode to indicate whether the pin is in active state or not (active being 0). This flow can be seen starting at "4" in the **GPIO Programming Model Flow (Part 2)** chart.

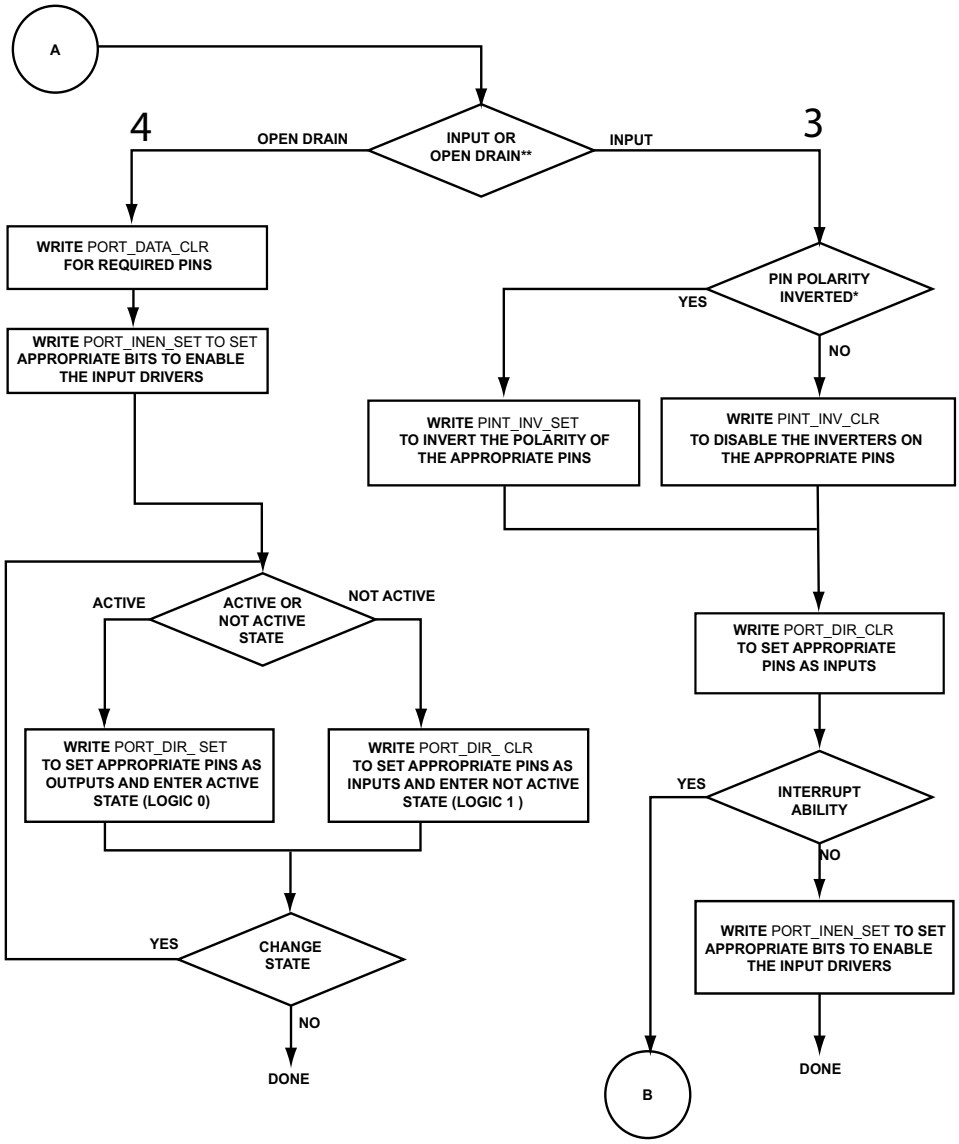


Figure 14-5: GPIO Programming Model Flow (Part 2)

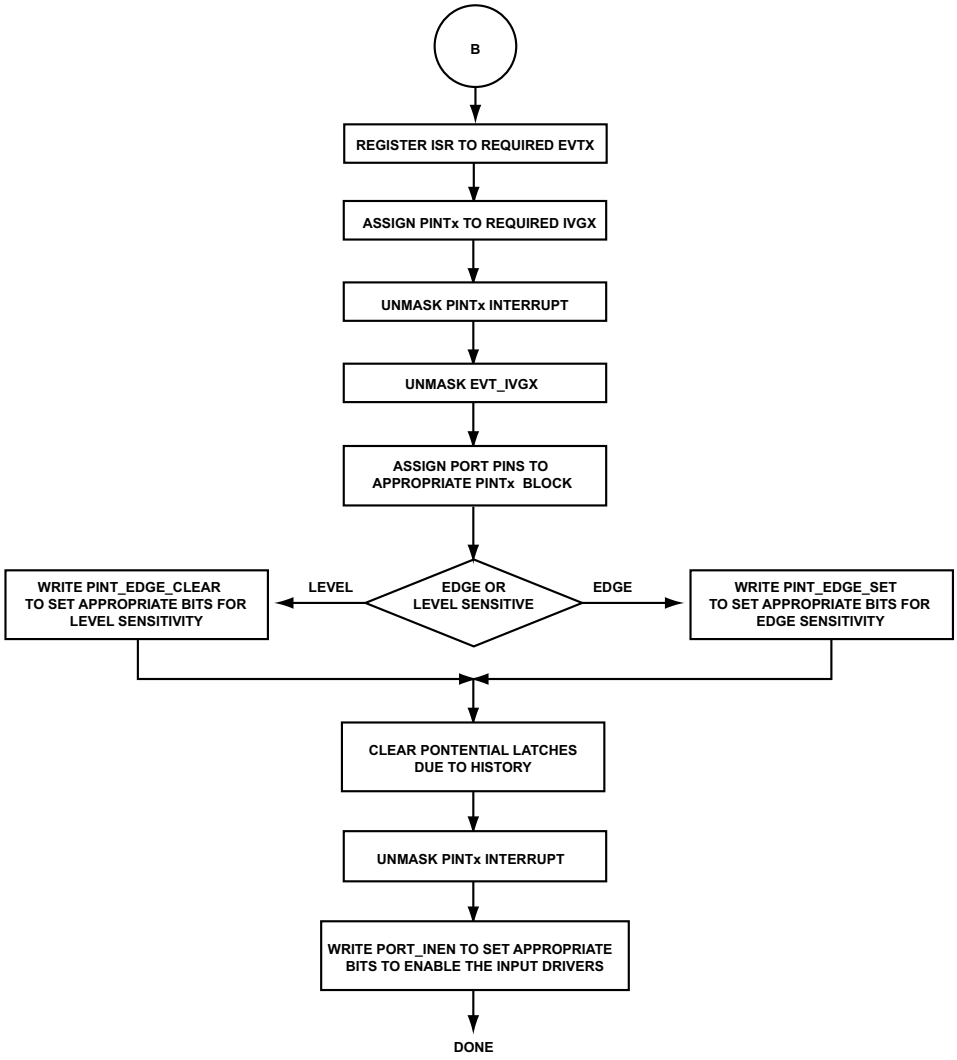


Figure 14-6: GPIO Programming Model Flow (Part 3)

ADSP-BF60x PORT Register Descriptions

General Purpose Input/Output (PORT) contains the following registers.

Table 14-8: ADSP-BF60x PORT Register List

Name	Description
PORT_FER	Port x Function Enable Register
PORT_FER_SET	Port x Function Enable Set Register

Table 14-8: ADSP-BF60x PORT Register List (Continued)

Name	Description
PORT_FER_CLR	Port x Function Enable Clear Register
PORT_DATA	Port x GPIO Data Register
PORT_DATA_SET	Port x GPIO Data Set Register
PORT_DATA_CLR	Port x GPIO Data Clear Register
PORT_DIR	Port x GPIO Direction Register
PORT_DIR_SET	Port x GPIO Direction Set Register
PORT_DIR_CLR	Port x GPIO Direction Clear Register
PORT_INEN	Port x GPIO Input Enable Register
PORT_INEN_SET	Port x GPIO Input Enable Set Register
PORT_INEN_CLR	Port x GPIO Input Enable Clear Register
PORT_MUX	Port x Multiplexer Control Register
PORT_DATA_TGL	Port x GPIO Input Enable Toggle Register
PORT_POL	Port x GPIO Polarity Invert Register
PORT_POL_SET	Port x GPIO Polarity Invert Set Register
PORT_POL_CLR	Port x GPIO Polarity Invert Clear Register
PORT_LOCK	Port x GPIO Lock Register

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.

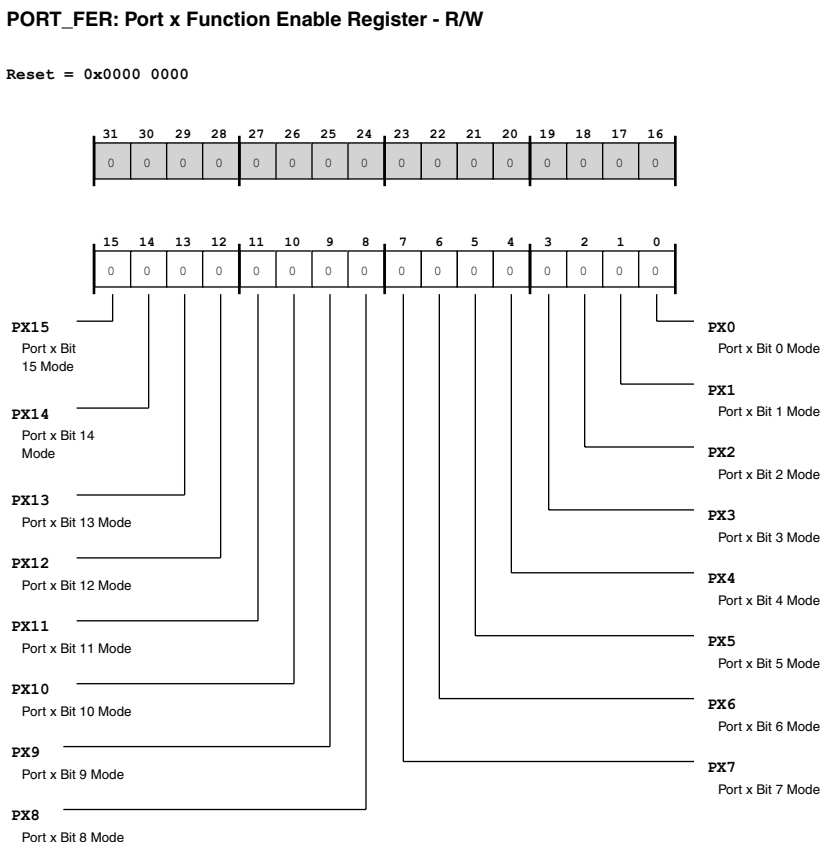


Figure 14-7: PORT_FER Register Diagram

Table 14-9: PORT_FER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
14 (R/W)	PX14	Port x Bit 14 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
13 (R/W)	PX13	Port x Bit 13 Mode.	
		0	GPIO Mode
		1	Peripheral Mode

Table 14-9: PORT_FER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	PX12	Port x Bit 12 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
11 (R/W)	PX11	Port x Bit 11 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
10 (R/W)	PX10	Port x Bit 10 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
9 (R/W)	PX9	Port x Bit 9 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
8 (R/W)	PX8	Port x Bit 8 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
7 (R/W)	PX7	Port x Bit 7 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
6 (R/W)	PX6	Port x Bit 6 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
5 (R/W)	PX5	Port x Bit 5 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
4 (R/W)	PX4	Port x Bit 4 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
3 (R/W)	PX3	Port x Bit 3 Mode.	
		0	GPIO Mode
		1	Peripheral Mode

Table 14-9: PORT_FER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	PX2	Port x Bit 2 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
1 (R/W)	PX1	Port x Bit 1 Mode.	
		0	GPIO Mode
		1	Peripheral Mode
0 (R/W)	PX0	Port x Bit 0 Mode.	
		0	GPIO Mode
		1	Peripheral 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.

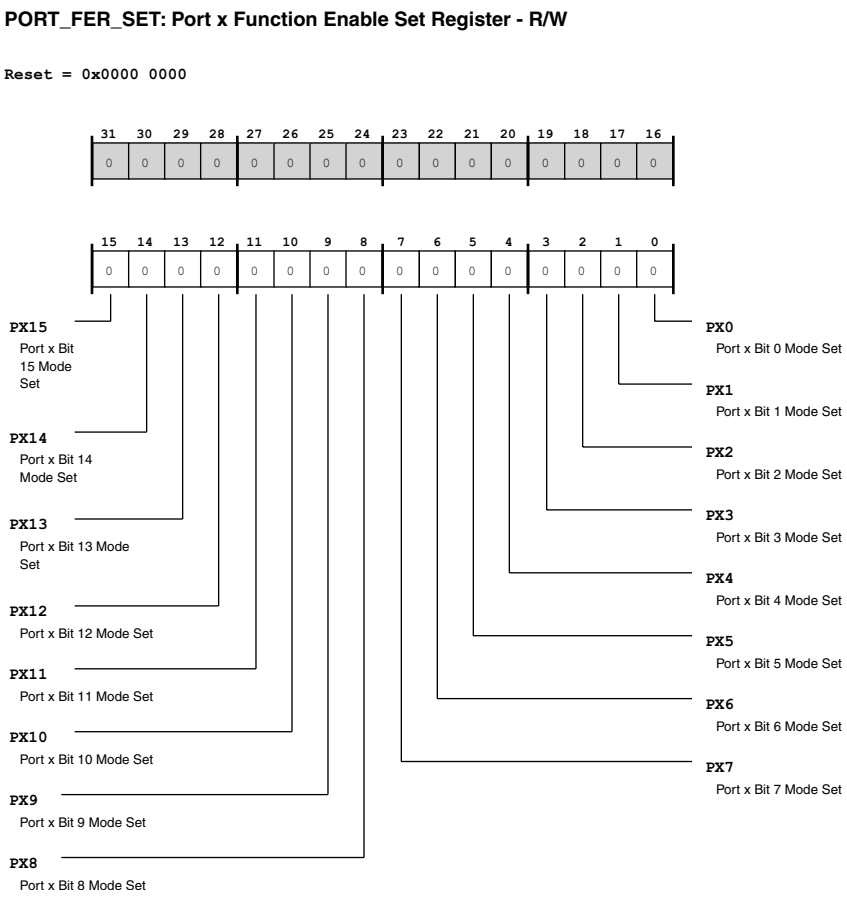


Figure 14-8: PORT_FER_SET Register Diagram

Table 14-10: PORT_FER_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
14 (R/W1S)	PX14	Port x Bit 14 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
13 (R/W1S)	PX13	Port x Bit 13 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode

Table 14-10: PORT_FER_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1S)	PX12	Port x Bit 12 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
11 (R/W1S)	PX11	Port x Bit 11 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
10 (R/W1S)	PX10	Port x Bit 10 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
9 (R/W1S)	PX9	Port x Bit 9 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
8 (R/W1S)	PX8	Port x Bit 8 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
7 (R/W1S)	PX7	Port x Bit 7 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
6 (R/W1S)	PX6	Port x Bit 6 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
5 (R/W1S)	PX5	Port x Bit 5 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
4 (R/W1S)	PX4	Port x Bit 4 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
3 (R/W1S)	PX3	Port x Bit 3 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode

Table 14-10: PORT_FER_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1S)	PX2	Port x Bit 2 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
1 (R/W1S)	PX1	Port x Bit 1 Mode Set.	
		0	No Effect
		1	Set Bit for Peripheral Mode
0 (R/W1S)	PX0	Port x Bit 0 Mode Set.	
		0	No Effect
		1	Set Bit for 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.

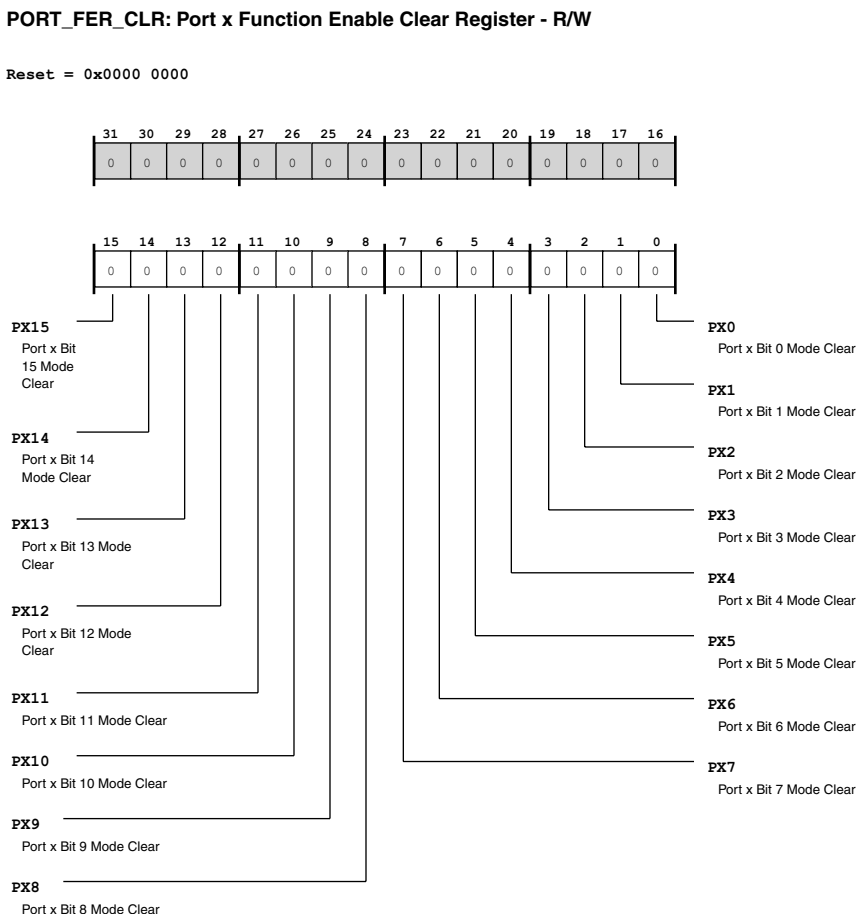


Figure 14-9: PORT_FER_CLR Register Diagram

Table 14-11: PORT_FER_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
14 (R/W1C)	PX14	Port x Bit 14 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
13 (R/W1C)	PX13	Port x Bit 13 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode

Table 14-11: PORT_FER_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1C)	PX12	Port x Bit 12 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
11 (R/W1C)	PX11	Port x Bit 11 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
10 (R/W1C)	PX10	Port x Bit 10 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
9 (R/W1C)	PX9	Port x Bit 9 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
8 (R/W1C)	PX8	Port x Bit 8 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
7 (R/W1C)	PX7	Port x Bit 7 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
6 (R/W1C)	PX6	Port x Bit 6 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
5 (R/W1C)	PX5	Port x Bit 5 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
4 (R/W1C)	PX4	Port x Bit 4 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
3 (R/W1C)	PX3	Port x Bit 3 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode

Table 14-11: PORT_FER_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	PX2	Port x Bit 2 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
1 (R/W1C)	PX1	Port x Bit 1 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode
0 (R/W1C)	PX0	Port x Bit 0 Mode Clear.	
		0	No Effect
		1	Set Bit for GPIO Mode

Port x GPIO Data Register

The PORT_DATA register operates differently for port bits/pins, depending 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, and 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.

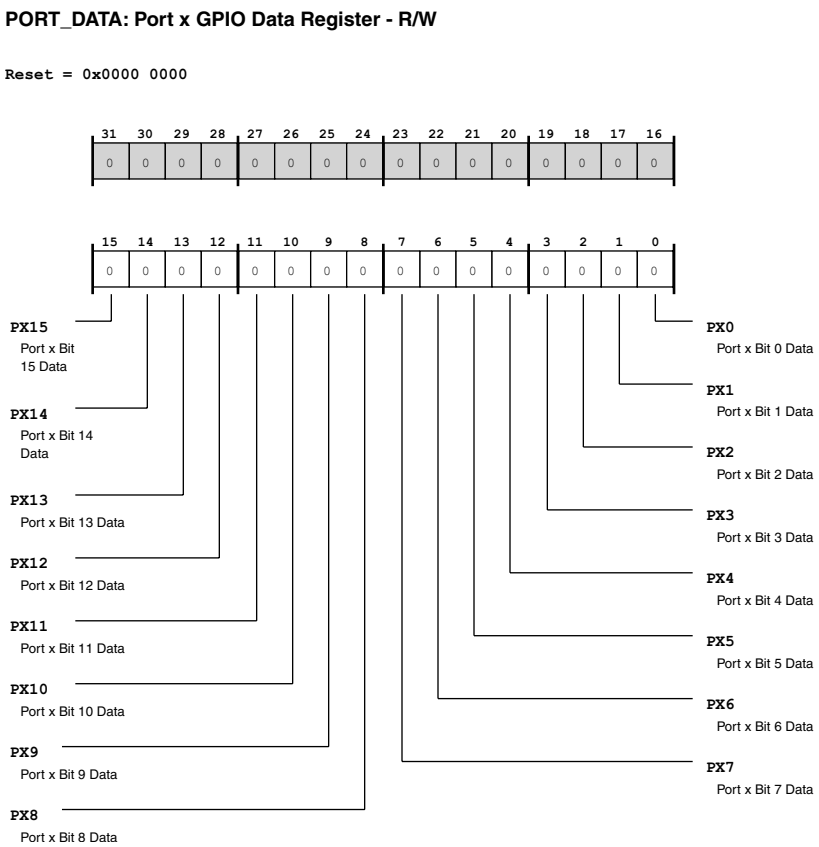


Figure 14-10: PORT_DATA Register Diagram

Table 14-12: PORT_DATA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Data.	
		0	Signal Low
		1	Signal High
14 (R/W)	PX14	Port x Bit 14 Data.	
		0	Signal Low
		1	Signal High
13 (R/W)	PX13	Port x Bit 13 Data.	
		0	Signal Low
		1	Signal High

Table 14-12: PORT_DATA Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	PX12	Port x Bit 12 Data.	
		0	Signal Low
		1	Signal High
11 (R/W)	PX11	Port x Bit 11 Data.	
		0	Signal Low
		1	Signal High
10 (R/W)	PX10	Port x Bit 10 Data.	
		0	Signal Low
		1	Signal High
9 (R/W)	PX9	Port x Bit 9 Data.	
		0	Signal Low
		1	Signal High
8 (R/W)	PX8	Port x Bit 8 Data.	
		0	Signal Low
		1	Signal High
7 (R/W)	PX7	Port x Bit 7 Data.	
		0	Signal Low
		1	Signal High
6 (R/W)	PX6	Port x Bit 6 Data.	
		0	Signal Low
		1	Signal High
5 (R/W)	PX5	Port x Bit 5 Data.	
		0	Signal Low
		1	Signal High
4 (R/W)	PX4	Port x Bit 4 Data.	
		0	Signal Low
		1	Signal High
3 (R/W)	PX3	Port x Bit 3 Data.	
		0	Signal Low
		1	Signal High

Table 14-12: PORT_DATA Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	PX2	Port x Bit 2 Data.	
		0	Signal Low
		1	Signal High
1 (R/W)	PX1	Port x Bit 1 Data.	
		0	Signal Low
		1	Signal High
0 (R/W)	PX0	Port x Bit 0 Data.	
		0	Signal Low
		1	Signal High

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.

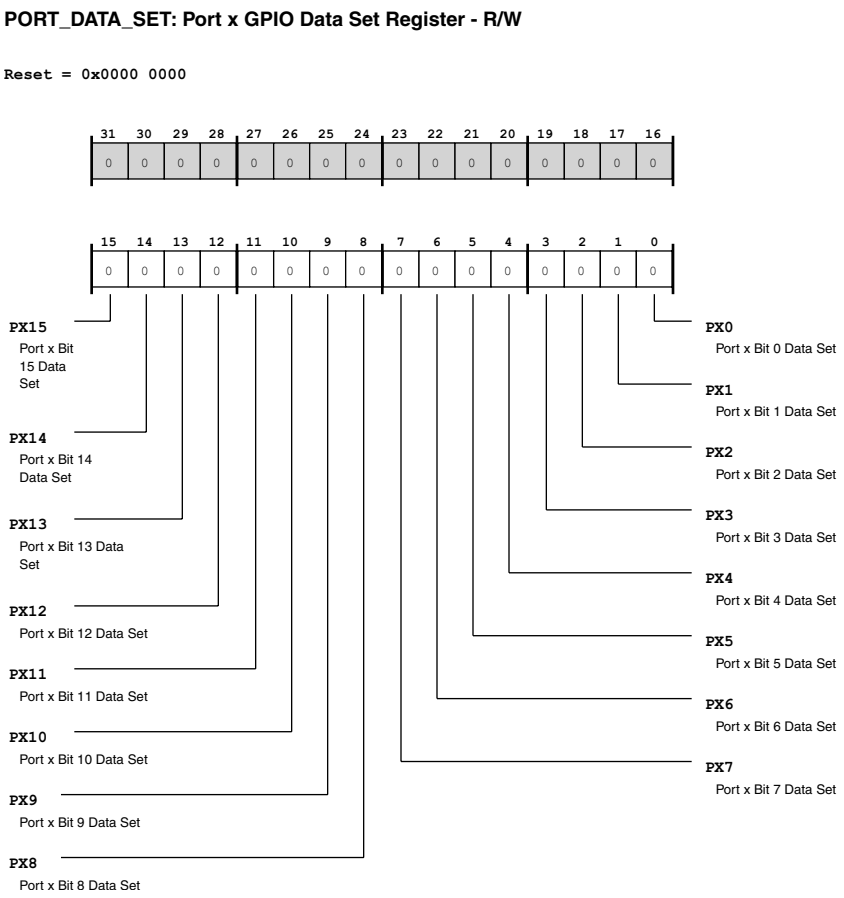


Figure 14-11: PORT_DATA_SET Register Diagram

Table 14-13: 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.

Table 14-13: PORT_DATA_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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.
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.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.

Table 14-13: PORT_DATA_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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.
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.

Table 14-13: PORT_DATA_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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 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.

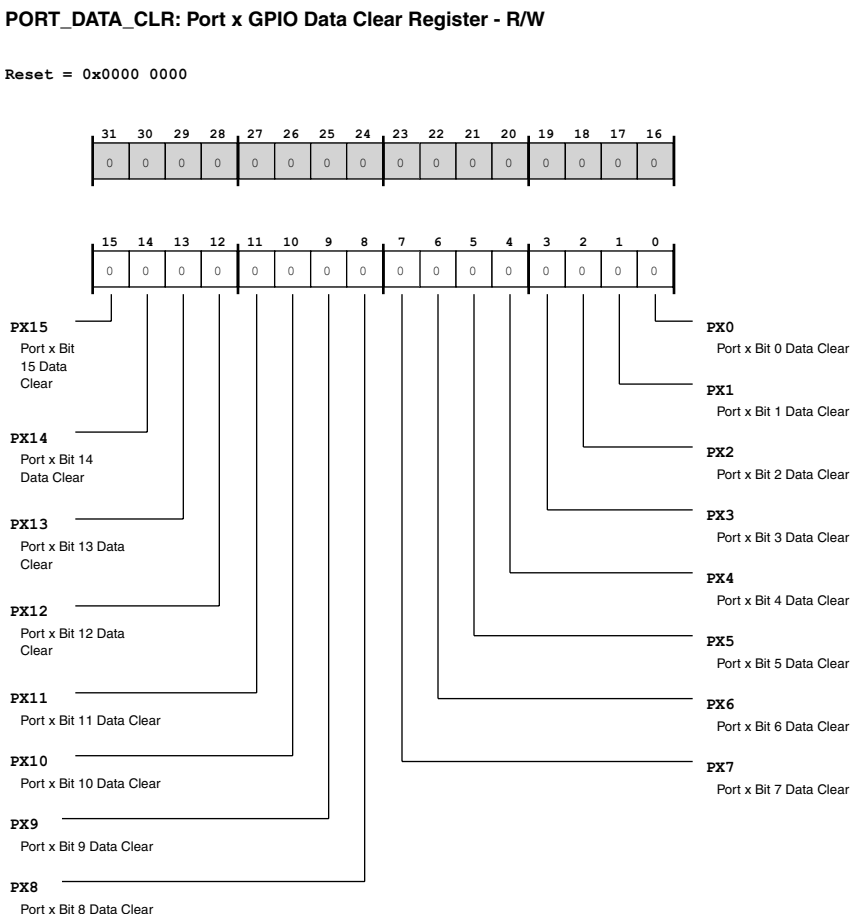


Figure 14-12: PORT_DATA_CLR Register Diagram

Table 14-14: PORT_DATA_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Data Clear.	
		0	No Effect
		1	Clear Bit Write 1 for signal low in output mode.
14 (R/W1C)	PX14	Port x Bit 14 Data Clear.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.

Table 14-14: PORT_DATA_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1C)	PX13	Port x Bit 13 Data Clear.	
		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.	
		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.	
		0	No Effect
		1	Clear Bit Write 1 for signal low in output mode.
10 (R/W1C)	PX10	Port x Bit 10 Data Clear.	
		0	No Effect Write 0 has no effect in output mode.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.	
		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.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.

Table 14-14: PORT_DATA_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	PX7	Port x Bit 7 Data Clear.	
		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.	
		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.	
		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.	
		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.	
		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.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.

Table 14-14: PORT_DATA_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	PX1	Port x Bit 1 Data Clear.	
		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.	
		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 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 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, and a 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 input mode, but the input drivers are not enabled. To enable any GPIO input drivers, set the corresponding bits in 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, 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, care must be taken not to exceed the VIH operating condition associated with the respective pin.

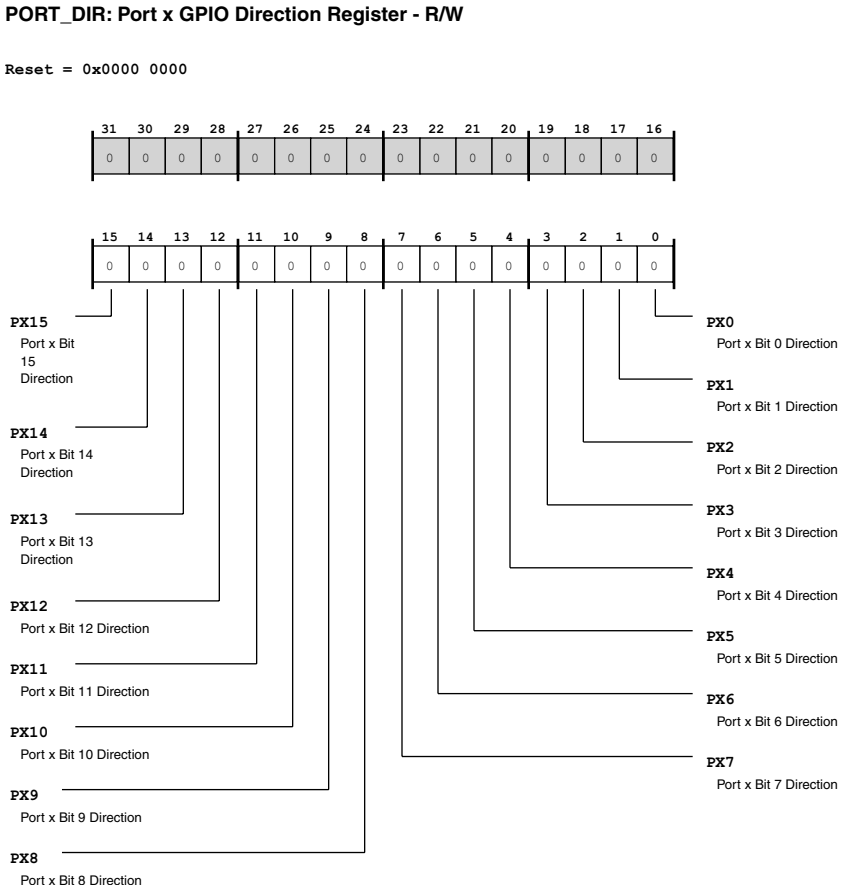


Figure 14-13: PORT_DIR Register Diagram

Table 14-15: PORT_DIR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.

Table 14-15: PORT_DIR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	PX14	Port x Bit 14 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
13 (R/W)	PX13	Port x Bit 13 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
12 (R/W)	PX12	Port x Bit 12 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
11 (R/W)	PX11	Port x Bit 11 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
10 (R/W)	PX10	Port x Bit 10 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
9 (R/W)	PX9	Port x Bit 9 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.

Table 14-15: PORT_DIR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	PX8	Port x Bit 8 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
7 (R/W)	PX7	Port x Bit 7 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
6 (R/W)	PX6	Port x Bit 6 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
5 (R/W)	PX5	Port x Bit 5 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
4 (R/W)	PX4	Port x Bit 4 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
3 (R/W)	PX3	Port x Bit 3 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.

Table 14-15: PORT_DIR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	PX2	Port x Bit 2 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
1 (R/W)	PX1	Port x Bit 1 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.
0 (R/W)	PX0	Port x Bit 0 Direction.	
		0	Input mode Output driver disabled.
		1	Output mode Output driver enabled.

Port x GPIO Direction Set Register

The PORT_DIR_SET register enable output mode and enables output drivers for GPIO pins. For more information, see the PORT_DIR register description.

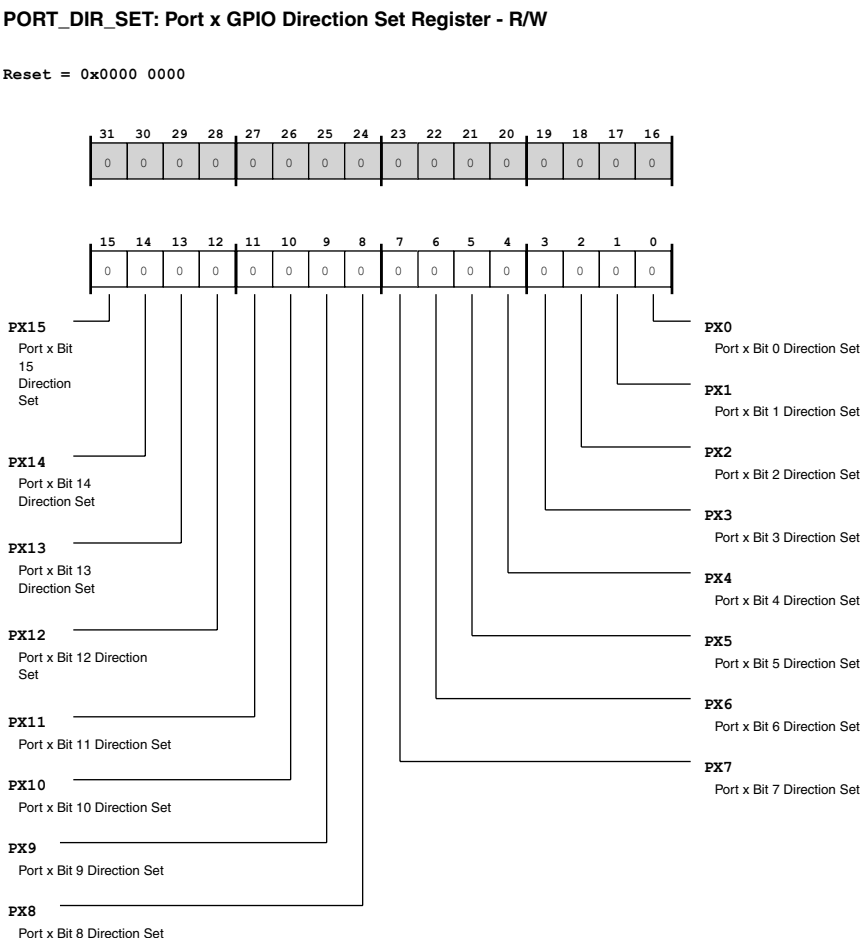


Figure 14-14: PORT_DIR_SET Register Diagram

Table 14-16: PORT_DIR_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Direction Set.	
		0	No Effect
		1	Set Bit Set to enable output mode/driver.
14 (R/W1S)	PX14	Port x Bit 14 Direction Set.	
		0	No Effect
		1	Set Bit Set to enable output mode/driver.

Table 14-16: PORT_DIR_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1S)	PX13	Port x Bit 13 Direction Set.	
		0	No Effect
		1	Set Bit Set to enable output mode/driver.
12 (R/W1S)	PX12	Port x Bit 12 Direction Set.	
		0	No Effect
		1	Set Bit Set to enable output mode/driver.
11 (R/W1S)	PX11	Port x Bit 11 Direction Set.	
		0	No Effect
		1	Set Bit Set to enable output mode/driver.
10 (R/W1S)	PX10	Port x Bit 10 Direction Set.	
		0	No Effect
		1	Set Bit Set to enable output mode/driver.
9 (R/W1S)	PX9	Port x Bit 9 Direction Set.	
		0	No Effect
		1	Set Bit Set to enable output mode/driver.
8 (R/W1S)	PX8	Port x Bit 8 Direction Set.	
		0	No Effect
		1	Set Bit Set to enable output mode/driver.
7 (R/W1S)	PX7	Port x Bit 7 Direction Set.	
		0	No Effect
		1	Set Bit Set to enable output mode/driver.
6 (R/W1S)	PX6	Port x Bit 6 Direction Set.	
		0	No Effect
		1	Set Bit Set to enable output mode/driver.

Table 14-16: PORT_DIR_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1S)	PX5	Port x Bit 5 Direction Set.	
		0	No Effect
		1	Set Bit Set to enable output mode/driver.
4 (R/W1S)	PX4	Port x Bit 4 Direction Set.	
		0	No Effect
		1	Set Bit Set to enable output mode/driver.
3 (R/W1S)	PX3	Port x Bit 3 Direction Set.	
		0	No Effect
		1	Set Bit Set to enable output mode/driver.
2 (R/W1S)	PX2	Port x Bit 2 Direction Set.	
		0	No Effect
		1	Set Bit Set to enable output mode/driver.
1 (R/W1S)	PX1	Port x Bit 1 Direction Set.	
		0	No Effect
		1	Set Bit Set to enable output mode/driver.
0 (R/W1S)	PX0	Port x Bit 0 Direction Set.	
		0	No Effect
		1	Set Bit Set to enable output mode/driver.

Port x GPIO Direction Clear Register

The `PORT_DIR_CLR` register disables output mode and disables output drivers for GPIO pins. For more information, see the `PORT_DIR` register description.

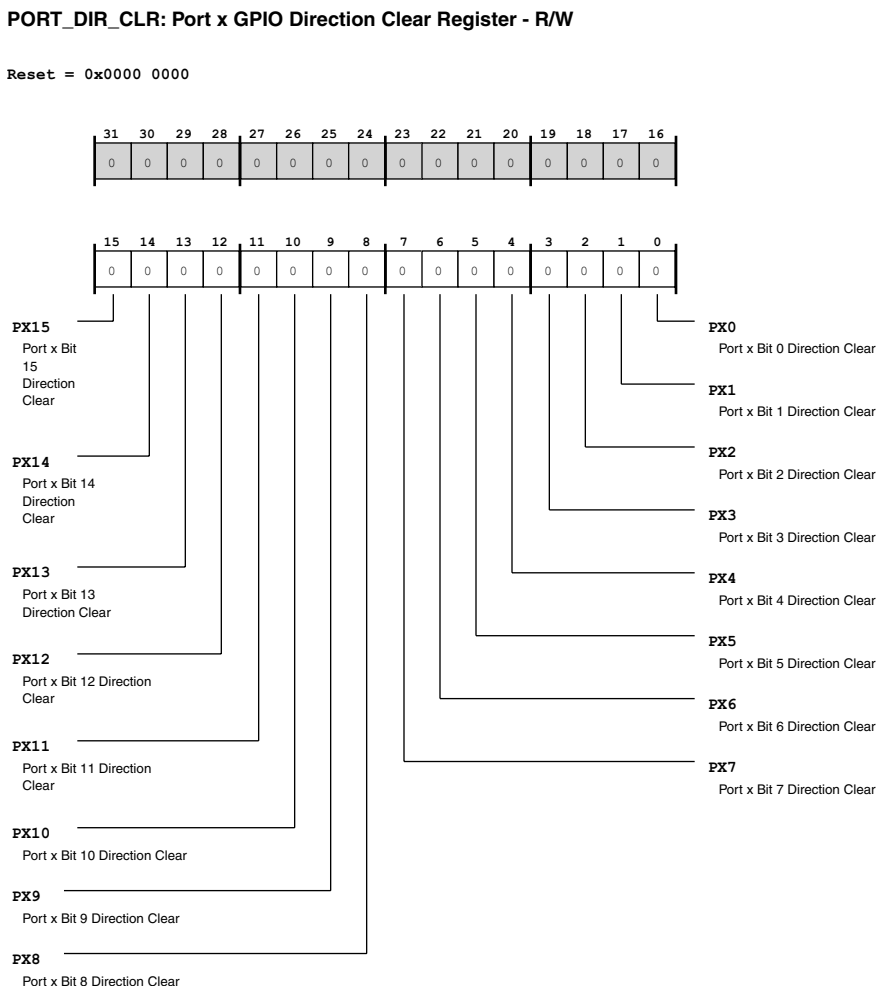


Figure 14-15: PORT_DIR_CLR Register Diagram

Table 14-17: PORT_DIR_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Direction Clear.	
		0	No Effect
		1	Clear Bit Set to disable output mode/driver.
14 (R/W1C)	PX14	Port x Bit 14 Direction Clear.	
		0	No Effect
		1	Clear Bit Set to disable output mode/driver.

Table 14-17: PORT_DIR_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1C)	PX13	Port x Bit 13 Direction Clear.	
		0	No Effect
		1	Clear Bit Set to disable output mode/driver.
12 (R/W1C)	PX12	Port x Bit 12 Direction Clear.	
		0	No Effect
		1	Clear Bit Set to disable output mode/driver.
11 (R/W1C)	PX11	Port x Bit 11 Direction Clear.	
		0	No Effect
		1	Clear Bit Set to disable output mode/driver.
10 (R/W1C)	PX10	Port x Bit 10 Direction Clear.	
		0	No Effect
		1	Clear Bit Set to disable output mode/driver.
9 (R/W1C)	PX9	Port x Bit 9 Direction Clear.	
		0	No Effect
		1	Clear Bit Set to disable output mode/driver.
8 (R/W1C)	PX8	Port x Bit 8 Direction Clear.	
		0	No Effect
		1	Clear Bit Set to disable output mode/driver.
7 (R/W1C)	PX7	Port x Bit 7 Direction Clear.	
		0	No Effect
		1	Clear Bit Set to disable output mode/driver.
6 (R/W1C)	PX6	Port x Bit 6 Direction Clear.	
		0	No Effect
		1	Clear Bit Set to disable output mode/driver.

Table 14-17: PORT_DIR_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1C)	PX5	Port x Bit 5 Direction Clear.	
		0	No Effect
		1	Clear Bit Set to disable output mode/driver.
4 (R/W1C)	PX4	Port x Bit 4 Direction Clear.	
		0	No Effect
		1	Clear Bit Set to disable output mode/driver.
3 (R/W1C)	PX3	Port x Bit 3 Direction Clear.	
		0	No Effect
		1	Clear Bit Set to disable output mode/driver.
2 (R/W1C)	PX2	Port x Bit 2 Direction Clear.	
		0	No Effect
		1	Clear Bit Set to disable output mode/driver.
1 (R/W1C)	PX1	Port x Bit 1 Direction Clear.	
		0	No Effect
		1	Clear Bit Set to disable output mode/driver.
0 (R/W1C)	PX0	Port x Bit 0 Direction Clear.	
		0	No Effect
		1	Clear Bit Set to disable output mode/driver.

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.

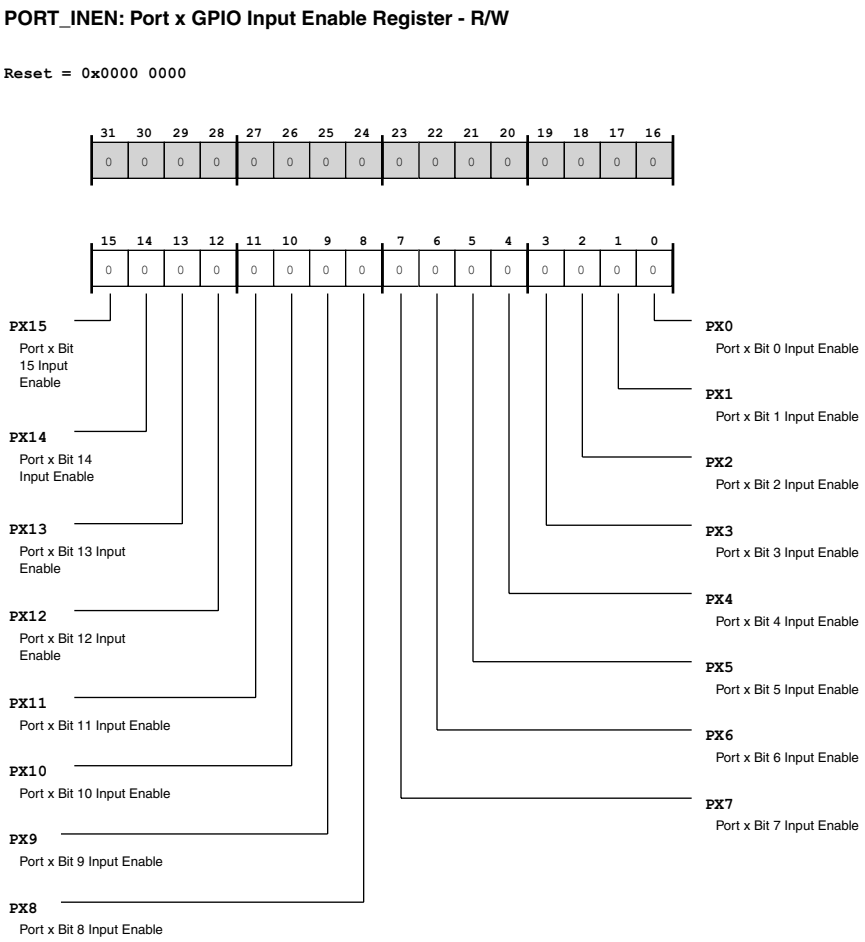


Figure 14-16: PORT_INEN Register Diagram

Table 14-18: 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 14-18: 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	Input disabled
		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 14-18: 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 Set Register

The PORT_INEN_SET register enables input drivers for GPIO pins. For more information, see the PORT_INEN register description.

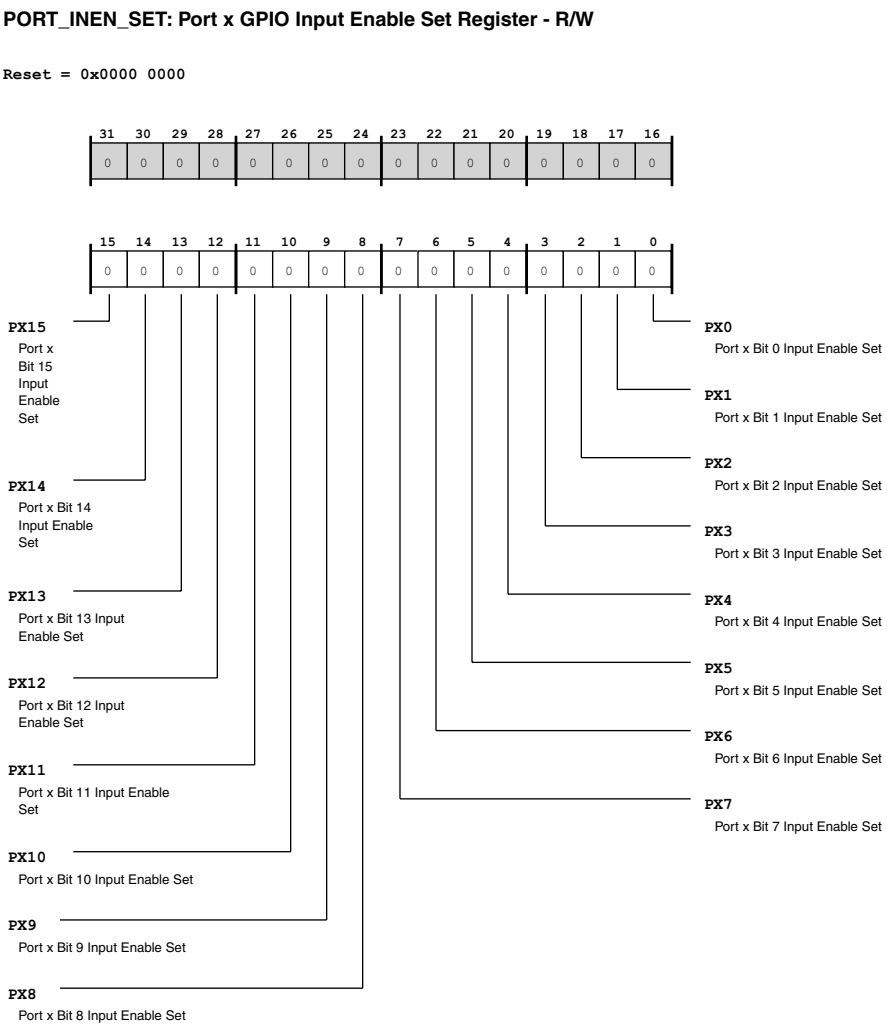


Figure 14-17: PORT_INEN_SET Register Diagram

Table 14-19: 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 input driver.
14 (R/W1S)	PX14	Port x Bit 14 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.

Table 14-19: PORT_INEN_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1S)	PX13	Port x Bit 13 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
12 (R/W1S)	PX12	Port x Bit 12 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
11 (R/W1S)	PX11	Port x Bit 11 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
10 (R/W1S)	PX10	Port x Bit 10 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
9 (R/W1S)	PX9	Port x Bit 9 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
8 (R/W1S)	PX8	Port x Bit 8 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
7 (R/W1S)	PX7	Port x Bit 7 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
6 (R/W1S)	PX6	Port x Bit 6 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.

Table 14-19: PORT_INEN_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1S)	PX5	Port x Bit 5 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
4 (R/W1S)	PX4	Port x Bit 4 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
3 (R/W1S)	PX3	Port x Bit 3 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
2 (R/W1S)	PX2	Port x Bit 2 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
1 (R/W1S)	PX1	Port x Bit 1 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.
0 (R/W1S)	PX0	Port x Bit 0 Input Enable Set.	
		0	No Effect
		1	Set Bit Set to enable input driver.

Port x GPIO Input Enable Clear Register

The `PORT_INEN_CLR` register disables input drivers for GPIO pins. For more information, see the `PORT_INEN` register description.

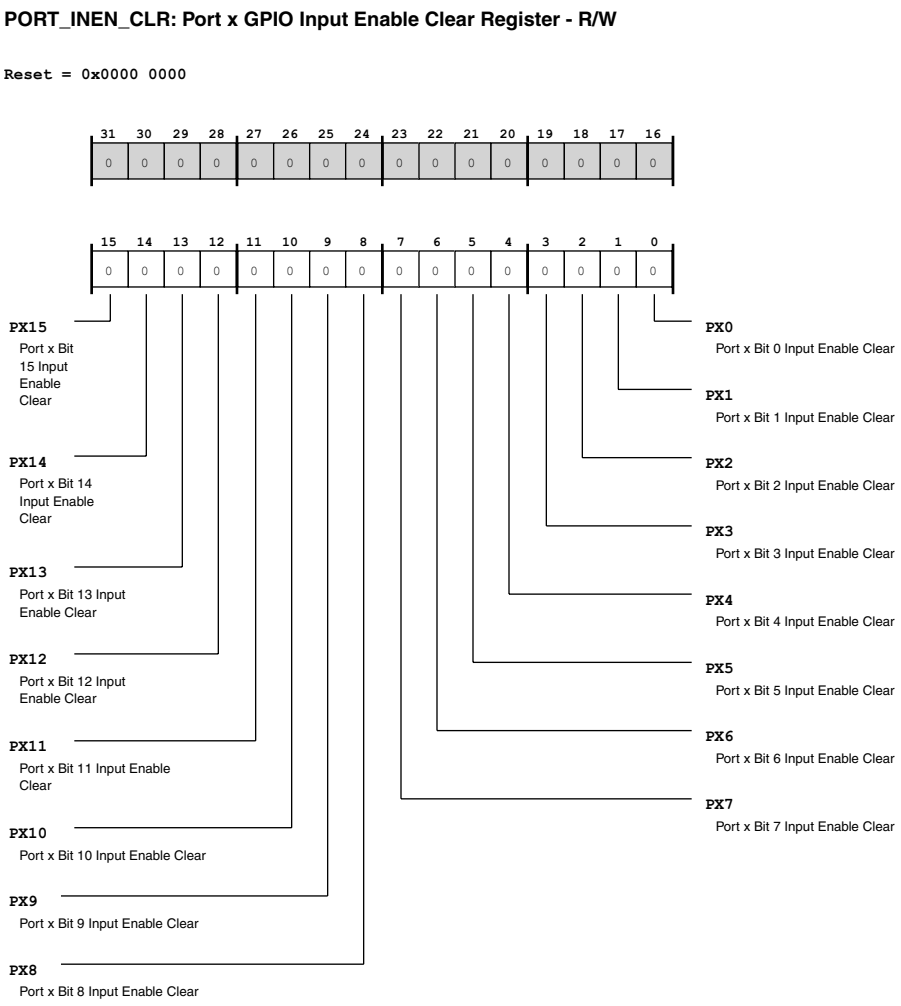


Figure 14-18: PORT_INEN_CLR Register Diagram

Table 14-20: 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 input driver.
14 (R/W1C)	PX14	Port x Bit 14 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.

Table 14-20: PORT_INEN_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1C)	PX13	Port x Bit 13 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
12 (R/W1C)	PX12	Port x Bit 12 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
11 (R/W1C)	PX11	Port x Bit 11 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
10 (R/W1C)	PX10	Port x Bit 10 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
9 (R/W1C)	PX9	Port x Bit 9 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
8 (R/W1C)	PX8	Port x Bit 8 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
7 (R/W1C)	PX7	Port x Bit 7 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
6 (R/W1C)	PX6	Port x Bit 6 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.

Table 14-20: PORT_INEN_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1C)	PX5	Port x Bit 5 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
4 (R/W1C)	PX4	Port x Bit 4 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
3 (R/W1C)	PX3	Port x Bit 3 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
2 (R/W1C)	PX2	Port x Bit 2 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
1 (R/W1C)	PX1	Port x Bit 1 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.
0 (R/W1C)	PX0	Port x Bit 0 Input Enable Clear.	
		0	No Effect
		1	Clear Bit Set to disable input driver.

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.

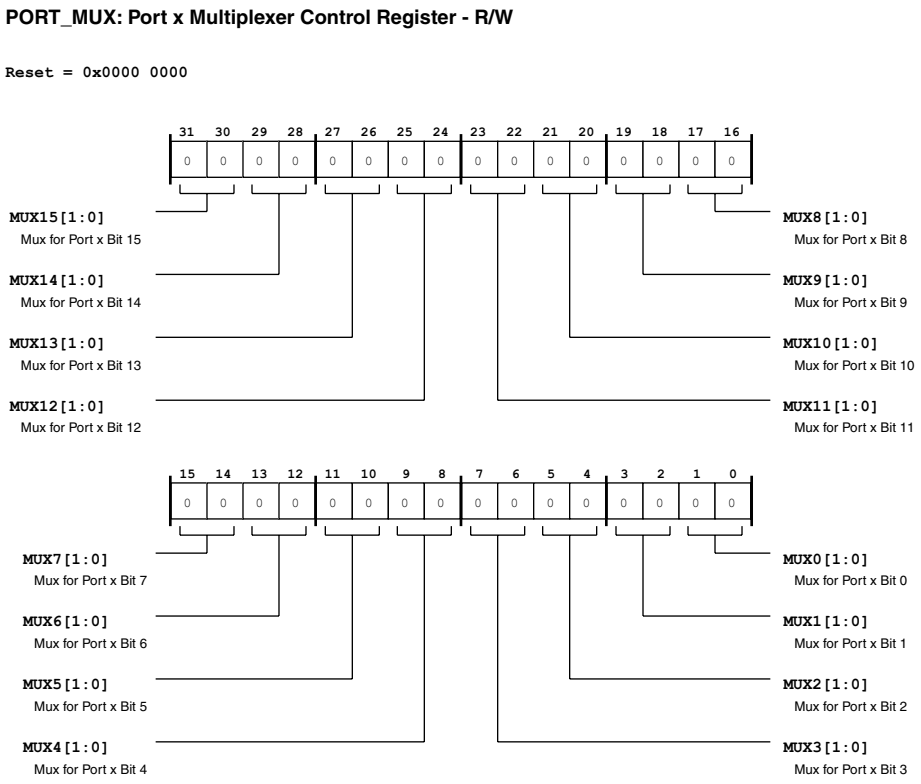


Figure 14-19: PORT_MUX Register Diagram

Table 14-21: PORT_MUX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:30 (R/W)	MUX15	Mux for Port x Bit 15. Multiplexer control for Port x bit 15.
29:28 (R/W)	MUX14	Mux for Port x Bit 14. Multiplexer control for Port x bit 14.
27:26 (R/W)	MUX13	Mux for Port x Bit 13. Multiplexer control for Port x bit 13.
25:24 (R/W)	MUX12	Mux for Port x Bit 12. Multiplexer control for Port x bit 12.
23:22 (R/W)	MUX11	Mux for Port x Bit 11. Multiplexer control for Port x bit 11.
21:20 (R/W)	MUX10	Mux for Port x Bit 10. Multiplexer control for Port x bit 10.

Table 14-21: PORT_MUX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
19:18 (R/W)	MUX9	Mux for Port x Bit 9. Multiplexer control for Port x bit 9.
17:16 (R/W)	MUX8	Mux for Port x Bit 8. Multiplexer control for Port x bit 8.
15:14 (R/W)	MUX7	Mux for Port x Bit 7. Multiplexer control for Port x bit 7.
13:12 (R/W)	MUX6	Mux for Port x Bit 6. Multiplexer control for Port x bit 6.
11:10 (R/W)	MUX5	Mux for Port x Bit 5. Multiplexer control for Port x bit 5.
9:8 (R/W)	MUX4	Mux for Port x Bit 4. Multiplexer control for Port x bit 4.
7:6 (R/W)	MUX3	Mux for Port x Bit 3. Multiplexer control for Port x bit 3.
5:4 (R/W)	MUX2	Mux for Port x Bit 2. Multiplexer control for Port x bit 2.
3:2 (R/W)	MUX1	Mux for Port x Bit 1. Multiplexer control for Port x bit 1.
1:0 (R/W)	MUX0	Mux for Port x Bit 0. Multiplexer control for Port x bit 0.

Port x GPIO Input Enable 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.

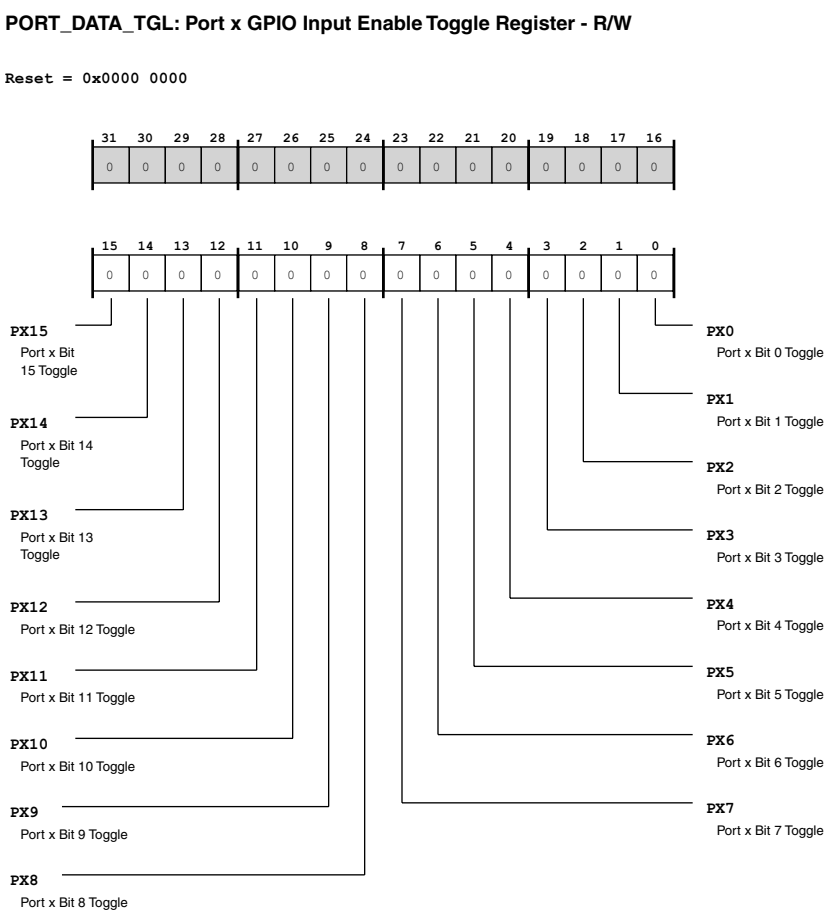


Figure 14-20: PORT_DATA_TGL Register Diagram

Table 14-22: PORT_DATA_TGL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1A)	PX15	Port x Bit 15 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
14 (R/W1A)	PX14	Port x Bit 14 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.

Table 14-22: PORT_DATA_TGL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1A)	PX13	Port x Bit 13 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
12 (R/W1A)	PX12	Port x Bit 12 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
11 (R/W1A)	PX11	Port x Bit 11 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
10 (R/W1A)	PX10	Port x Bit 10 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
9 (R/W1A)	PX9	Port x Bit 9 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
8 (R/W1A)	PX8	Port x Bit 8 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
7 (R/W1A)	PX7	Port x Bit 7 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
6 (R/W1A)	PX6	Port x Bit 6 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.

Table 14-22: PORT_DATA_TGL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1A)	PX5	Port x Bit 5 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
4 (R/W1A)	PX4	Port x Bit 4 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
3 (R/W1A)	PX3	Port x Bit 3 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
2 (R/W1A)	PX2	Port x Bit 2 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
1 (R/W1A)	PX1	Port x Bit 1 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.
0 (R/W1A)	PX0	Port x Bit 0 Toggle.	
		0	No Effect
		1	Toggle Bit Set to toggle output GPIO bit/pin state.

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.

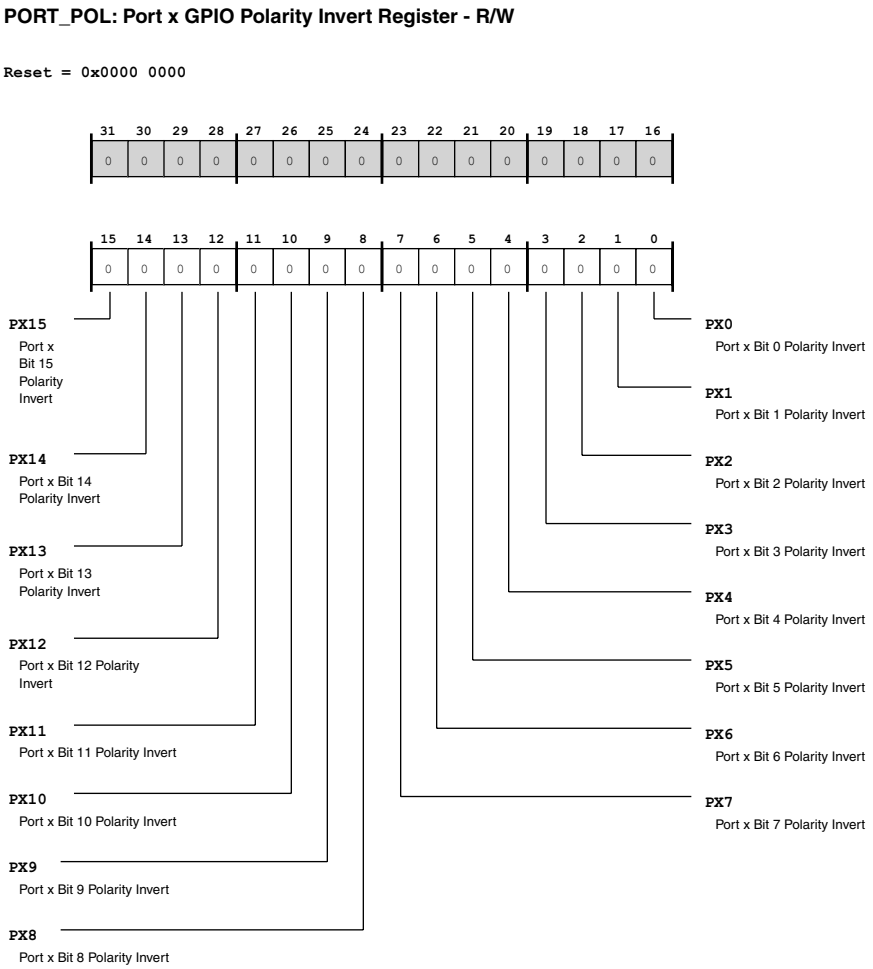


Figure 14-21: PORT_POL Register Diagram

Table 14-23: PORT_POL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.

Table 14-23: PORT_POL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	PX14	Port x Bit 14 Polarity Invert.	
		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.	
		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.	
		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.	
		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.	
		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.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.

Table 14-23: PORT_POL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	PX8	Port x Bit 8 Polarity Invert.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.
7 (R/W)	PX7	Port x Bit 7 Polarity Invert.	
		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.	
		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.	
		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.	
		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.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.

Table 14-23: PORT_POL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	PX2	Port x Bit 2 Polarity Invert.	
		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.	
		0	No Invert GPIO is active high or rising edge sensitive.
		1	Invert GPIO is active low or falling edge sensitive.
0 (R/W)	PX0	Port x Bit 0 Polarity Invert.	
		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 Set Register

The PORT_POL_SET register enables polarity inversion for GPIO pins. For more information, see the PORT_POL register description.

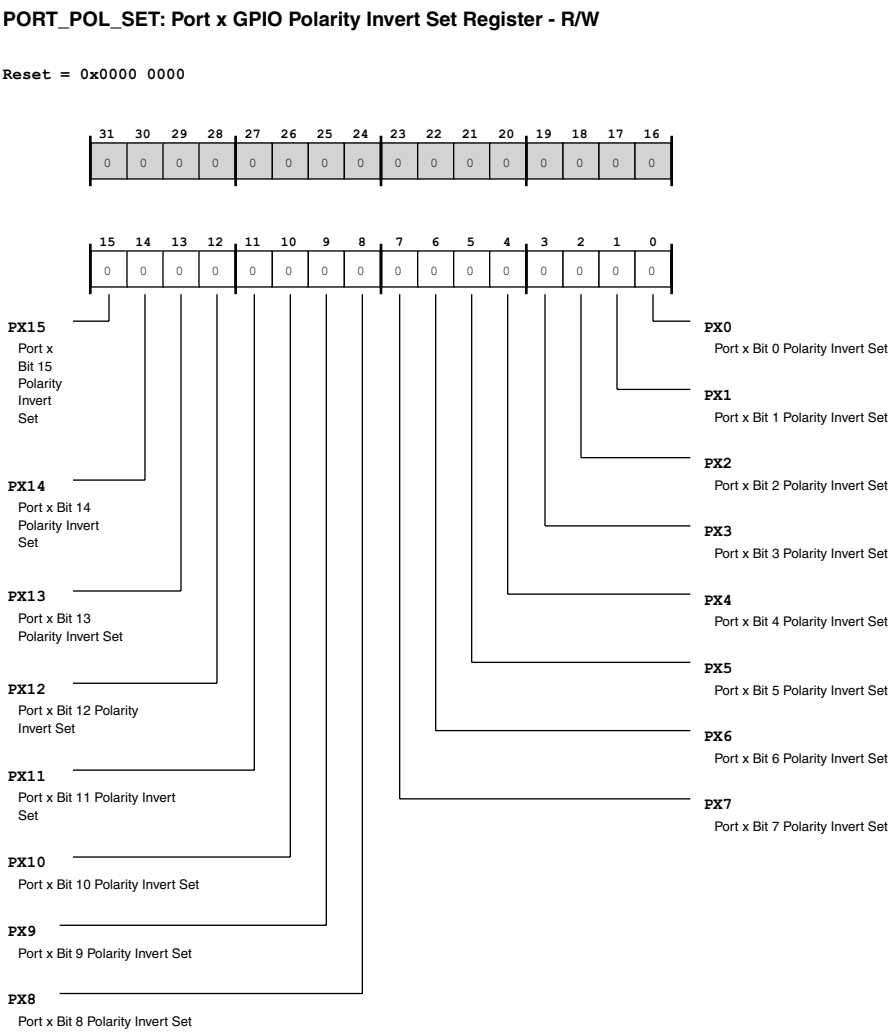


Figure 14-22: PORT_POL_SET Register Diagram

Table 14-24: PORT_POL_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
14 (R/W1S)	PX14	Port x Bit 14 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.

Table 14-24: PORT_POL_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1S)	PX13	Port x Bit 13 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
12 (R/W1S)	PX12	Port x Bit 12 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
11 (R/W1S)	PX11	Port x Bit 11 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
10 (R/W1S)	PX10	Port x Bit 10 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
9 (R/W1S)	PX9	Port x Bit 9 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
8 (R/W1S)	PX8	Port x Bit 8 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
7 (R/W1S)	PX7	Port x Bit 7 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
6 (R/W1S)	PX6	Port x Bit 6 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.

Table 14-24: PORT_POL_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1S)	PX5	Port x Bit 5 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
4 (R/W1S)	PX4	Port x Bit 4 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
3 (R/W1S)	PX3	Port x Bit 3 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
2 (R/W1S)	PX2	Port x Bit 2 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
1 (R/W1S)	PX1	Port x Bit 1 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.
0 (R/W1S)	PX0	Port x Bit 0 Polarity Invert Set.	
		0	No Effect
		1	Set Bit Set to enable GPIO pin polarity invert.

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.

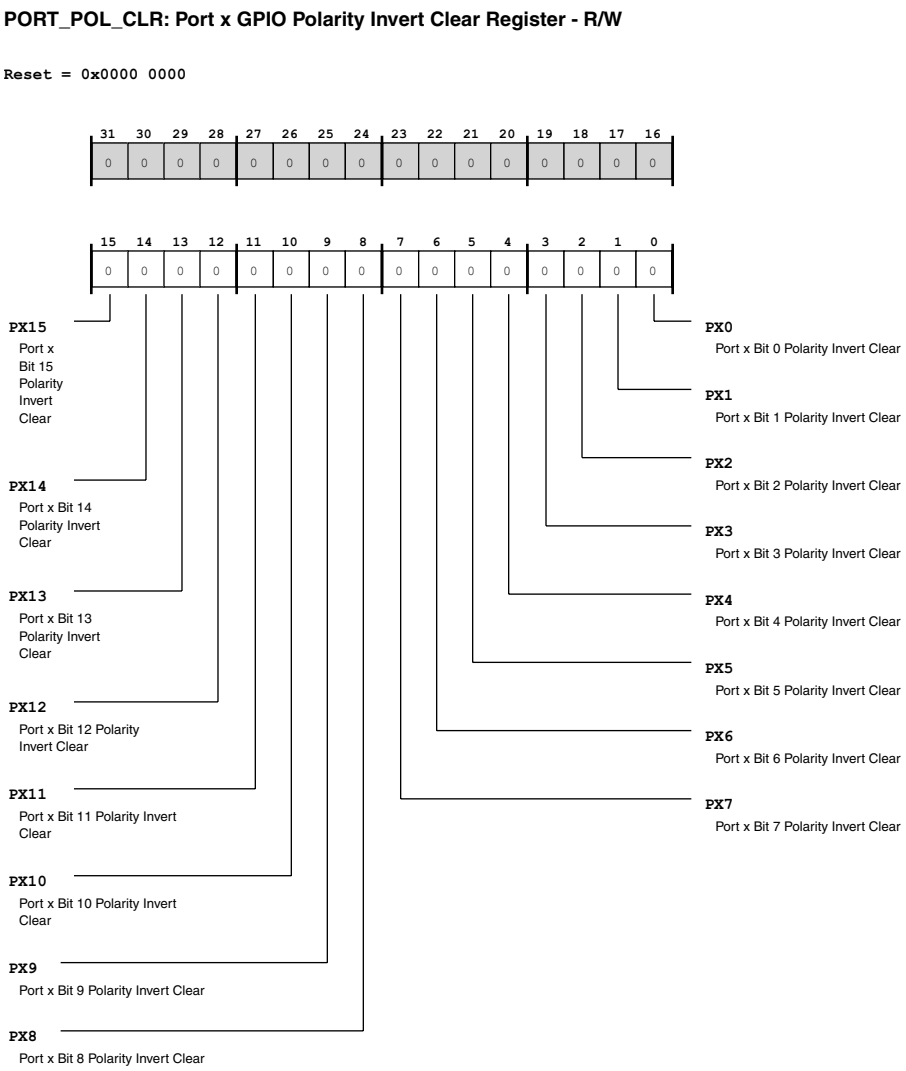


Figure 14-23: PORT_POL_CLR Register Diagram

Table 14-25: 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.

Table 14-25: PORT_POL_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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.
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.

Table 14-25: PORT_POL_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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.
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 Lock Register

The PORT_LOCK register enables (unlocks) or disables (locks) write access selectively for the PORT control registers.

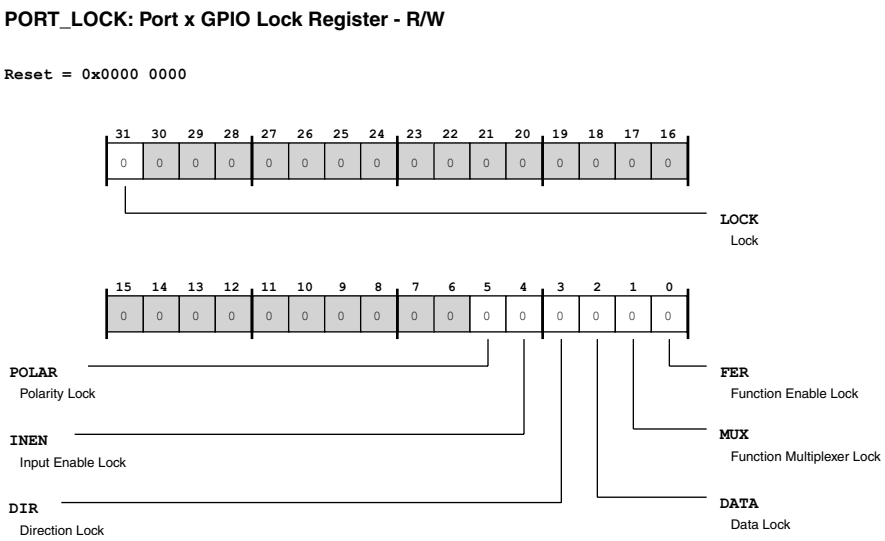


Figure 14-24: PORT_LOCK Register Diagram

Table 14-26: PORT_LOCK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the PORT_LOCK.LOCK bit is set, the PORT_LOCK register is read only (locked).	
		0	Unlock
		1	Lock
5 (R/W)	POLAR	Polarity Lock. The PORT_LOCK.POLAR disables write access to the PORT_POL, PORT_POL_SET, and PORT_POL_CLR registers.	
		0	Unlock POL
		1	Lock POL
4 (R/W)	INEN	Input Enable Lock. The PORT_LOCK.INEN disables write access to the PORT_INEN, PORT_INEN_SET, and PORT_INEN_CLR registers.	
		0	Unlock INEN
		1	Lock INEN
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	Lock DIR
		1	Unlock DIR

Table 14-26: PORT_LOCK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	DATA	Data 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

ADSP-BF60x PINT Register Descriptions

PINT (PINT) contains the following registers.

Table 14-27: ADSP-BF60x PINT Register List

Name	Description
PINT_MSK_SET	Pint Mask Set Register
PINT_MSK_CLR	Pint Mask Clear Register
PINT_REQ	Pint Request Register
PINT_ASSIGN	Pint Assign Register
PINT_EDGE_SET	Pint Edge Set Register
PINT_EDGE_CLR	Pint Edge Clear Register
PINT_INV_SET	Pint Invert Set Register
PINT_INV_CLR	Pint Invert Clear Register

Table 14-27: ADSP-BF60x PINT Register List (Continued)

Name	Description
PINT_PINSTATE	Pint Pinstate Register
PINT_LATCH	Pint Latch Register

Pint Mask Set Register

The PINT_MSK_SET register permits unmasking (enabling) of interrupts. Writing 1 to a bit in PINT_MSK_SET unmasks the corresponding pin interrupt.

PINT_MSK_SET: Pint Mask Set Register - R/W

Reset = 0x0000 0000

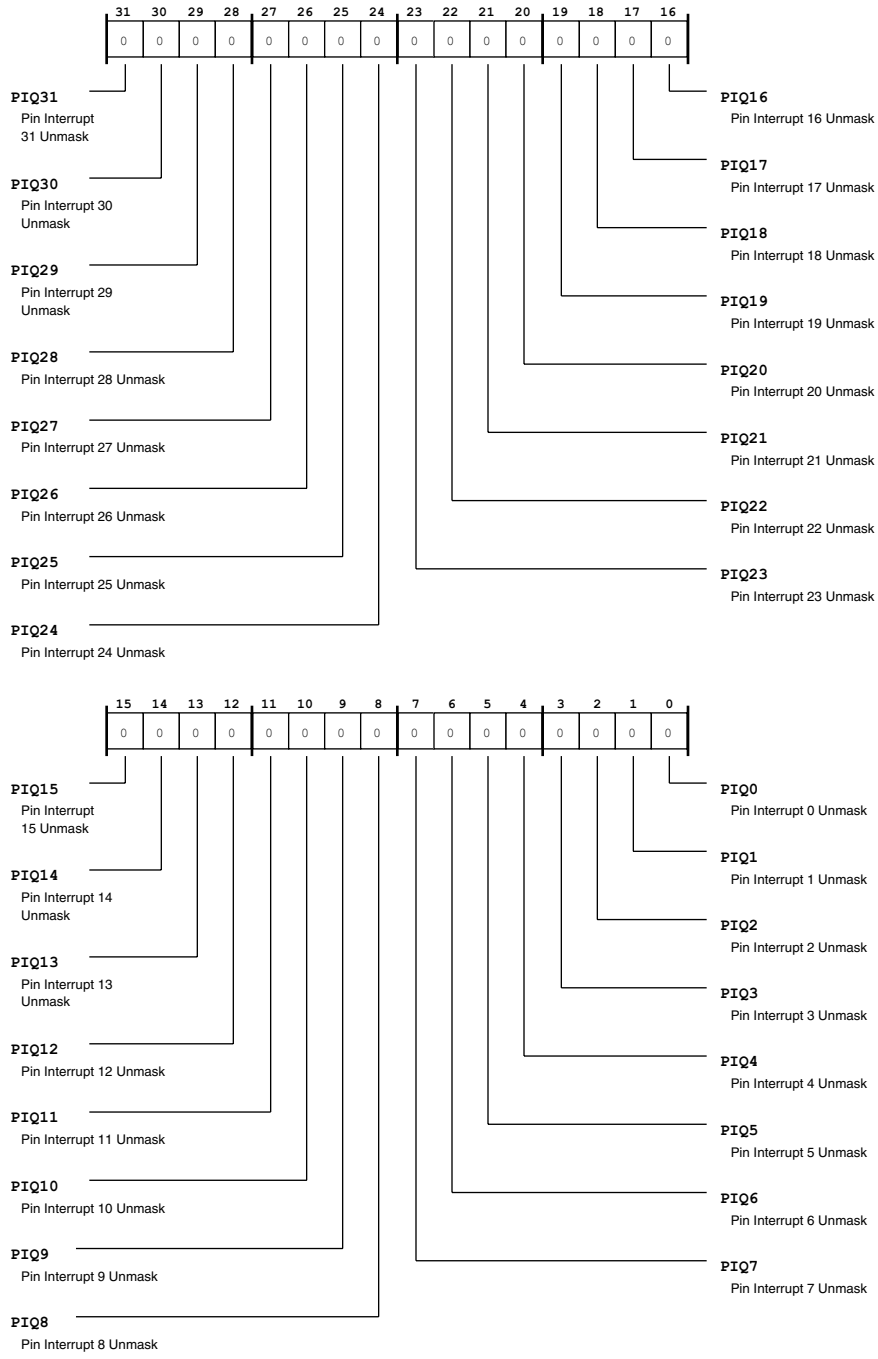


Figure 14-25: PINT_MSK_SET Register Diagram

Table 14-28: PINT_MSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	PIQ31	Pin Interrupt 31 Unmask. Set to enable interrupt.
30 (R/W1S)	PIQ30	Pin Interrupt 30 Unmask. Set to enable interrupt.
29 (R/W1S)	PIQ29	Pin Interrupt 29 Unmask. Set to enable interrupt.
28 (R/W1S)	PIQ28	Pin Interrupt 28 Unmask. Set to enable interrupt.
27 (R/W1S)	PIQ27	Pin Interrupt 27 Unmask. Set to enable interrupt.
26 (R/W1S)	PIQ26	Pin Interrupt 26 Unmask. Set to enable interrupt.
25 (R/W1S)	PIQ25	Pin Interrupt 25 Unmask. Set to enable interrupt.
24 (R/W1S)	PIQ24	Pin Interrupt 24 Unmask. Set to enable interrupt.
23 (R/W1S)	PIQ23	Pin Interrupt 23 Unmask. Set to enable interrupt.
22 (R/W1S)	PIQ22	Pin Interrupt 22 Unmask. Set to enable interrupt.
21 (R/W1S)	PIQ21	Pin Interrupt 21 Unmask. Set to enable interrupt.
20 (R/W1S)	PIQ20	Pin Interrupt 20 Unmask. Set to enable interrupt.
19 (R/W1S)	PIQ19	Pin Interrupt 19 Unmask. Set to enable interrupt.
18 (R/W1S)	PIQ18	Pin Interrupt 18 Unmask. Set to enable interrupt.
17 (R/W1S)	PIQ17	Pin Interrupt 17 Unmask. Set to enable interrupt.
16 (R/W1S)	PIQ16	Pin Interrupt 16 Unmask. Set to enable interrupt.
15 (R/W1S)	PIQ15	Pin Interrupt 15 Unmask. Set to enable interrupt.

Table 14-28: PINT_MSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1S)	PIQ14	Pin Interrupt 14 Unmask. Set to enable interrupt.
13 (R/W1S)	PIQ13	Pin Interrupt 13 Unmask. Set to enable interrupt.
12 (R/W1S)	PIQ12	Pin Interrupt 12 Unmask. Set to enable interrupt.
11 (R/W1S)	PIQ11	Pin Interrupt 11 Unmask. Set to enable interrupt.
10 (R/W1S)	PIQ10	Pin Interrupt 10 Unmask. Set to enable interrupt.
9 (R/W1S)	PIQ9	Pin Interrupt 9 Unmask. Set to enable interrupt.
8 (R/W1S)	PIQ8	Pin Interrupt 8 Unmask. Set to enable interrupt.
7 (R/W1S)	PIQ7	Pin Interrupt 7 Unmask. Set to enable interrupt.
6 (R/W1S)	PIQ6	Pin Interrupt 6 Unmask. Set to enable interrupt.
5 (R/W1S)	PIQ5	Pin Interrupt 5 Unmask. Set to enable interrupt.
4 (R/W1S)	PIQ4	Pin Interrupt 4 Unmask. Set to enable interrupt.
3 (R/W1S)	PIQ3	Pin Interrupt 3 Unmask. Set to enable interrupt.
2 (R/W1S)	PIQ2	Pin Interrupt 2 Unmask. Set to enable interrupt.
1 (R/W1S)	PIQ1	Pin Interrupt 1 Unmask. Set to enable interrupt.
0 (R/W1S)	PIQ0	Pin Interrupt 0 Unmask. Set to enable interrupt.

Pint Mask Clear Register

The PINT_MSK_CLR register permits masking (disabling) of interrupts. Writing 1 to a bit in PINT_MSK_CLR masks the corresponding pin interrupt.

PINT_MSK_CLR: Pint Mask Clear Register - R/W

Reset = 0x0000 0000

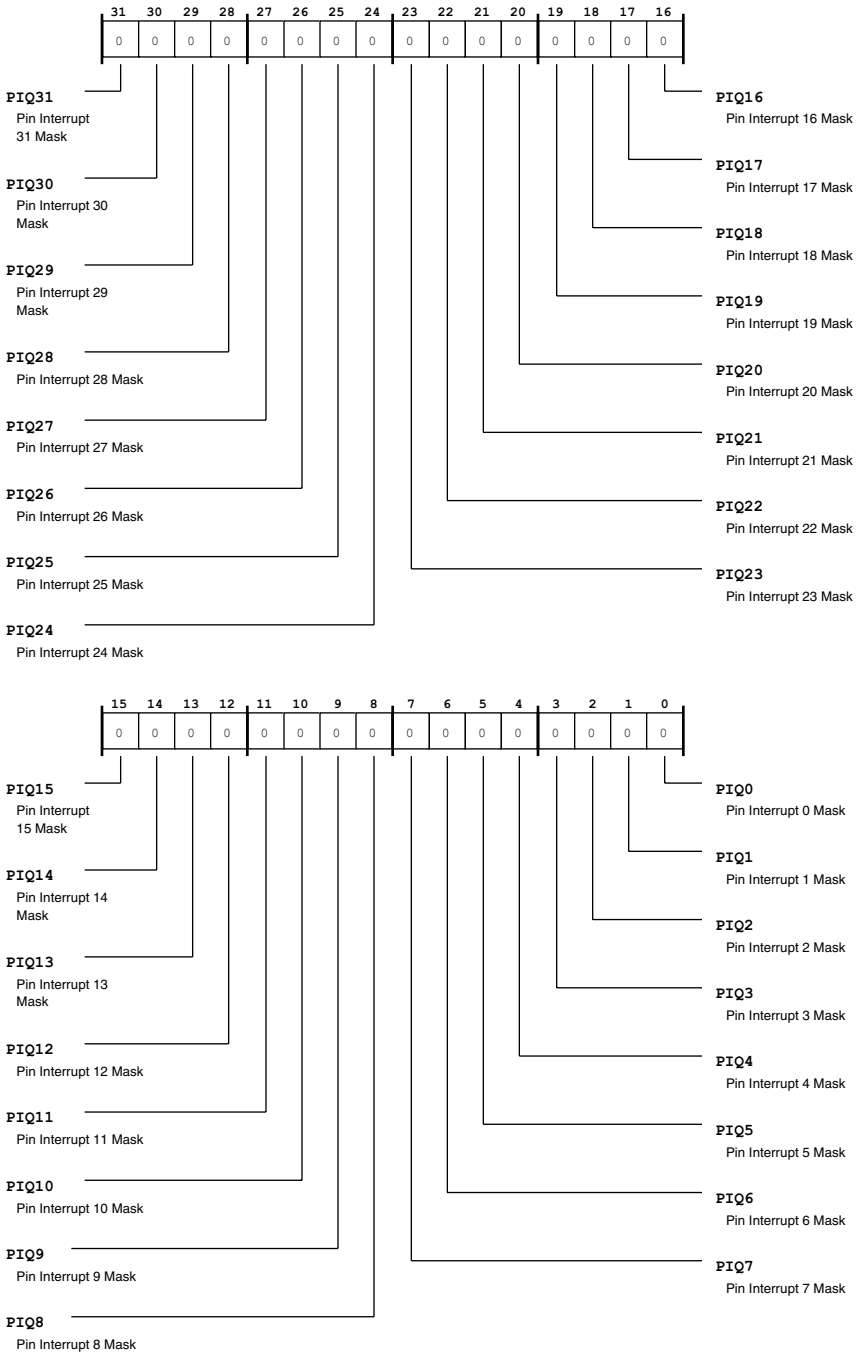


Figure 14-26: PINT_MSK_CLR Register Diagram

Table 14-29: PINT_MSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Mask. Set to disable interrupt.
30 (R/W1C)	PIQ30	Pin Interrupt 30 Mask. Set to disable interrupt.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Mask. Set to disable interrupt.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Mask. Set to disable interrupt.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Mask. Set to disable interrupt.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Mask. Set to disable interrupt.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Mask. Set to disable interrupt.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Mask. Set to disable interrupt.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Mask. Set to disable interrupt.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Mask. Set to disable interrupt.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Mask. Set to disable interrupt.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Mask. Set to disable interrupt.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Mask. Set to disable interrupt.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Mask. Set to disable interrupt.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Mask. Set to disable interrupt.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Mask. Set to disable interrupt.
15 (R/W1C)	PIQ15	Pin Interrupt 15 Mask. Set to disable interrupt.

Table 14-29: PINT_MSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1C)	PIQ14	Pin Interrupt 14 Mask. Set to disable interrupt.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Mask. Set to disable interrupt.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Mask. Set to disable interrupt.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Mask. Set to disable interrupt.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Mask. Set to disable interrupt.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Mask. Set to disable interrupt.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Mask. Set to disable interrupt.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Mask. Set to disable interrupt.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Mask. Set to disable interrupt.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Mask. Set to disable interrupt.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Mask. Set to disable interrupt.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Mask. Set to disable interrupt.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Mask. Set to disable interrupt.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Mask. Set to disable interrupt.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Mask. Set to disable interrupt.

Pint Request Register

The PINT_REQ register indicates 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.

PINT_REQ: Pint Request Register - R/W

Reset = 0x0000 0000

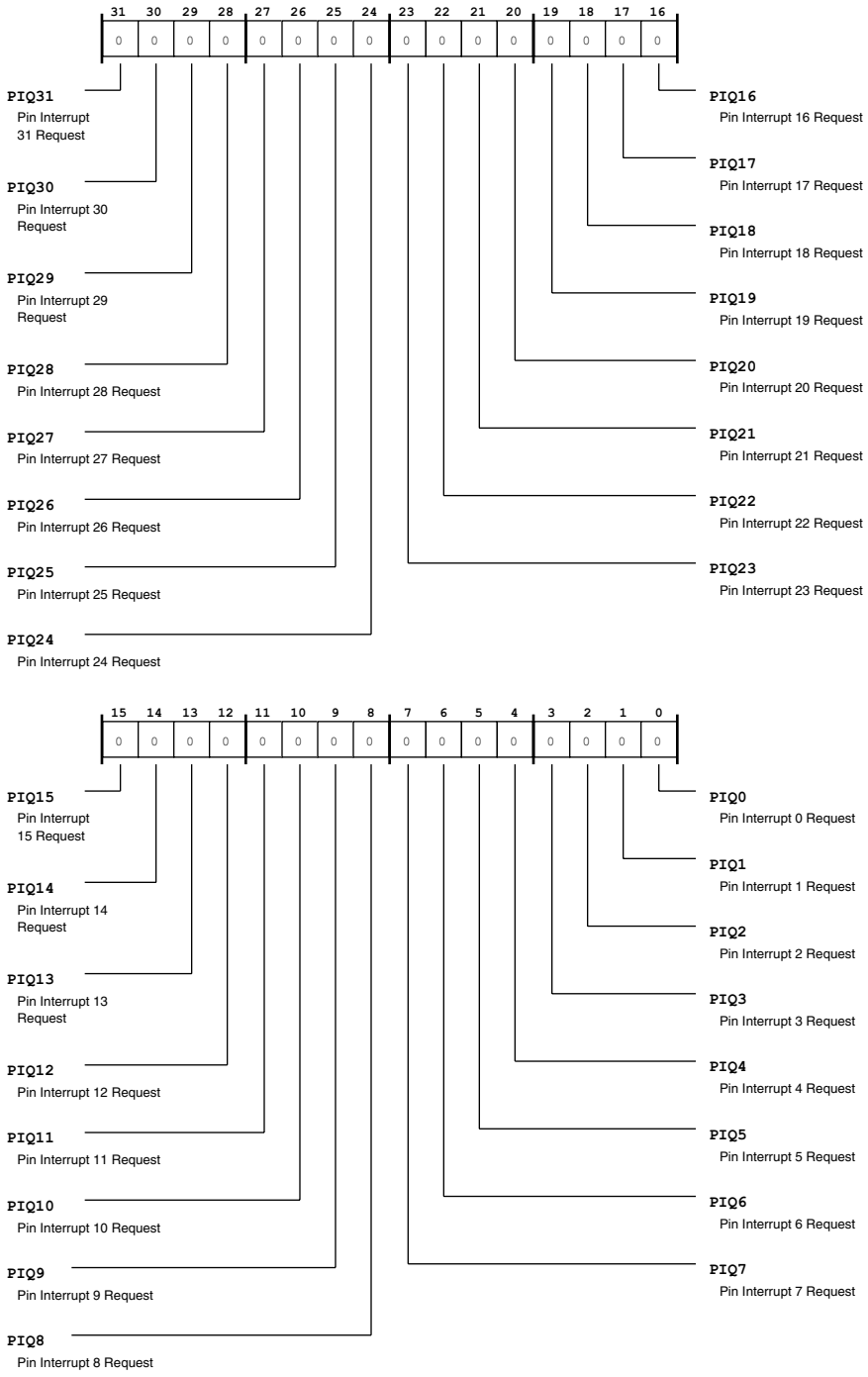


Figure 14-27: PINT_REQ Register Diagram

Table 14-30: PINT_REQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Request. If set, request pending.
30 (R/W1C)	PIQ30	Pin Interrupt 30 Request. If set, request pending.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Request. If set, request pending.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Request. If set, request pending.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Request. If set, request pending.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Request. If set, request pending.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Request. If set, request pending.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Request. If set, request pending.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Request. If set, request pending.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Request. If set, request pending.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Request. If set, request pending.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Request. If set, request pending.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Request. If set, request pending.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Request. If set, request pending.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Request. If set, request pending.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Request. If set, request pending.
15 (R/W1C)	PIQ15	Pin Interrupt 15 Request. If set, request pending.

Table 14-30: PINT_REQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1C)	PIQ14	Pin Interrupt 14 Request. If set, request pending.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Request. If set, request pending.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Request. If set, request pending.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Request. If set, request pending.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Request. If set, request pending.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Request. If set, request pending.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Request. If set, request pending.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Request. If set, request pending.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Request. If set, request pending.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Request. If set, request pending.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Request. If set, request pending.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Request. If set, request pending.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Request. If set, request pending.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Request. If set, request pending.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Request. If set, request pending.

Pint Assign Register

The PINT_ASSIGN register controls the pin-to-interrupt 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, and the upper half units can be forwarded to either byte 1 or byte 3 of the PINT block, without further restrictions.

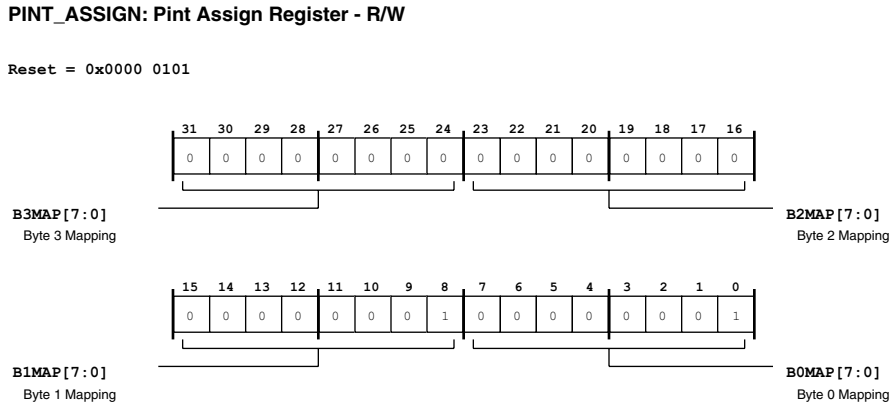


Figure 14-28: PINT_ASSIGN Register Diagram

Table 14-31: PINT_ASSIGN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:24 (R/W)	B3MAP	Byte 3 Mapping.	
		0	B3MAP_PAH Byte 3 = PA.H
		1	B3MAP_PBH Byte 3 = PB.H
23:16 (R/W)	B2MAP	Byte 2 Mapping.	
		0	B2MAP_PAL Byte 2 = PA.L
		1	B2MAP_PBL Byte 2 = PB.L
15:8 (R/W)	B1MAP	Byte 1 Mapping.	
		0	B1MAP_PAH Byte 1 = PA.H
		1	B1MAP_PBH Byte 1 = PB.H

Table 14-31: PINT_ASSIGN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7:0 (R/W)	B0MAP	Byte 0 Mapping.	
		0	B0MAP_PAL Byte 0 = PA.L
		1	B0MAP_PBL Byte 0 = PB.L

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.

PINT_EDGE_SET: Pint Edge Set Register - R/W

Reset = 0x0000 0000

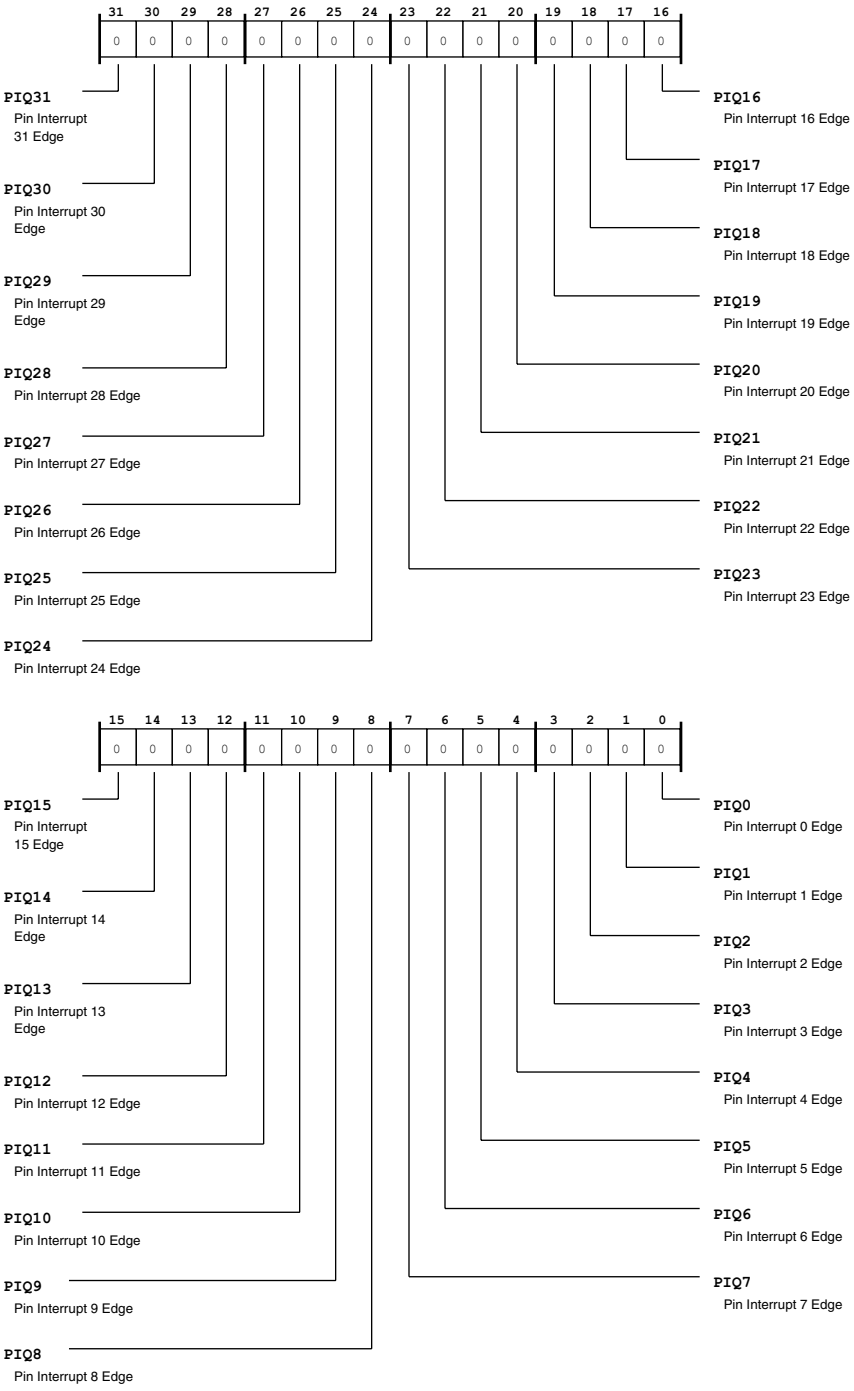


Figure 14-29: PINT_EDGE_SET Register Diagram

Table 14-32: PINT_EDGE_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	PIQ31	Pin Interrupt 31 Edge. Set to enable edge sensitivity.
30 (R/W1S)	PIQ30	Pin Interrupt 30 Edge. Set to enable edge sensitivity.
29 (R/W1S)	PIQ29	Pin Interrupt 29 Edge. Set to enable edge sensitivity.
28 (R/W1S)	PIQ28	Pin Interrupt 28 Edge. Set to enable edge sensitivity.
27 (R/W1S)	PIQ27	Pin Interrupt 27 Edge. Set to enable edge sensitivity.
26 (R/W1S)	PIQ26	Pin Interrupt 26 Edge. Set to enable edge sensitivity.
25 (R/W1S)	PIQ25	Pin Interrupt 25 Edge. Set to enable edge sensitivity.
24 (R/W1S)	PIQ24	Pin Interrupt 24 Edge. Set to enable edge sensitivity.
23 (R/W1S)	PIQ23	Pin Interrupt 23 Edge. Set to enable edge sensitivity.
22 (R/W1S)	PIQ22	Pin Interrupt 22 Edge. Set to enable edge sensitivity.
21 (R/W1S)	PIQ21	Pin Interrupt 21 Edge. Set to enable edge sensitivity.
20 (R/W1S)	PIQ20	Pin Interrupt 20 Edge. Set to enable edge sensitivity.
19 (R/W1S)	PIQ19	Pin Interrupt 19 Edge. Set to enable edge sensitivity.
18 (R/W1S)	PIQ18	Pin Interrupt 18 Edge. Set to enable edge sensitivity.
17 (R/W1S)	PIQ17	Pin Interrupt 17 Edge. Set to enable edge sensitivity.
16 (R/W1S)	PIQ16	Pin Interrupt 16 Edge. Set to enable edge sensitivity.
15 (R/W1S)	PIQ15	Pin Interrupt 15 Edge. Set to enable edge sensitivity.

Table 14-32: PINT_EDGE_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1S)	PIQ14	Pin Interrupt 14 Edge. Set to enable edge sensitivity.
13 (R/W1S)	PIQ13	Pin Interrupt 13 Edge. Set to enable edge sensitivity.
12 (R/W1S)	PIQ12	Pin Interrupt 12 Edge. Set to enable edge sensitivity.
11 (R/W1S)	PIQ11	Pin Interrupt 11 Edge. Set to enable edge sensitivity.
10 (R/W1S)	PIQ10	Pin Interrupt 10 Edge. Set to enable edge sensitivity.
9 (R/W1S)	PIQ9	Pin Interrupt 9 Edge. Set to enable edge sensitivity.
8 (R/W1S)	PIQ8	Pin Interrupt 8 Edge. Set to enable edge sensitivity.
7 (R/W1S)	PIQ7	Pin Interrupt 7 Edge. Set to enable edge sensitivity.
6 (R/W1S)	PIQ6	Pin Interrupt 6 Edge. Set to enable edge sensitivity.
5 (R/W1S)	PIQ5	Pin Interrupt 5 Edge. Set to enable edge sensitivity.
4 (R/W1S)	PIQ4	Pin Interrupt 4 Edge. Set to enable edge sensitivity.
3 (R/W1S)	PIQ3	Pin Interrupt 3 Edge. Set to enable edge sensitivity.
2 (R/W1S)	PIQ2	Pin Interrupt 2 Edge. Set to enable edge sensitivity.
1 (R/W1S)	PIQ1	Pin Interrupt 1 Edge. Set to enable edge sensitivity.
0 (R/W1S)	PIQ0	Pin Interrupt 0 Edge. Set to enable edge sensitivity.

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.

PINT_EDGE_CLR: Pint Edge Clear Register - R/W

Reset = 0x0000 0000

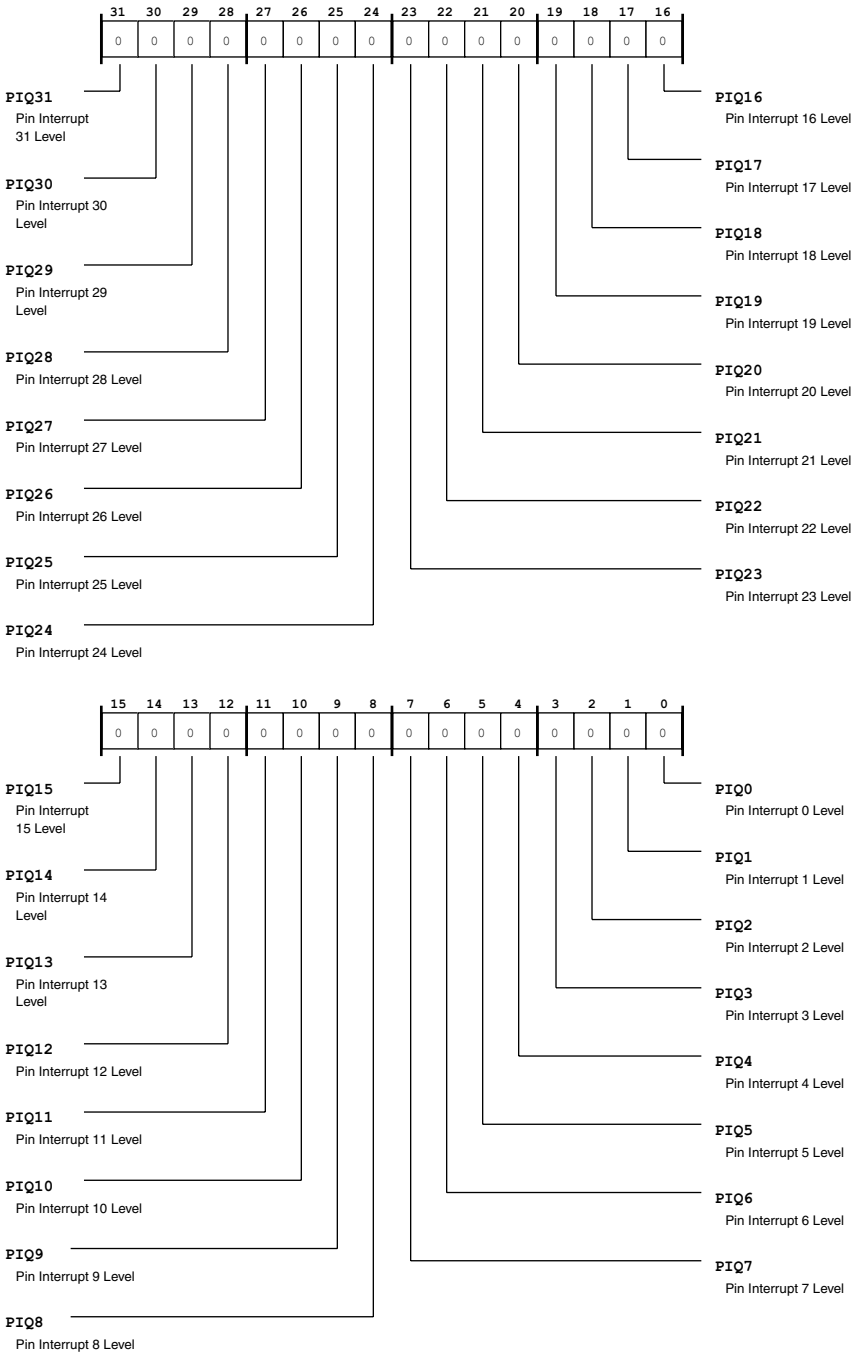


Figure 14-30: PINT_EDGE_CLR Register Diagram

Table 14-33: PINT_EDGE_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Level. Set to enable level sensitivity.
30 (R/W1C)	PIQ30	Pin Interrupt 30 Level. Set to enable level sensitivity.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Level. Set to enable level sensitivity.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Level. Set to enable level sensitivity.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Level. Set to enable level sensitivity.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Level. Set to enable level sensitivity.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Level. Set to enable level sensitivity.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Level. Set to enable level sensitivity.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Level. Set to enable level sensitivity.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Level. Set to enable level sensitivity.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Level. Set to enable level sensitivity.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Level. Set to enable level sensitivity.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Level. Set to enable level sensitivity.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Level. Set to enable level sensitivity.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Level. Set to enable level sensitivity.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Level. Set to enable level sensitivity.
15 (R/W1C)	PIQ15	Pin Interrupt 15 Level. Set to enable level sensitivity.

Table 14-33: PINT_EDGE_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1C)	PIQ14	Pin Interrupt 14 Level. Set to enable level sensitivity.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Level. Set to enable level sensitivity.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Level. Set to enable level sensitivity.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Level. Set to enable level sensitivity.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Level. Set to enable level sensitivity.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Level. Set to enable level sensitivity.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Level. Set to enable level sensitivity.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Level. Set to enable level sensitivity.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Level. Set to enable level sensitivity.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Level. Set to enable level sensitivity.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Level. Set to enable level sensitivity.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Level. Set to enable level sensitivity.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Level. Set to enable level sensitivity.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Level. Set to enable level sensitivity.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Level. Set to enable level sensitivity.

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.

PINT_INV_SET: Pint Invert Set Register - R/W

Reset = 0x0000 0000

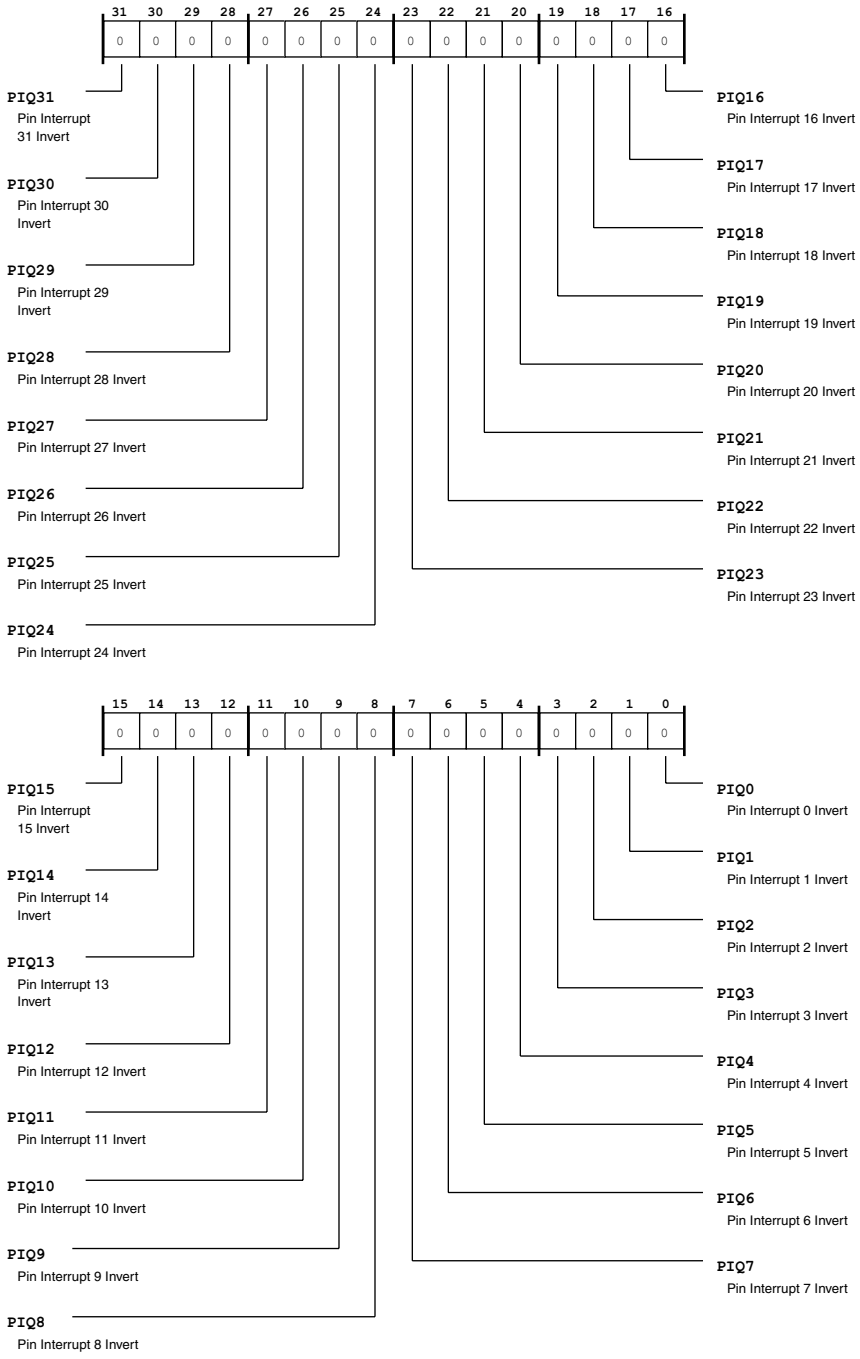


Figure 14-31: PINT_INV_SET Register Diagram

Table 14-34: PINT_INV_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	PIQ31	Pin Interrupt 31 Invert. Set to enable inverted input.
30 (R/W1S)	PIQ30	Pin Interrupt 30 Invert. Set to enable inverted input.
29 (R/W1S)	PIQ29	Pin Interrupt 29 Invert. Set to enable inverted input.
28 (R/W1S)	PIQ28	Pin Interrupt 28 Invert. Set to enable inverted input.
27 (R/W1S)	PIQ27	Pin Interrupt 27 Invert. Set to enable inverted input.
26 (R/W1S)	PIQ26	Pin Interrupt 26 Invert. Set to enable inverted input.
25 (R/W1S)	PIQ25	Pin Interrupt 25 Invert. Set to enable inverted input.
24 (R/W1S)	PIQ24	Pin Interrupt 24 Invert. Set to enable inverted input.
23 (R/W1S)	PIQ23	Pin Interrupt 23 Invert. Set to enable inverted input.
22 (R/W1S)	PIQ22	Pin Interrupt 22 Invert. Set to enable inverted input.
21 (R/W1S)	PIQ21	Pin Interrupt 21 Invert. Set to enable inverted input.
20 (R/W1S)	PIQ20	Pin Interrupt 20 Invert. Set to enable inverted input.
19 (R/W1S)	PIQ19	Pin Interrupt 19 Invert. Set to enable inverted input.
18 (R/W1S)	PIQ18	Pin Interrupt 18 Invert. Set to enable inverted input.
17 (R/W1S)	PIQ17	Pin Interrupt 17 Invert. Set to enable inverted input.
16 (R/W1S)	PIQ16	Pin Interrupt 16 Invert. Set to enable inverted input.
15 (R/W1S)	PIQ15	Pin Interrupt 15 Invert. Set to enable inverted input.

Table 14-34: PINT_INV_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1S)	PIQ14	Pin Interrupt 14 Invert. Set to enable inverted input.
13 (R/W1S)	PIQ13	Pin Interrupt 13 Invert. Set to enable inverted input.
12 (R/W1S)	PIQ12	Pin Interrupt 12 Invert. Set to enable inverted input.
11 (R/W1S)	PIQ11	Pin Interrupt 11 Invert. Set to enable inverted input.
10 (R/W1S)	PIQ10	Pin Interrupt 10 Invert. Set to enable inverted input.
9 (R/W1S)	PIQ9	Pin Interrupt 9 Invert. Set to enable inverted input.
8 (R/W1S)	PIQ8	Pin Interrupt 8 Invert. Set to enable inverted input.
7 (R/W1S)	PIQ7	Pin Interrupt 7 Invert. Set to enable inverted input.
6 (R/W1S)	PIQ6	Pin Interrupt 6 Invert. Set to enable inverted input.
5 (R/W1S)	PIQ5	Pin Interrupt 5 Invert. Set to enable inverted input.
4 (R/W1S)	PIQ4	Pin Interrupt 4 Invert. Set to enable inverted input.
3 (R/W1S)	PIQ3	Pin Interrupt 3 Invert. Set to enable inverted input.
2 (R/W1S)	PIQ2	Pin Interrupt 2 Invert. Set to enable inverted input.
1 (R/W1S)	PIQ1	Pin Interrupt 1 Invert. Set to enable inverted input.
0 (R/W1S)	PIQ0	Pin Interrupt 0 Invert. Set to enable inverted input.

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.

PINT_INV_CLR: Pint Invert Clear Register - R/W

Reset = 0x0000 0000

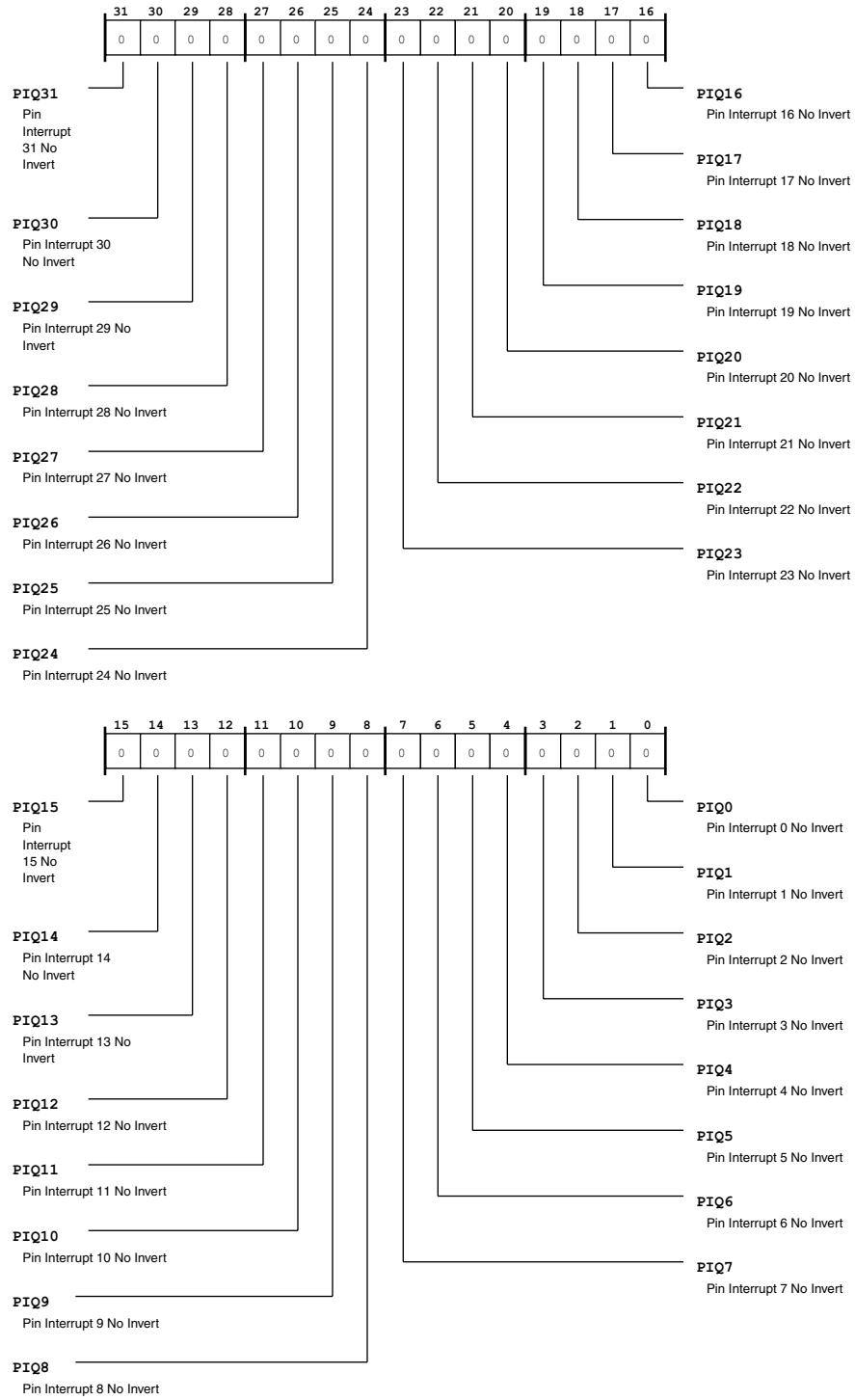


Figure 14-32: PINT_INV_CLR Register Diagram

Table 14-35: PINT_INV_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 No Invert. Set to disable inverted input.
30 (R/W1C)	PIQ30	Pin Interrupt 30 No Invert. Set to disable inverted input.
29 (R/W1C)	PIQ29	Pin Interrupt 29 No Invert. Set to disable inverted input.
28 (R/W1C)	PIQ28	Pin Interrupt 28 No Invert. Set to disable inverted input.
27 (R/W1C)	PIQ27	Pin Interrupt 27 No Invert. Set to disable inverted input.
26 (R/W1C)	PIQ26	Pin Interrupt 26 No Invert. Set to disable inverted input.
25 (R/W1C)	PIQ25	Pin Interrupt 25 No Invert. Set to disable inverted input.
24 (R/W1C)	PIQ24	Pin Interrupt 24 No Invert. Set to disable inverted input.
23 (R/W1C)	PIQ23	Pin Interrupt 23 No Invert. Set to disable inverted input.
22 (R/W1C)	PIQ22	Pin Interrupt 22 No Invert. Set to disable inverted input.
21 (R/W1C)	PIQ21	Pin Interrupt 21 No Invert. Set to disable inverted input.
20 (R/W1C)	PIQ20	Pin Interrupt 20 No Invert. Set to disable inverted input.
19 (R/W1C)	PIQ19	Pin Interrupt 19 No Invert. Set to disable inverted input.
18 (R/W1C)	PIQ18	Pin Interrupt 18 No Invert. Set to disable inverted input.
17 (R/W1C)	PIQ17	Pin Interrupt 17 No Invert. Set to disable inverted input.
16 (R/W1C)	PIQ16	Pin Interrupt 16 No Invert. Set to disable inverted input.
15 (R/W1C)	PIQ15	Pin Interrupt 15 No Invert. Set to disable inverted input.

Table 14-35: PINT_INV_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1C)	PIQ14	Pin Interrupt 14 No Invert. Set to disable inverted input.
13 (R/W1C)	PIQ13	Pin Interrupt 13 No Invert. Set to disable inverted input.
12 (R/W1C)	PIQ12	Pin Interrupt 12 No Invert. Set to disable inverted input.
11 (R/W1C)	PIQ11	Pin Interrupt 11 No Invert. Set to disable inverted input.
10 (R/W1C)	PIQ10	Pin Interrupt 10 No Invert. Set to disable inverted input.
9 (R/W1C)	PIQ9	Pin Interrupt 9 No Invert. Set to disable inverted input.
8 (R/W1C)	PIQ8	Pin Interrupt 8 No Invert. Set to disable inverted input.
7 (R/W1C)	PIQ7	Pin Interrupt 7 No Invert. Set to disable inverted input.
6 (R/W1C)	PIQ6	Pin Interrupt 6 No Invert. Set to disable inverted input.
5 (R/W1C)	PIQ5	Pin Interrupt 5 No Invert. Set to disable inverted input.
4 (R/W1C)	PIQ4	Pin Interrupt 4 No Invert. Set to disable inverted input.
3 (R/W1C)	PIQ3	Pin Interrupt 3 No Invert. Set to disable inverted input.
2 (R/W1C)	PIQ2	Pin Interrupt 2 No Invert. Set to disable inverted input.
1 (R/W1C)	PIQ1	Pin Interrupt 1 No Invert. Set to disable inverted input.
0 (R/W1C)	PIQ0	Pin Interrupt 0 No Invert. Set to disable inverted input.

Pint Pinstate 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_SETregister. The inverter can be enabled on a 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.

PINT_PINSTATE: Pint Pinstate Register - R/WE

Reset = 0x0000 0000

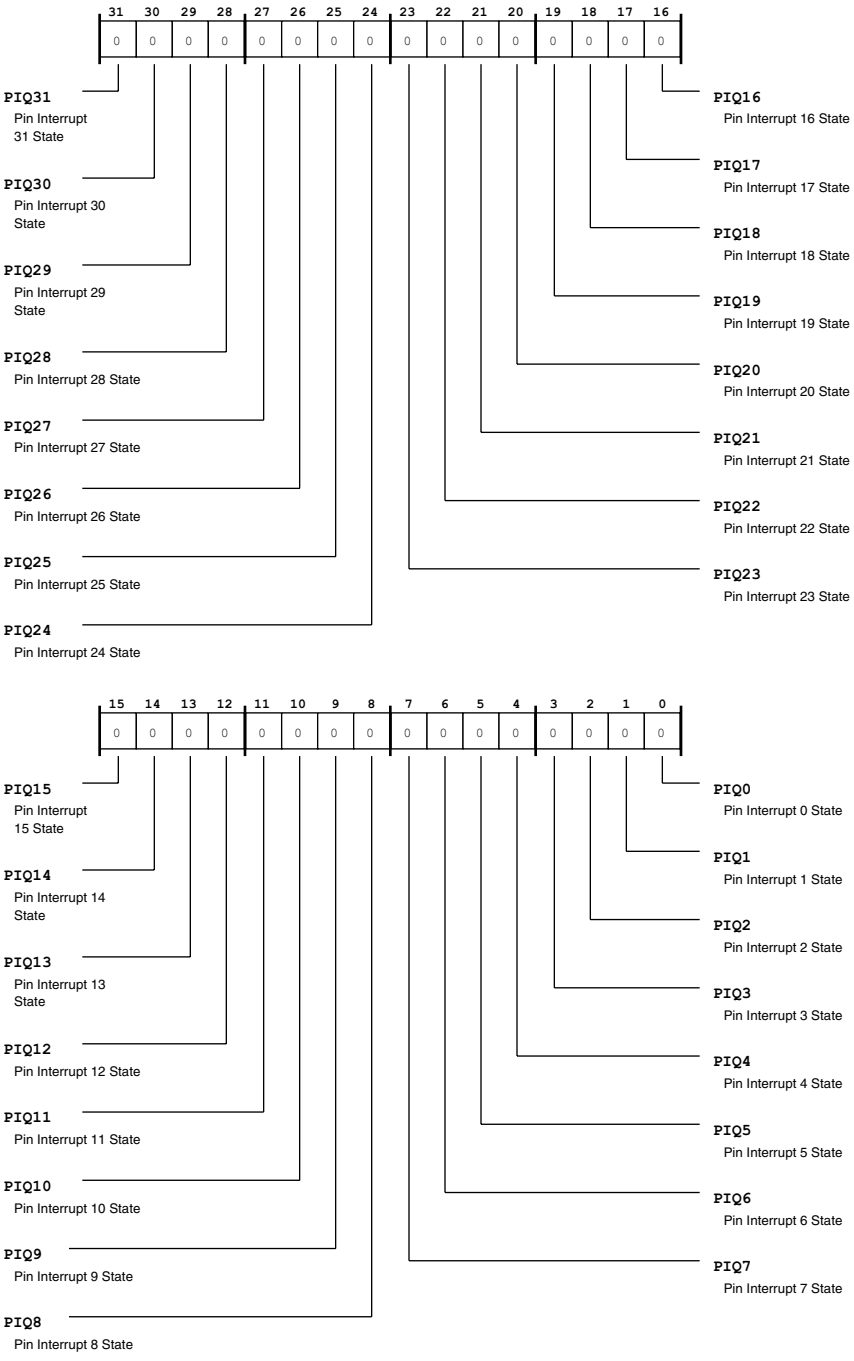


Figure 14-33: PINT_PINSTATE Register Diagram

Table 14-36: PINT_PINSTATE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/NW)	PIQ31	Pin Interrupt 31 State. Read returns pin state.
30 (R/NW)	PIQ30	Pin Interrupt 30 State. Read returns pin state.
29 (R/NW)	PIQ29	Pin Interrupt 29 State. Read returns pin state.
28 (R/NW)	PIQ28	Pin Interrupt 28 State. Read returns pin state.
27 (R/NW)	PIQ27	Pin Interrupt 27 State. Read returns pin state.
26 (R/NW)	PIQ26	Pin Interrupt 26 State. Read returns pin state.
25 (R/NW)	PIQ25	Pin Interrupt 25 State. Read returns pin state.
24 (R/NW)	PIQ24	Pin Interrupt 24 State. Read returns pin state.
23 (R/NW)	PIQ23	Pin Interrupt 23 State. Read returns pin state.
22 (R/NW)	PIQ22	Pin Interrupt 22 State. Read returns pin state.
21 (R/NW)	PIQ21	Pin Interrupt 21 State. Read returns pin state.
20 (R/NW)	PIQ20	Pin Interrupt 20 State. Read returns pin state.
19 (R/NW)	PIQ19	Pin Interrupt 19 State. Read returns pin state.
18 (R/NW)	PIQ18	Pin Interrupt 18 State. Read returns pin state.
17 (R/NW)	PIQ17	Pin Interrupt 17 State. Read returns pin state.
16 (R/NW)	PIQ16	Pin Interrupt 16 State. Read returns pin state.
15 (R/NW)	PIQ15	Pin Interrupt 15 State. Read returns pin state.

Table 14-36: PINT_PINSTATE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/NW)	PIQ14	Pin Interrupt 14 State. Read returns pin state.
13 (R/NW)	PIQ13	Pin Interrupt 13 State. Read returns pin state.
12 (R/NW)	PIQ12	Pin Interrupt 12 State. Read returns pin state.
11 (R/NW)	PIQ11	Pin Interrupt 11 State. Read returns pin state.
10 (R/NW)	PIQ10	Pin Interrupt 10 State. Read returns pin state.
9 (R/NW)	PIQ9	Pin Interrupt 9 State. Read returns pin state.
8 (R/NW)	PIQ8	Pin Interrupt 8 State. Read returns pin state.
7 (R/NW)	PIQ7	Pin Interrupt 7 State. Read returns pin state.
6 (R/NW)	PIQ6	Pin Interrupt 6 State. Read returns pin state.
5 (R/NW)	PIQ5	Pin Interrupt 5 State. Read returns pin state.
4 (R/NW)	PIQ4	Pin Interrupt 4 State. Read returns pin state.
3 (R/NW)	PIQ3	Pin Interrupt 3 State. Read returns pin state.
2 (R/NW)	PIQ2	Pin Interrupt 2 State. Read returns pin state.
1 (R/NW)	PIQ1	Pin Interrupt 1 State. Read returns pin state.
0 (R/NW)	PIQ0	Pin Interrupt 0 State. Read returns pin state.

Pint Latch Register

The PINT_LATCH register indicates 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 clears 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.

Regardless 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.

PINT_LATCH: Pint Latch Register - R/W

Reset = 0x0000 0000

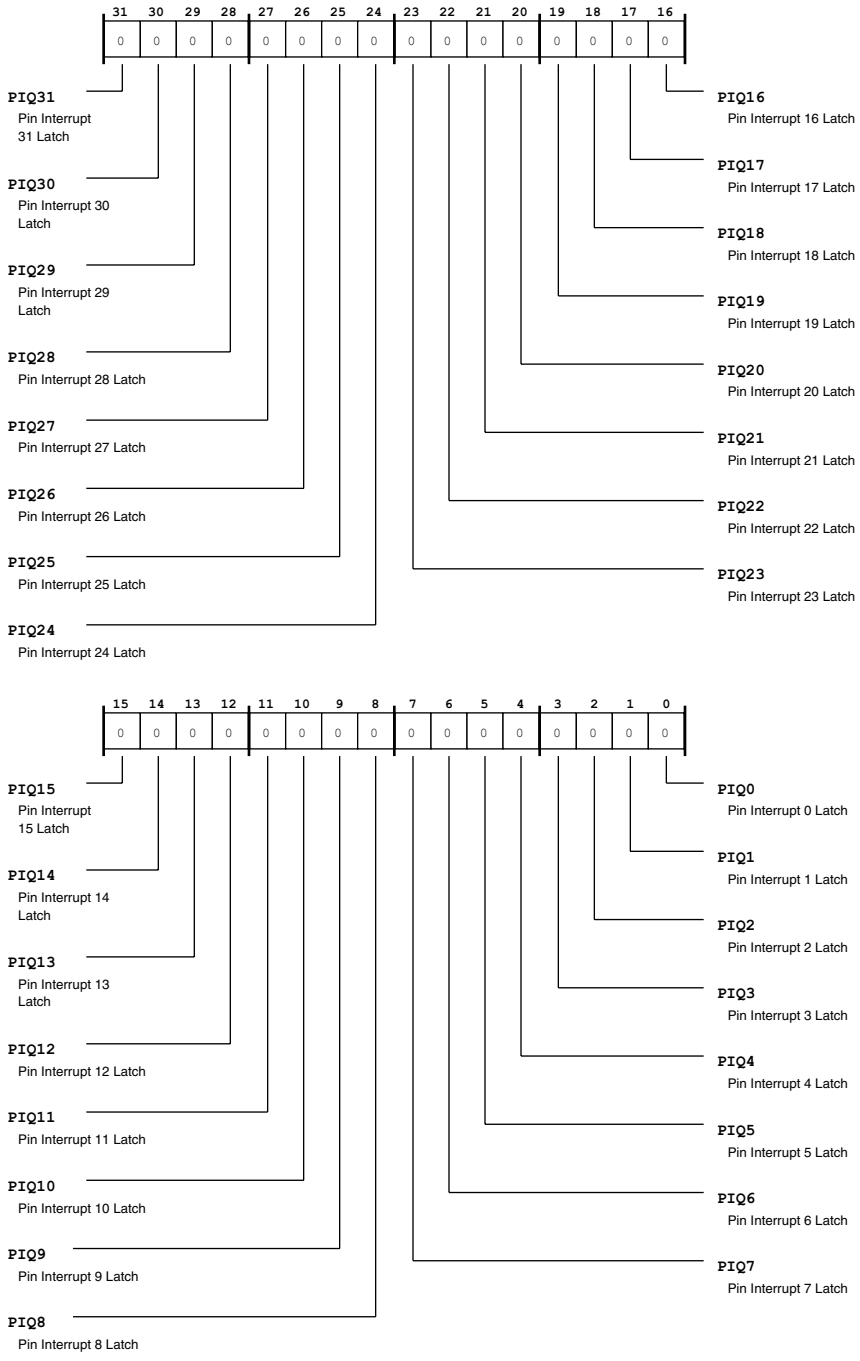


Figure 14-34: PINT_LATCH Register Diagram

Table 14-37: PINT_LATCH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Latch. If set, request latched.
30 (R/W1C)	PIQ30	Pin Interrupt 30 Latch. If set, request latched.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Latch. If set, request latched.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Latch. If set, request latched.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Latch. If set, request latched.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Latch. If set, request latched.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Latch. If set, request latched.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Latch. If set, request latched.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Latch. If set, request latched.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Latch. If set, request latched.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Latch. If set, request latched.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Latch. If set, request latched.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Latch. If set, request latched.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Latch. If set, request latched.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Latch. If set, request latched.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Latch. If set, request latched.
15 (R/W1C)	PIQ15	Pin Interrupt 15 Latch. If set, request latched.

Table 14-37: PINT_LATCH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1C)	PIQ14	Pin Interrupt 14 Latch. If set, request latched.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Latch. If set, request latched.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Latch. If set, request latched.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Latch. If set, request latched.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Latch. If set, request latched.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Latch. If set, request latched.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Latch. If set, request latched.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Latch. If set, request latched.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Latch. If set, request latched.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Latch. If set, request latched.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Latch. If set, request latched.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Latch. If set, request latched.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Latch. If set, request latched.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Latch. If set, request latched.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Latch. If set, request latched.

ADSP-BF60x PADS Register Descriptions

Pads Controller (PADS) contains the following registers.

Table 14-38: ADSP-BF60x PADS Register List

Name	Description
PADS_EMAC_PTP_CLKSEL	EMAC and PTP Clock Select Register
PADS_TWI_VSEL	TWI Voltage Selection
PADS_PORTS_HYST	GPIO Pin Hysteresis Enable Register

EMAC and PTP Clock Select Register

The PADS_EMAC_PTP_CLKSEL register selects the clock source for the EMAC module's PTP signal. The external clock (from pads) input is same for both EMAC0 and EMAC1.

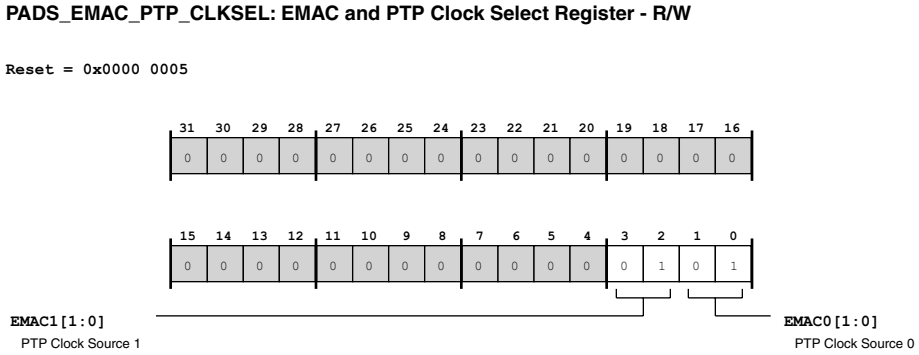


Figure 14-35: PADS_EMAC_PTP_CLKSEL Register Diagram

Table 14-39: PADS_EMAC_PTP_CLKSEL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3:2 (R/W)	EMAC1	PTP Clock Source 1. The PADS_EMAC_PTP_CLKSEL.EMAC1 selects the clock source for the PTP Block in EMAC1.	
		0	EMAC1_RMII CLK
		1	SCLK
		2	External Clock
		3	SCLK

Table 14-39: PADS_EMAC_PTP_CLKSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	EMAC0	PTP Clock Source 0. The PADS_EMAC_PTP_CLKSEL.EMAC0 selects the clock source for the PTP Block in EMAC0.	
		0	EMAC0_RMII CLK
		1	SCLK
		2	External Clock
		3	SCLK

TWI Voltage Selection

The PADS_TWI_VSEL register sets the voltage requirements for the TWI signals.

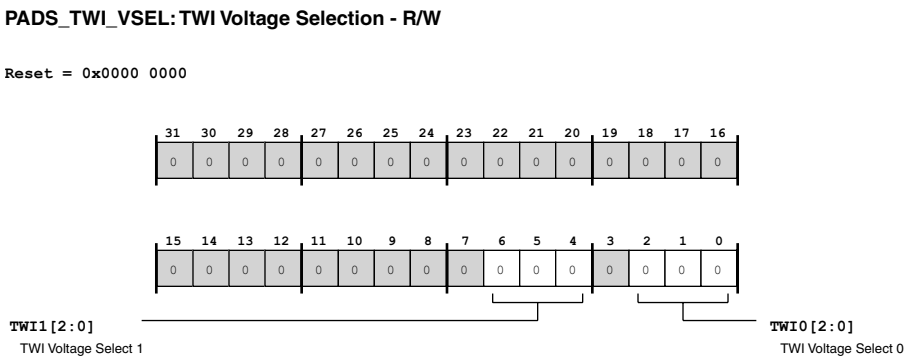


Figure 14-36: PADS_TWI_VSEL Register Diagram

Table 14-40: PADS_TWI_VSEL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6:4 (R/W)	TWI1	TWI Voltage Select 1. The PADS_TWI_VSEL.TWI1 sets the voltage requirements for the TWI_SCL and TWI_SDA pins on TWI1.	
		0	VDD_EXT=3.3V, VBUS_TWI=3.3V
		1	VDD_EXT=1.8V, VBUS_TWI=1.8V
		2	Reserved
		3	VDD_EXT=1.8V, VBUS_TWI=3.3V
		4	VDD_EXT=3.3V, VBUS_TWI=5V
		5	Reserved
		7	Reserved
2:0 (R/W)	TWI0	TWI Voltage Select 0. The PADS_TWI_VSEL.TWI0 sets the voltage requirements for the TWI_SCL and TWI_SDA pins on TWI0.	
		0	VDD_EXT=3.3V, VBUS_TWI=3.3V
		1	VDD_EXT=1.8V, VBUS_TWI=1.8V
		2	Reserved
		3	VDD_EXT=1.8V, VBUS_TWI=3.3V
		4	VDD_EXT=3.3V, VBUS_TWI=5V
		5	Reserved
		7	Reserved

GPIO Pin Hysteresis Enable Register

The PADS_PORTS_HYST register configures hysteresis for the PORT inputs. The hysteresis enable can be set only for pin groups, classified by the PORT pin multiplexing controls. For each controlled group of pins, setting the corresponding bit enables hysteresis, and clearing the bit disables hysteresis.

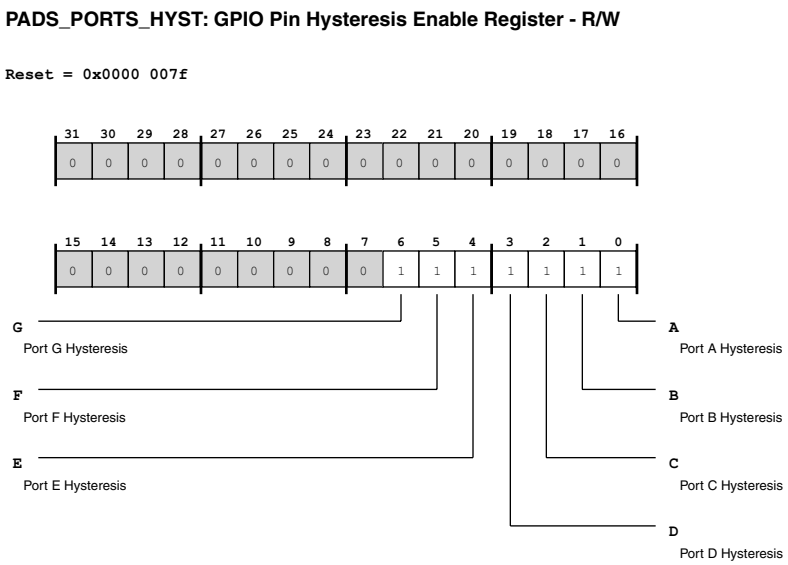


Figure 14-37: PADS_PORTS_HYST Register Diagram

Table 14-41: PADS_PORTS_HYST Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	G	Port G Hysteresis.	
		0	Disable
		1	Enable
5 (R/W)	F	Port F Hysteresis.	
		0	Disable
		1	Enable
4 (R/W)	E	Port E Hysteresis.	
		0	Disable
		1	Enable
3 (R/W)	D	Port D Hysteresis.	
		0	Disable
		1	Enable
2 (R/W)	C	Port C Hysteresis.	
		0	Disable
		1	Enable

Table 14-41: PADS_PORTS_HYST Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	B	Port B Hysteresis.	
		0	Disable
		1	Enable
0 (R/W)	A	Port A Hysteresis.	
		0	Disable
		1	Enable

15 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, and general time base functionality. Additionally, a variety of interrupts 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
- Unused timers can function as edge-sensitive pin interrupts

NOTE:

Each timer has a EMURUN bit in its TIMER_TMRn_CFG register which controls whether to run or stop the timer during emulation. The emulation event is controlled by the SDU (System Debug Unit). Please refer to the SDU chapter for more details on generation of an emulation event.

ADSP-BF60x TIMER Register List

Table 15-1: ADSP-BF60x TIMER Register List

Name	Description
TIMER_RUN	Run Register
TIMER_RUN_SET	Run Set Register
TIMER_RUN_CLR	Run Clear Register
TIMER_STOP_CFG	Stop Configuration Register
TIMER_STOP_CFG_SET	Stop Configuration Set Register
TIMER_STOP_CFG_CLR	Stop Configuration Clear Register
TIMER_DATA_IMSK	Data Interrupt Mask Register
TIMER_STAT_IMSK	Status Interrupt Mask Register
TIMER_TRG_MSK	Trigger Master Mask Register
TIMER_TRG_IE	Trigger Slave Enable Register
TIMER_DATA_ILAT	Data Interrupt Latch Register
TIMER_STAT_ILAT	Status Interrupt Latch Register
TIMER_ERR_TYPE	Error Type Status Register
TIMER_BCAST_PER	Broadcast Period Register
TIMER_BCAST_WID	Broadcast Width Register
TIMER_BCAST_DLY	Broadcast Delay Register
TIMER_TMRn_CFG	Timer n Configuration Register

Table 15-1: ADSP-BF60x TIMER Register List (Continued)

Name	Description
TIMER_TMRn_CNT	Timer n Counter Register
TIMER_TMRn_PER	Timer n Period Register
TIMER_TMRn_WID	Timer n Width Register
TIMER_TMRn_DLY	Timer n Delay Register

ADSP-BF60x TIMER Interrupt List

Table 15-2: ADSP-BF60x TIMER Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
TIMER0 Timer 0	12		LEVEL
TIMER0 Timer 1	13		LEVEL
TIMER0 Timer 2	14		LEVEL
TIMER0 Timer 3	15		LEVEL
TIMER0 Timer 4	16		LEVEL
TIMER0 Timer 5	17		LEVEL
TIMER0 Timer 6	18		LEVEL
TIMER0 Timer 7	19		LEVEL
TIMER0 Status	20		LEVEL

ADSP-BF60x TIMER Trigger List

Table 15-3: ADSP-BF60x TIMER Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
TIMER0 Timer 0	2	PULSE/EDGE
TIMER0 Timer 1	3	PULSE/EDGE
TIMER0 Timer 2	4	PULSE/EDGE
TIMER0 Timer 3	5	PULSE/EDGE
TIMER0 Timer 4	6	PULSE/EDGE
TIMER0 Timer 5	7	PULSE/EDGE

Table 15-3: ADSP-BF60x TIMER Trigger List Trigger Masters (Continued)

Description	Trigger ID	Sensitivity
TIMER0 Timer 6	8	PULSE/EDGE
TIMER0 Timer 7	9	PULSE/EDGE

Table 15-4: ADSP-BF60x TIMER Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
TIMER0 Timer 0	2	
TIMER0 Timer 1	3	
TIMER0 Timer 2	4	
TIMER0 Timer 3	5	
TIMER0 Timer 4	6	
TIMER0 Timer 5	7	
TIMER0 Timer 6	8	
TIMER0 Timer 7	9	

GP Timer Internal Interface

Timer registers are always accessed by the processor core 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 its dedicated data interrupt request. There is also one common timer status/error interrupt request output that connects to the System Event Controller. Whenever a data interrupt 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 output lines and N data trigger lines.

GP Timer 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 that processor's data sheet.

Every timer has one main input/output signal (`TMRx`) and, usually, one auxiliary input pin, used as an alternate capture input (`TM_ACIx`). Each `TMR` can either run with a time base of *SCLK* or can reference an external clock on one of two `TMR_ALT_CLKx` pins. The `TMR_ALT_CLK0`signal maps to individual alternate clock (`TM_ACLKx`) 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 TMR3's reference clock. Likewise, the `TMR_ALT_`

CLK1signal from each timer unit is connected together internally to provide a single global timer clock pin (TM_CLK) for the GP timer module, for use as an additional time base.

When clocked internally from *SCLK*, the maximum period for the timer count is ((2^{32})-1) / *SCLK* (in MHz)). The TM_ACLK and TM_ACI capture input pins are sampled every *SCLK* cycle. The duration of every low or high state must be slightly more than one *SCLK* cycle. Therefore the maximum allowed frequency of timer input signals is slightly less than *SCLK*/2. For exact timing requirements, please refer to the processor's data sheet).

GP Timer General Operation

The core of every timer is a 32-bit counter that can be interrogated through the read-only TIMER_TMR_CNT register. Once a timer has been enabled, its TIMER_TMR_CNT register is loaded with a starting value.

A timer can operate in one of several different modes, configured through the TIMER_TMR_CFG register for that timer. These modes are known as PWMOUT, EXTCLK, WIDCAP, WATCHDOG, PININT and IDLE, and are summarized in the following table.

Table 15-5: Timer Mode Descriptions

Timer Mode	Description
PWMOUT	Generates single or continuous PWM waveforms with programmable pulse width, period and delay
EXTCLK	Counts “clock ticks” from the system clock (SCLK) or 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 if it falls inside or outside of that window
PININT	Can generate an interrupt 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 period, pulsewidth, and delay registers. Therefore, these values are ready for use in the first timer period. When a timer is already running, software can write new values to the TIMER_TMR_PER, TIMER_TMR_WID and TIMER_TMR_DLY registers. The written values are buffered and do not update into the registers until the end of the current period (when the value in TIMER_TMR_CNT equals the value in TIMER_TMR_PER).

If new values are not written to these registers, the value from the previous period is re-used. Writes to these registers are atomic; it is not possible for the high word to be written without the low word also being written.All three registers are double buffered. Values written to the period, pulsewidth, 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 read back until they become active.

The usage of the TIMER_TMR_PER, TIMER_TMR_WID and TIMER_TMR_DLY registers varies, depending on the mode of the timer and is specified by TIMER_TMR_CFG.TMODE bits. See the table below for more information.

Table 15-6: Usage of the Period, Width and Delay Registers in Different Timer Modes

Timer Mode	Period	Width	DELAY
IDLE	Not writable	Not writable	Not writable
WATCHDOG	Can be updated on-the-fly. New value takes effect either upon 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.	Can be updated on-the-fly. New value takes effect either upon 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	Can be updated on-the-fly. New value takes effect either upon timer start or at the end of the current period. A write followed by immediate read returns the current operating values.	Can be updated on-the-fly. New value takes effect either upon timer start or at the end of the current period. A write followed by immediate read returns the current operating values.	Can be updated on-the-fly. New value takes effect either upon 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

NOTE: It is important to note that, if in a particular timer mode any of these three registers is not used, then software is not allowed to write into the unused one(s). For example, in WIDCAP mode, delay registers are not used. Therefore, software is not allowed to write any value to TIMER_TMR_DLY. The software must program TIMER_TMR_CFG.TMODE bits before programming these 3 registers, in order to prevent undesired operation.

Also, if software is changing TIMER_TMR_CFG.TMODE bits such that these registers change configuration from status register to writable register (for example for PWMOUT mode), hardware does not clear these registers. These values are automatically overwritten by new values specified by software.

In PWM_OUT mode with very small periods, there may not be enough time between updates from the buffer registers to write these registers; the next period may use one old value and one new

value. In order to prevent (width + pulse delay) > period errors, write the width and delay registers before the period register when decreasing the values, and write the period register before the width and delay registers when increasing the value.

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/or delay settings for each of the first three timer periods after the timer is started.

1. Program the first set of period, width and delay register values.
2. Enable the timer.
3. Immediately program the second set of register values, as needed.
4. Wait for the first timer interrupt.
5. Program the third set of register values.

RESULT:

Each new setting is then programmed when the preceding timer interrupt is received.

Configuring, Enabling and Disabling One or More Timers

1. Configure the relevant timer(s) for the operating mode and other properties through the `TIMER_TMR_CFG` register.
2. Write a 1 to the representative `TIMER_RUN` bit(s) or, alternately, use the `TIMER_RUN_SET` register to avoid disturbing the settings of other timers not being presently configured.

   STEP RESULT: The timer(s) should now be 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: Note that PWMOUT modes are the only modes where a timer can be configured for graceful termination.

4. Write a 0 to the representative TIMER_RUN bit(s) to stop the timer(s) according to their TIMER_STOP_CFG settings. Alternately, write a 1 to the appropriate TIMER_RUN_CLR bits to avoid disturbing the settings of other timers not being presently stopped.

   STEP RESULT: The timer(s) should now be stopped.

Configuring Timer Data and Status Interrupts

1. Program the proper value in the TIMER_TMR_IRQMODE field, according to the desired interrupt properties.
2. Unmask the interrupt source at the system event controller.
3. To poll the timer's TIMER_DATA_ILAT bit without generating an interrupt, set the IRQMODE field but leave the interrupt masked at the system level.
4. If enabled by the TIMER_STAT_IMSK register, interrupt requests are also generated by overflow or error conditions (wrong programming values), as reported by the TMR_STAT_ILAT bits, provided that the timer status interrupt source is unmasked at the system event controller.
5. To poll the timer's TIMER_STAT_ILAT bit without generating an interrupt, the corresponding bit must be unmasked in the TIMER_STAT_IMASK register but leave the interrupt masked at the system level.

Using the Timer Broadcast Feature

The broadcast feature provides a means to update period, width and/or delay registers simultaneously across more than one timer.

1. Enable the appropriate broadcast bits (BPEREN, BWIDEN, BDLYEN) in the TIMER_TMR_CFG registers for the timers involved in the broadcast. The broadcast bits use depend on which TIMER_BCAST registers are involved.
2. Program the TIMER_BCAST_PER register (for instance), assuming you want to broadcast the period setting across the multiple timers enabled above.

   STEP RESULT: This causes only those timers enabled above to load their TIMER_TMR_PER registers with the value specified in the TIMER_BCAST_PER register.

3. Repeat Step 2 as needed for TIMER_BCAST_WID and TIMER_BCAST_DLY settings.

Single-Pulse PWMOUT Mode

In single-pulse PWMOUT mode, the timer generates a single pulse on the TMR pin. This mode is frequently used to implement a precise delay, often in conjunction with generation of an output trigger. The assertion of a pulse is controlled by the value in TIMER_TMR_DLY, and pulse width is defined by the TIMER_TMR_WID value. TIMER_TMR_PER is not used and cannot be written in this mode. After completion of the pulse the

timer is stopped automatically, optionally generating an interrupt, if configured to do so. Pulse polarity is controlled through the TIMER_TMR_CFG.PULSEHI bit.

If configured as such, the timer can generate a data interrupt upon satisfying various conditions specified by the TIMER_TMR_CFG.IRQMODE bits.

It is not necessary to clear the relevant TIMER_RUN bit in order to stop the timer cleanly. At the end of the pulse, the timer stops automatically and the corresponding TIMER_RUN is cleared. To generate multiple discrete pulses (as opposed to a continuous PWM waveform), write a 1 to the appropriate TIMER_RUN bit, wait for the timer to stop, and then write another 1 to the same TIMER_RUN bit.

Timer Continuous PWMOUT Mode

In continuous PWMOUT mode, the timer generates repetitive pulses with well-defined period, duty cycle and pulse position. The TIMER_TMR_DLY, TIMER_TMR_PER and TIMER_TMR_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 TIMER_TMR_PER. Initially, the TMR pin remains in a de-asserted state. It toggles to an asserted state when TIMER_TMR_CNT=TIMER_TMR_DLY. The assertion sense of the TMR pin can be controlled with the TIMER_TMR_CFG.PULSEHI bit. The TMR pin holds this value for the number of clock cycles specified in TIMER_TMR_WID, after which it de-asserts and holds this value until the completion of the programmed period. The same waveform is generated repeatedly until the timer is disabled.

If configured as such, the timer can generate a data interrupt upon satisfying any of various conditions specified by the TIMER_TMR_CFG.IRQMODE bits.

It is important to guarantee that the programmed Period >= (Width+Delay). Similarly, delay must be less than period. Violating either of these criteria will result in an unpredictable waveform on the TMR pin until the situation is rectified by writing proper values to these registers.

The maximum frequency possible to generate on the TMR pin is achieved by setting TIMER_TMR_PER to 2 and TIMER_TMR_WID to 1. This makes the TMR pin toggle each SCLK clock cycle (assuming the timer is configured to clock internally), producing a duty cycle of 50%.

When a timer's TIMER_STOP_CFG bit 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 prevents truncation of the current pulse and unwanted PWM patterns at the TMR pin. The processor can determine when the timer stops running by polling the corresponding TIMER_RUN bit until it reads as read 0 or by waiting for the last interrupt (if enabled).

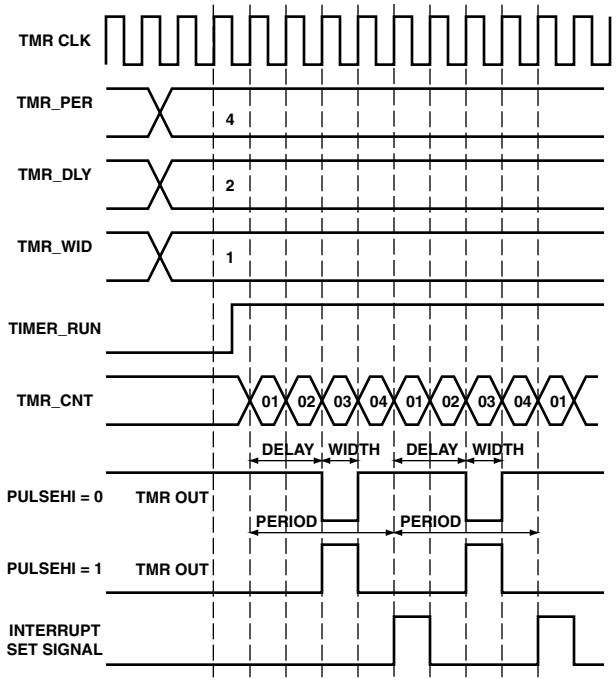


Figure 15-1: Signal Generation in Continuous PWMOUT Mode

Note that the TIMER_TMR_CFG register cannot be reconfigured until after the timer stops and TIMER_RUN reads 0.

If required, the software can force a timer to stop immediately in PWM_OUT mode by writing a 1 into TIMER_STOP_CFG followed by writing a 1 to TIMER_RUN_CLR (or by writing a 0 to the appropriate TIMER_RUN bit). This stops the timer whether the pending stop was waiting for the end of the current period or the end of the current pulse width. This feature may be used to regain immediate control of a timer during an error recovery sequence.

Use this feature carefully, because it may corrupt the PWM pattern generated at the TMR pin, though after such a stop the pin deasserts automatically. Each timer samples its TIMER_RUN bit at the end of each period. It stops cleanly at the end of the first period after TIMER_RUN is low. This implies that a timer that is disabled and then re-started, all before the end of the current period, will continue to run as if nothing happened. Typically, software should disable a PWMOUT timer and then wait for it to stop itself.

TIMER Width Capture (WIDCAP) Mode

The WIDCAP mode, often simply called capture mode, is used to measure pulse widths on the TMR or TMR_AUX_IN inputs. The polarity (active high/low) of the input signal can be selected with the TIMER_TMR_CFG.PULSEHI bit. The figure below shows the control signal flow for WIDCAP_CAP mode.

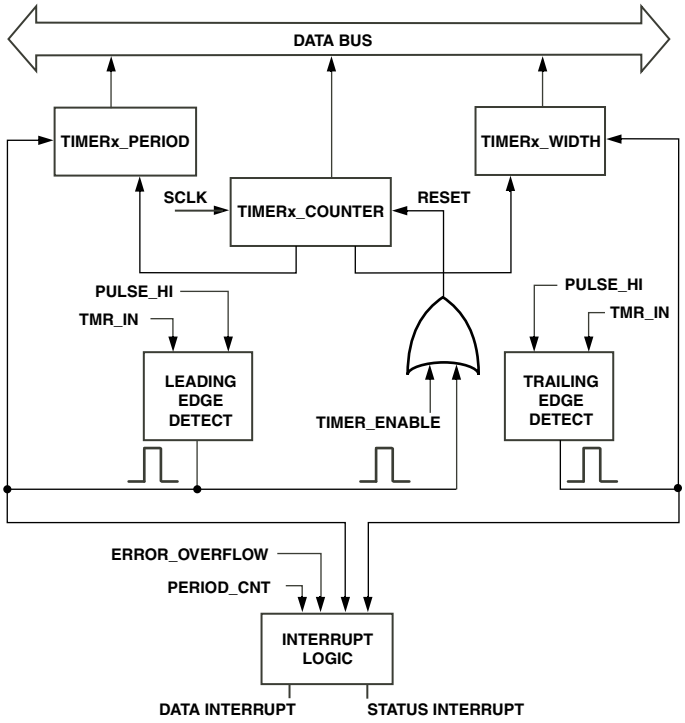


Figure 15-2: Timer Signal Flow in Width Capture Mode

In this mode, the TIMER_TMR_CFG.TINSEL bit selects between the TMR or TMR_AUX_IN 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_CNT register to 0x0000 0001 and 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 TMR_CNT register into its width buffer register. At the next leading edge, the timer transfers the current 32-bit value of its TMR_CNT register into its period buffer register. The TMR_CNT register is reset to 0x0000 0001 again, and the timer continues counting and capturing until it is disabled.

In this mode, software can measure both the pulse width and the pulse period of a waveform. The TMR_DLY register is not used in this mode. The TIMER_TMR_CFG.PULSEHI bit controls the definition of leading edge and trailing edge of the TMR/TMR_AUX_IN pin.

In WIDCAP mode, the following events always occur at the same time as one unit:

1. The TIMER_TMR_PER register is updated from the period buffer register.
2. The TIMER_TMR_WIDTH register is updated from the width buffer register.
3. The TIMER_DATA_ILAT bit gets set (if enabled).

4. A timer data trigger pulse is generated (if enabled).

The TIMER_TMR_CFG.TMODE [0] bit controls the point in time at which this set of events is executed. Taken together, these four events are called a measurement report. The TMR_STAT_ILAT register does not get set at a measurement report. A measurement report occurs, at most, once per input signal period. The current TMR_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 to signal that TMR_PER and TMR_WID are ready to be read.

When TMODE=b#1011, the measurement report occurs just after the width buffer register captures its value (at a falling edge). Subsequently, the TMR_WID register reports the pulse width measured in the pulse that has just ended, but the TMR_PER register reports the pulse period measured at the end of the previous period. This is because, if only the first trailing edge 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 TMR_PER value in this case returns 0. See the following figure for more information.

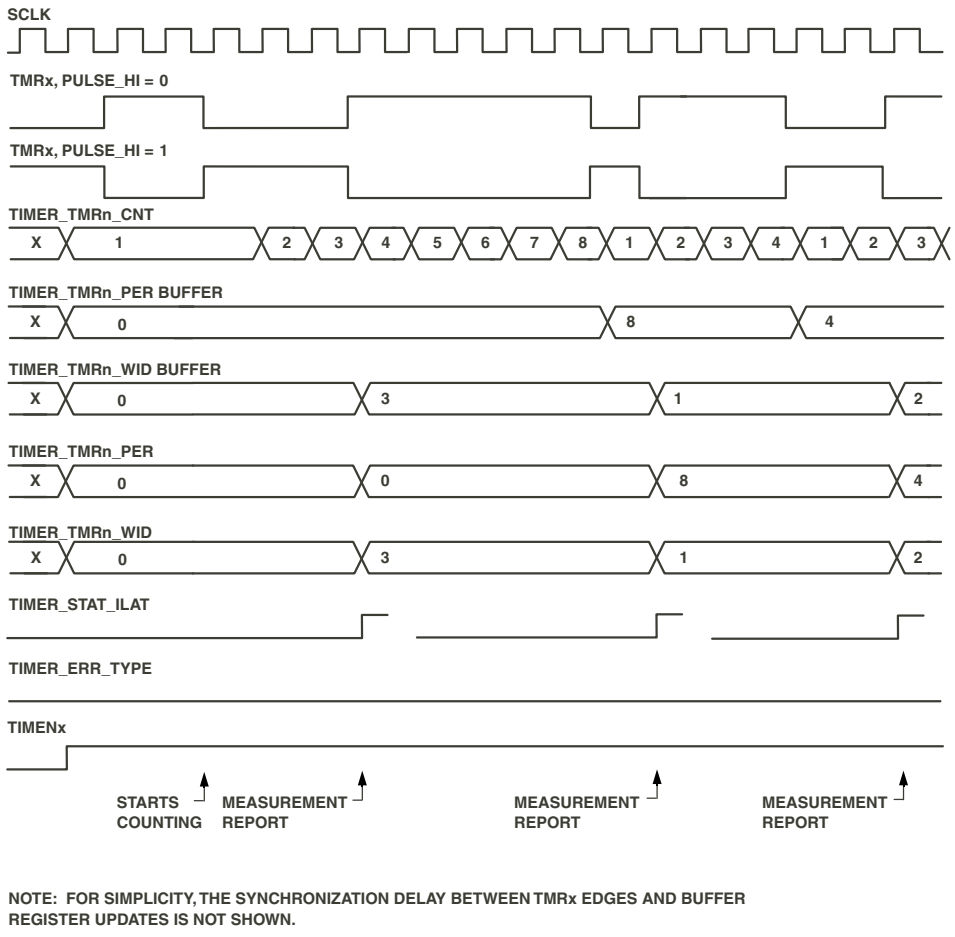


Figure 15-3: Example of Width Capture Deasserted Mode (TMODE=b#1011)

When TMODE=b#1010, the measurement report occurs just after the period buffer register captures its value (at a leading edge). Subsequently, the TMR_PER and TMR_WID registers report the pulse period and pulse width is measured in the period that has just ended. Refer to the following figure for more information.

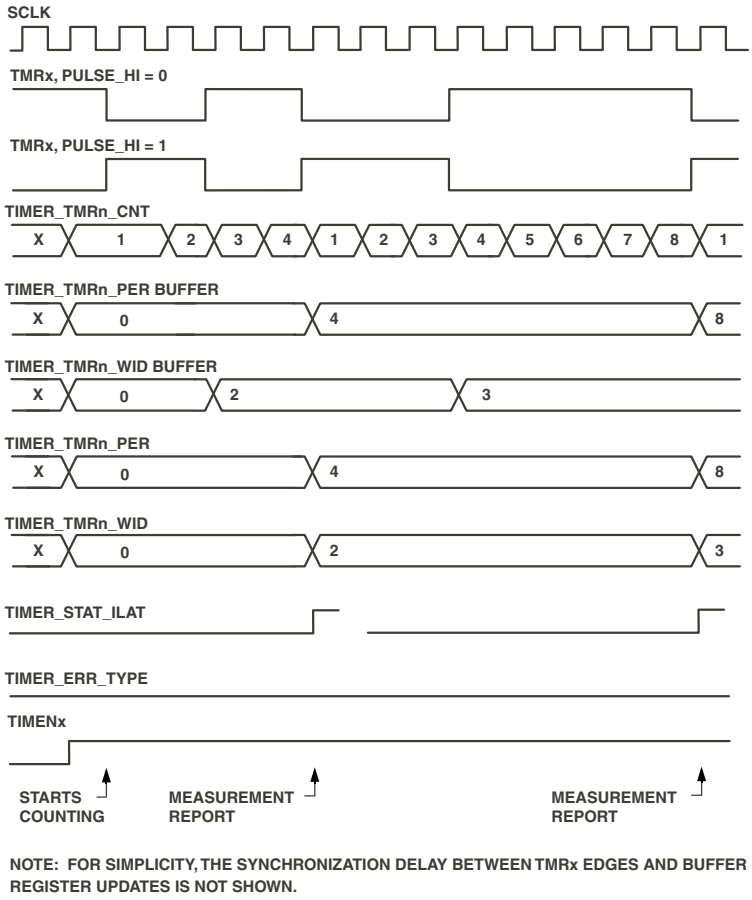


Figure 15-4: Example of Width Capture Asserted Mode (TMODE=b#1010)

To measure the pulse width of a waveform that has only one leading edge and one trailing edge, set TMODE = b#1011. If TMODE = b#1010 for this case, no period value is captured in the period buffer register. Instead, an error report interrupt is generated (if enabled) when the TMR_CNT range is exceeded and the counter wraps around. In this case, both TMR_WID and TMR_PER read 0 (because no measurement report occurred to copy the value captured in the width buffer register to TMR_WID).

If using the TMODE = b#1010 mode to measure the width of a single pulse, it is recommended to disable the timer after taking the interrupt that ends the measurement interval. If desired, the timer can then be restarted 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.

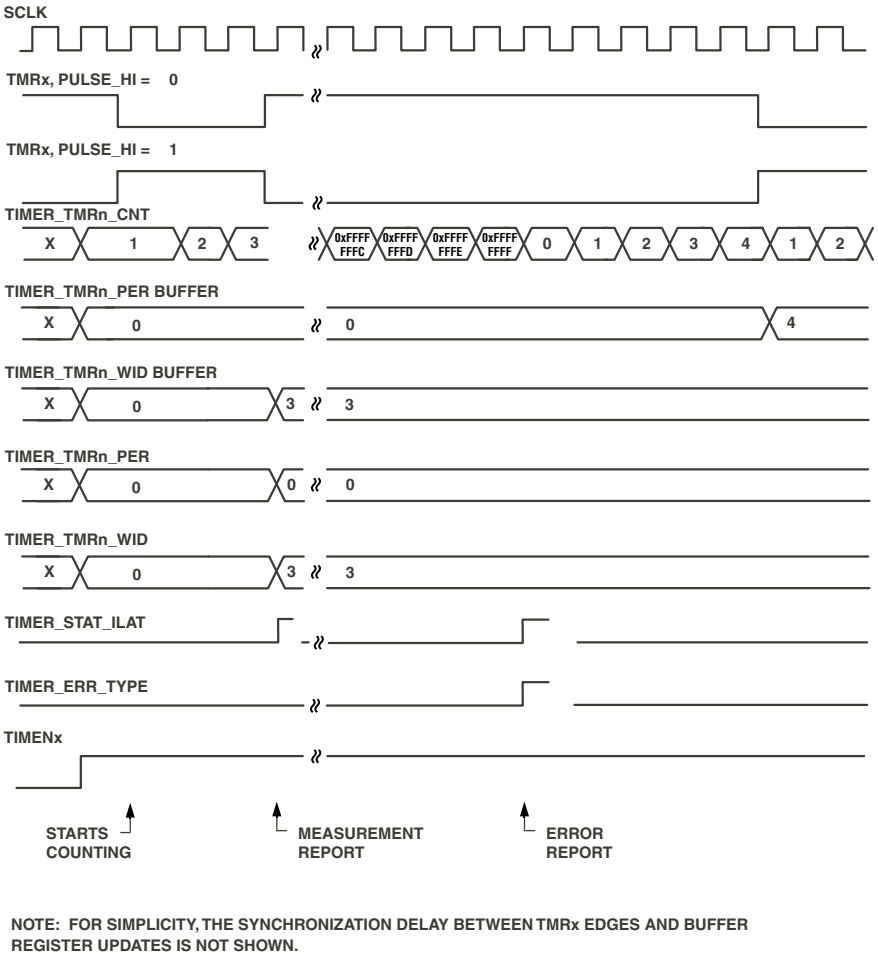
GP Timer Width Capture Mode Overflow

A timer status interrupt (if enabled) is generated if the TMR_CNT register wraps around from 0xFFFF FFFF to 0 in the absence of a leading edge. At that point, the timer's TIMER_STAT_ILAT bit gets set and the TIMER_ERR_TYP bits change appropriately, indicating a count overflow due to a period greater than the counter's range. This is called an error report. A data interrupt in WIDCAP mode indicates a new measurement is ready to be read (a measurement report). Similarly, an interrupt on the timer status interrupt line (shared interrupt for all timers) indicates an overflow if generated in WIDCAP mode.

The TMR_PER and TMR_WID registers are never updated at the time an overflow is signaled. If the timer overflowed and TMODE=b#1010, neither the TMR_PER nor the TMR_WID register were updated. If the timer overflowed and TMODE=b#1011, the TMR_PER and TMR_WID registers were updated only if a trailing edge was detected at a previous measurement report.

Software can count the number of error reports between measurement report interrupts to measure input signal periods longer than 0xFFFF FFFF. Each error report interrupt adds a full 2^{32}SCLK counts to the total for the period, but the width is ambiguous. Make sure that, if only the status interrupt is monitored by software in this case, then status interrupts from all other timers are masked.

For example, in the following figure, the period is 0x1 0000 0004, but the pulse width could be either 0x0 0000 0002 or 0x1 0000 0002.



NOTE: FOR SIMPLICITY, THE SYNCHRONIZATION DELAY BETWEEN TMRx EDGES AND BUFFER REGISTER UPDATES IS NOT SHOWN.

Figure 15-5: Example Timing for Width Capture Followed by Period Overflow (TMR_CFG.TMODE=b#1010)

The waveform applied to the TMR (or TMR_AUX_IN) pin is not required to have a 50% duty cycle, but the minimum input low time is little more than one SCLK period and the minimum input high time is little more than one SCLK period (refer to the product data sheet for details). This implies the maximum TMR input frequency is somewhat less than SCLK/2, with a 50% duty cycle. Under these conditions, the WIDCAP mode timer would measure Period = 2 and Pulse Width = 1.

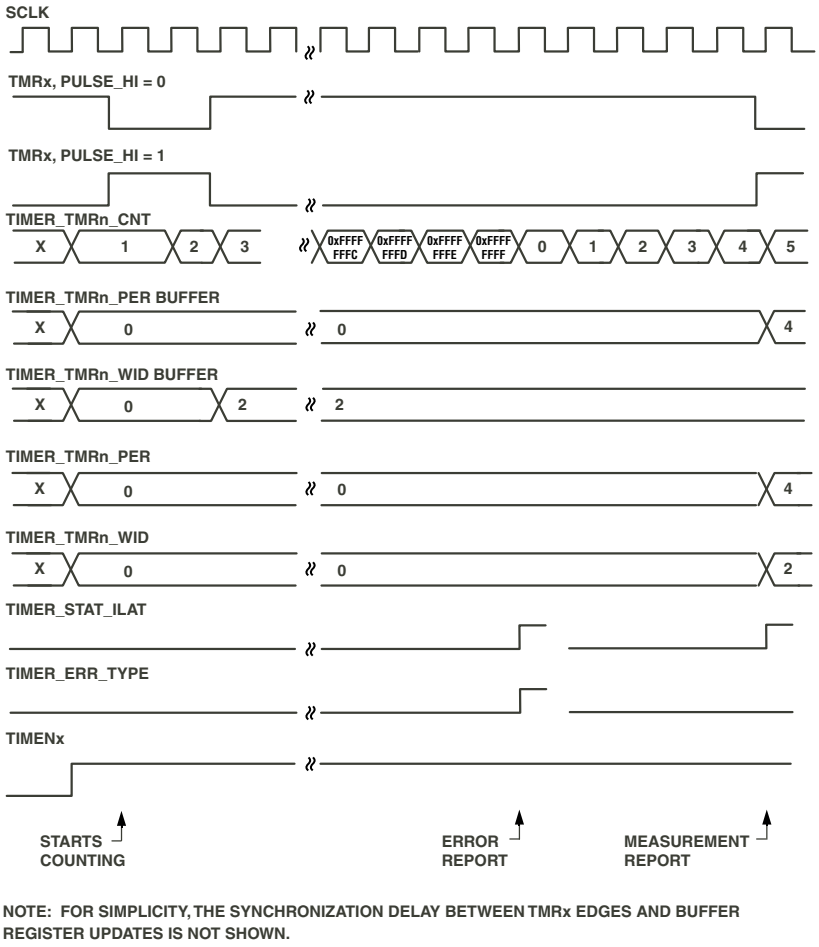


Figure 15-6: Example Timing for Width Capture Followed by Period Overflow (TMR_CFG.TMODE=b#1011)

Windowed Watchdog (WATCHDOG) Modes

In windowed watchdog (WATCHDOG) modes, a timer can take inputs from either the `TMR` pin or the `TMR_AUX_IN` 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 appropriately. Polarity selection of the input signal is performed by the `TIMER_TMR_CFG.PULSEHI` bit.

The waveform applied to the input pin in watchdog mode is not required to have a 50% duty cycle, but the minimum input pulse low time is slightly more than one *SCLK* period, and the minimum input pulse high time is slightly more than one *SCLK* period (refer to the product data sheet for details). This implies the maximum input frequency is somewhat less than *SCLK*/2 in this mode.

Timer Windowed Watchdog Width Mode

In this mode, the timer counter monitors the pulse width of an input signal on either the TMR pin or TMR_ALT_CLK pin. Software needs to program the minimum pulse width (Pmin) in the TIMER_TMR_DLY register and the maximum pulse width (Pmax) in the TIMER_TMR_PER register. Both values are programmed in terms of number of clock cycles (SCLK or TMR_ALT_CLKx). The timer can generate an interrupt if the deasserting pulse edge occurs inside the window (Pmin < Pulse Width =< Pmax) or outside the window (Pulse Width =< Pmin or Pulse Width > Pmax).

After enabling the timer in this mode, it always starts counting at the asserting edge of the input signal. This means any pulse that is already active when the timer is enabled is ignored.

With TIMER_TMR_CFG.IRQMODE=b#11, the timer generates an interrupt (if enabled) if the timed pulse width exceeds Pmax, or if the pulse width is less than Pmin. After attaining Pmax, the pulse still remains 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_CFG.IRQMODE=b#10.

In this mode, a trailing edge on the input pin triggers capturing of pulse width into the TIMER_TMR_WID register. During the inactive portion of the input signal, the internal counter does not increment. Refer to the figure below for signal flow in this mode.

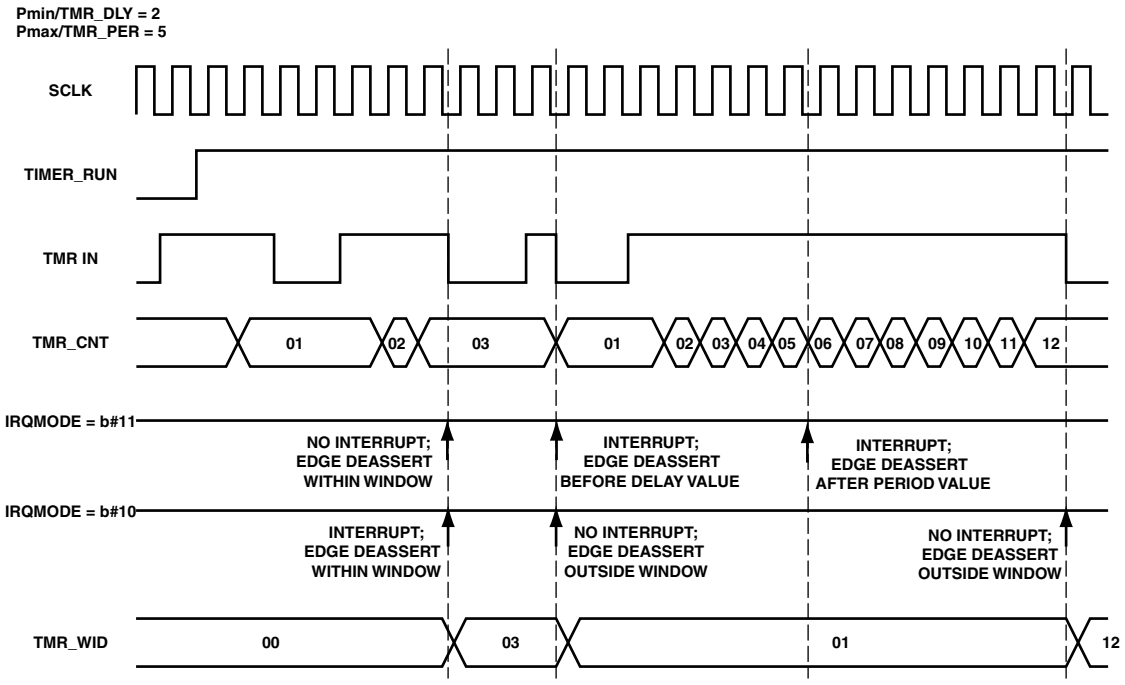


Figure 15-7: Watchdog Width Mode Timing

If it is required to check only the upper limit on pulse width (Pmax but not Pmin) then Pmin must be programmed as 0 or 1. In such a case, it is better to use IRQ_MODE =b#11. With IRQ_MODE = b#10, a pulse width of 1 clock cycle will result in an interrupt. For details see the table below.

Table 15-7: 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 =< TMR_ PER	Interrupt at deasserting edge of input signal	No Interrupt	No Error Interrupt
		PW > TMR_ PER	No Interrupt	Interrupt when Pulse with exceeds Pmax (Period Register) Value	No Error Interrupt
> 1 but <= (Period -1)	Anything > 1	PW =< TMR_ DLY	No Interrupt	Interrupt at De-asserting edge of input Signal	No Error Interrupt
		TMR_DLY < PW<= TMR_ PER	Interrupt at deasserting edge of input signal	No Interrupt	No Error Interrupt
		PW > TMR_ PER	No Interrupt	Interrupt when Pulse with exceeds Pmax (Period Register) Value	No Error Interrupt
>= Period	-	PW <= TMR_ PER	Undefined	Undefined	No Error Interrupt
	-	PW > TMR_ PER	Undefined	Undefined	b#11 Error Type
-	0	-	Undefined	Undefined	b#10 Error Type

Timer Windowed Watchdog Period Mode

In this mode, the timer monitors the number of clock cycles between two consecutive rising/falling edges of an input signal on either the TMR pin or `TMR_AUX_IN` pin. Software needs to program the required

minimum number of clock cycles (Tmin) in the TIMER_TMR_DLY register and the required maximum allowed number of clock cycles (Tmax) in the TIMER_TMR_PER register. Both values are programmed in terms of number of clock cycles (SCLK or TMR_ALT_CLKx). The timer can generate an interrupt if two consecutive occurrences of an active edge are within a specified window (Tmin < Pulse Period <=Tmax) or outside (Pulse Width<=Tmin or Tmax < Pulse Width) a specified window.

With TIMER_TMR_CFG.IRQMODE=b#11, if the pulse period ever exceeds Tmax or if it is less than or equal to Tmin, the timer generates an interrupt if unmasked. After attaining the Tmax 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_CFG.IRQMODE=b#10. Refer to the figure below for timer functionality in period watchdog mode.

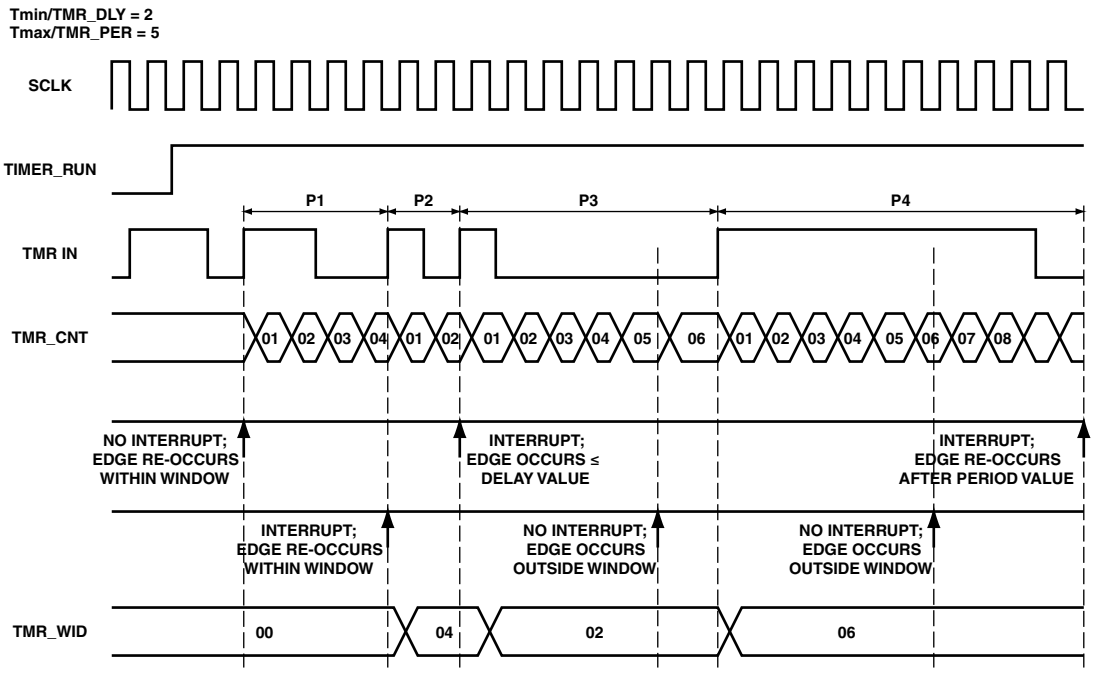


Figure 15-8: Watchdog Period Mode Timing

If it is required to check only the upper limit on period (Tmax value but not Tmin value) then Tmin can be programmed as 0 or 1 in this mode. For details refer to the table below.

Table 15-8: 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 =< 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 =< Period -1	Anything >= 2	Pulse Period =< TMR_DLY	No Interrupt	Interrupt at deasserting edge of input signal	No Error Interrupt
		TMR_DLY < Pulse Period =< 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
>= Period	-	Pulse Period < TMR_PER	Undefined	Undefined	No Error Interrupt
		Pulse Period >= 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 TMR pin or the `TMR_AUX_IN` pin (whichever is selected by `TIMER_TMR_CFG.TINSEL`) can cause an edge-based interrupt if enabled. The event on the input pin can set the `TIMER_DATA_ILAT` bit and issue a system interrupt request. Active edge polarity can be changed by programming the `TIMER_TMR_CFG.PULSEHI` bit.

Since the interrupt 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, the timer will also generate a trigger pulse if it is enabled in the `TIMER_TRG_MSK` register. Due to configuration of polarity, glitches at the input may cause

an undesired interrupt to be generated. To avoid this, software must ensure that interrupts are unmasked only after configuring desired polarity.

TIMER External Clock (EXTCLK) Mode

Use the EXTCLK mode, sometimes referred to as the counter mode, to count external events, that is, signal edges on either the TMR or TMR_AUX_IN 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 TIMER_TMR_CNT represents the number of leading edge events detected. The TIMER_TMR_PER register is programmed with the value of the maximum timer external count desired before stopping and/or issuing an interrupt or trigger.

The TIMER_TMR_CFG.PULSEHI bit determines the polarity of the leading edge on the input pin. The TIMER_TMR_CFG.TINSEL bit selects whether event is counted on TMR pin or on the TMR_AUX_IN pin. The TIMER_STAT_ILAT and TIMER_ERR_TYP bits are set if TIMER_TMR_CNT wraps around from 0xFFFF FFFF to 0 or if the period = 0 at startup or when TIMER_TMR_CNT rolls over (from count = period to count = 0x1). The TIMER_TMR_WID and TIMER_TMR_DLY registers are unused in this mode.

The figure below shows a flow diagram for EXTCLK mode.

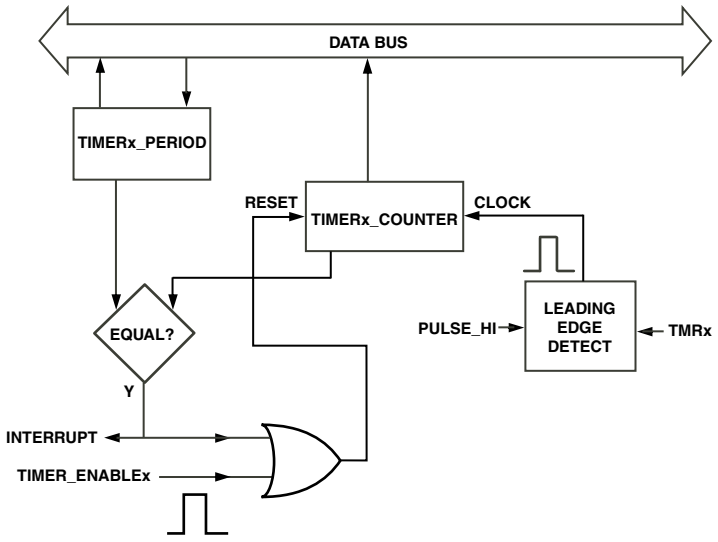


Figure 15-9: EXTCLK Mode Control Flow

The waveform applied to the input pin is not required to have a 50% duty cycle, but the minimum input low time and input high time are both slightly more than one SCLK period, (refer to the product data sheet for details). This implies the maximum input frequency is slightly less than SCLK/2. The period may be programmed to any value from 1 to (2^{32} – 1), inclusive.

After the timer has started, it resets TIMER_TMR_CNT to 0x0 and then waits for the first leading edge on the input pin. This edge causes TMR_CNT to be incremented to the value 0x1, and every subsequent leading edge increments it by one. After TMR_CNT reaches the value programmed in TIMER_TMR_PER, the corresponding

TIMER_DATA_ILAT bit is set, and an interrupt and trigger are both generated (if enabled). The next leading edge reloads the TMR_CNT again with 0x1, and the timer continues counting until it is disabled.

Timer Illegal States

These definitions are used in the table below:

- 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 TMR_CNT matches the value in TMR_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 possible count value of 0xFFFF FFFF. The counter does not have a large enough range to express the next greater value and so erroneously loads a new value of 0x0000 0000.
- Unchanged. No new error.

When ERR_TYPE is designated unchanged, it displays the previously reported error code orb# 00 if there has been no error since this timer was enabled.

When TIMER_STAT_ILAT is unchanged, it reads 0 if there has been no error or overflow since this timer was enabled, or if software has performed a W1C to clear any previous error. If a previous error has not been acknowledged by software, STAT_ILAT reads 1. Software should read STAT_ILAT to check for an error. If a particular timer's bit is set there, software can then read TIMER_ERR_TYPE for more information. Once detected, software should W1C the appropriate STAT_ILAT bit to acknowledge the error.

The following tables can be read as:

- In mode __ at event __,
- if TMR_PER is __ and TMR_WID is __ and TMR_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 may cause unwanted behavior of the TMR pin.

NOTE: For PININT mode error functionality is not used.

Continuous PWMOUT Mode

Table 15-9: Startup Event

TMR_PER	TMR_DLY	MR_WID	TMR_WID + TMR_DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
=< 1	Anything other than period[8]	Anything	Anything	b#10	Set
>= 2	Anything including 0, excluding TMR_ PER value	Anything including 0	=< 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 15-10: Rollover Event

TMR_PER	TMR_DLY	TMR_WID	TMR_ WID + TMR_ DLY	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	=<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 15-11: Overflow Event (On TMR_PER Register Programming Error Only)

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	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 15-12: Startup Event

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE TIMER_ STAT_ILAT (if enabled)	
NA	Anything	== 0	Anything	b#11[11]	Set
NA	Anything including 0	>=1	$> 2^{32} -1$	Unchanged	Unchanged
NA	Anything including 0	>=1	$> 2^{32} -1$	b#11	Set

Table 16: Overflow Event (On another error, such as DELAY + WIDTH >= 2^{32} -1)

TMR_PER	TMR_DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STA_ILAT (if enabled)
Anything	Anything	Anything	Anything	b#01	Set

WID CAP Mode

For WID CAP mode, the `TMR_PER` and `TMR_WID` registers are read-only and the `TMR_DLY` register is not used. Therefore no startup or rollover errors are possible.

Table 15-1: Overflow Event

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
Anything	NA	Anything	NA	b#01	Set

EXTCLK Mode

Table 15-2: Startup Event

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
=0	NA	NA	NA	b#01	Set
>=1	NA	NA	NA	Unchanged	Unchanged

Table 15-3: Rollover Event

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
=0	NA	NA	NA	b#01	Set
>=1	NA	NA	NA	Unchanged	Unchanged

Table 15-4: Overflow Event (On TMR_PER Register = 0 Only)

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
Anything	NA	NA	NA	b#01	Set

WATCHDOG Events

Table 15-5: Startup Event

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
=< Allowed MIN[12]	Anything < PERIOD	NA	NA	b#01	Set
> Allowed MIN	Anything < PERIOD	NA	NA	Unchanged	Unchanged
> Allowed MIN	Anything >= PERIOD	Refer to WATCHDOG Mode tables			

Table 15-6: Rollover Event

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
=< Allowed MIN[10]	Anything < PERIOD	NA	NA	b#01	Set

Table 15-6: Rollover Event (Continued)

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
> Allowed MIN	Anything	NA	NA	Unchanged	Unchanged
> Allowed MIN	Anything >= PERIOD	Refer to WATCHDOG Mode tables			

Table 15-7: Overflow Event

TMR_ PER	TMR_ DLY	TMR_ WID	TMR_ WID + TMR_ DLY	ERR_TYPE	TIMER_STAT_ILAT (if enabled)
Anything	Anything	NA	NA	b#01	Set

ADSP-BF60x TIMER Register Descriptions

General Purpose Timer Block (TIMER) contains the following registers.

Table 15-8: ADSP-BF60x TIMER Register List

Name	Description
TIMER_RUN	Run Register
TIMER_RUN_SET	Run Set Register
TIMER_RUN_CLR	Run Clear Register
TIMER_STOP_CFG	Stop Configuration Register
TIMER_STOP_CFG_SET	Stop Configuration Set Register
TIMER_STOP_CFG_CLR	Stop Configuration Clear Register
TIMER_DATA_IMSK	Data Interrupt Mask Register
TIMER_STAT_IMSK	Status Interrupt Mask Register
TIMER_TRG_MSK	Trigger Master Mask Register
TIMER_TRG_IE	Trigger Slave Enable Register
TIMER_DATA_ILAT	Data Interrupt Latch Register
TIMER_STAT_ILAT	Status Interrupt Latch Register

Table 15-8: ADSP-BF60x TIMER Register List (Continued)

Name	Description
TIMER_ERR_TYPE	Error Type Status Register
TIMER_BCAST_PER	Broadcast Period Register
TIMER_BCAST_WID	Broadcast Width Register
TIMER_BCAST_DLY	Broadcast Delay Register
TIMER_TMRn_CFG	Timer n Configuration Register
TIMER_TMRn_CNT	Timer n Counter Register
TIMER_TMRn_PER	Timer n Period Register
TIMER_TMRn_WID	Timer n Width Register
TIMER_TMRn_DLY	Timer n Delay Register

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_TMRn_CFG.TMODE bits are set to 111.

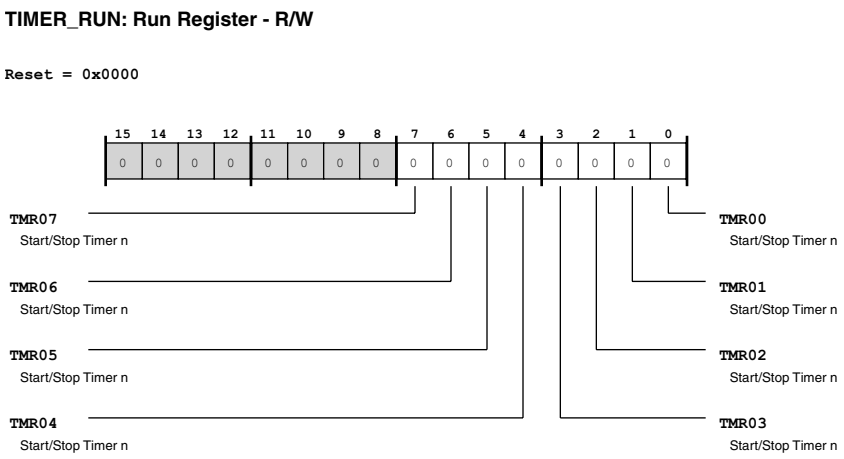


Figure 15-10: TIMER_RUN Register Diagram

Table 15-9: TIMER_RUN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMRnn	Start/Stop Timer n. For all TIMER_RUN.TMRnn bits, write =0 for stop, and write =1 for start. Read =1 when timer is running.

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.

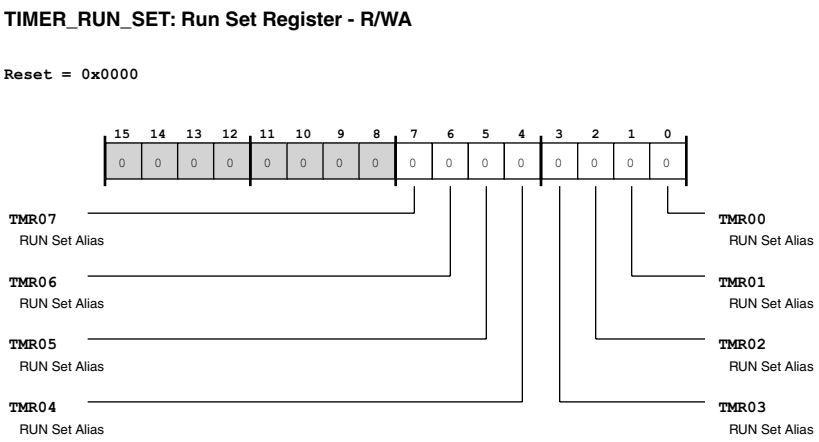


Figure 15-11: TIMER_RUN_SET Register Diagram

Table 15-10: TIMER_RUN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W1S)	TMRnn	RUN Set Alias. For all TIMER_RUN_SET.TMRnn bits, write =0 has no effect, and write =1 for start (setting the corresponding start/stop bit in the TIMER_RUN register). Using TIMER_RUN_SET to set start/stop bits permits starting specific timers without influencing the run status of other timers.

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.

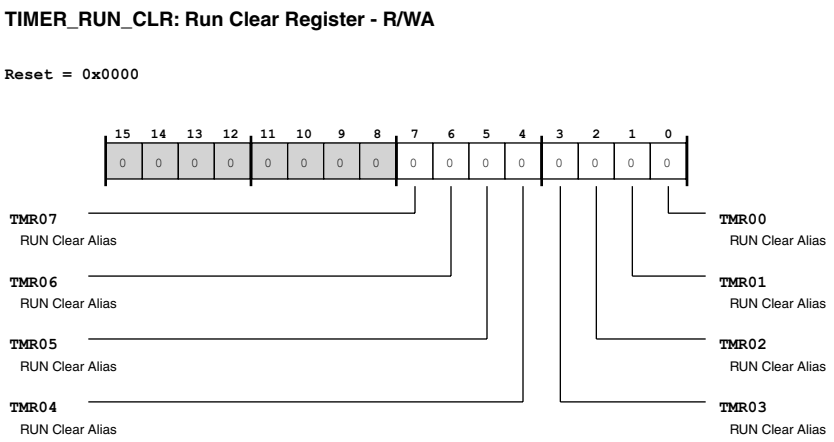


Figure 15-12: TIMER_RUN_CLR Register Diagram

Table 15-11: TIMER_RUN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W1C)	TMRnn	RUN Clear Alias. For all TIMER_RUN_CLR.TMRnn bits, write =0 has no effect, and write =1 for stop (clearing the corresponding in start/stop bit in the TIMER_RUN register). Using TIMER_RUN_CLR to clear start/stop bits permits stopping specific timers without influencing run status of other timers.

Stop Configuration Register

The TIMER_STOP_CFG 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.

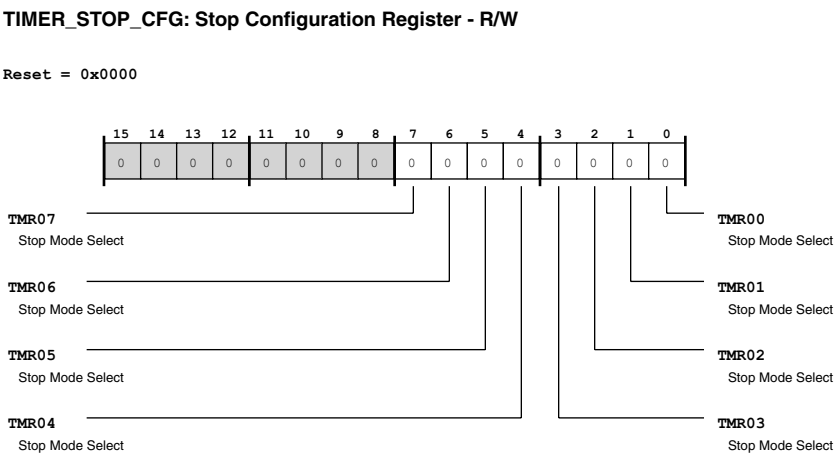


Figure 15-13: TIMER_STOP_CFG Register Diagram

Table 15-12: TIMER_STOP_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMRnn	Stop Mode Select. For all TIMER_STOP_CFG.TMRnn bits, write =0 for graceful termination (PWMOUT modes only), and write =1 for abrupt (immediate halt) on stop.

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 TIMER_STOP_CFG, software must write a 1 to the corresponding bit of TIMER_STOP_CFG_SET. Writing a zero has no effect. Because TIMER_STOP_CFG_SET 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 returns 0x0000.

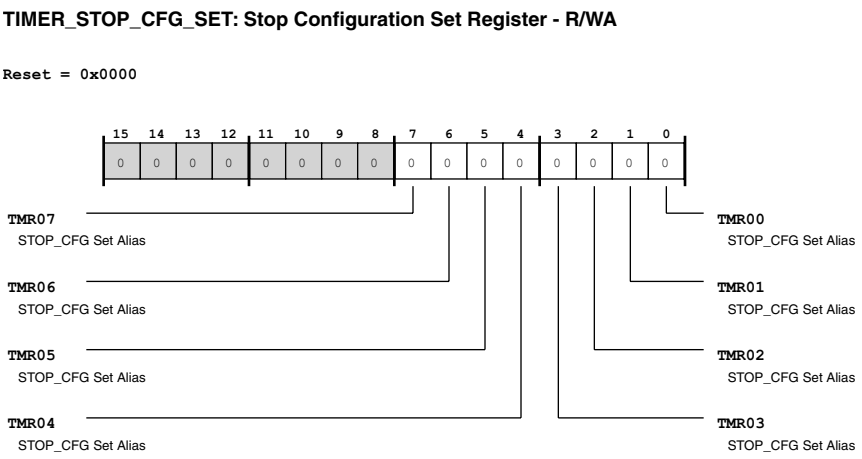


Figure 15-14: TIMER_STOP_CFG_SET Register Diagram

Table 15-13: TIMER_STOP_CFG_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W1S)	TMRnn	STOP_CFG Set Alias. For all TIMER_STOP_CFG_SET.TMRnn bits, write =0 has no effect, and write =1 for abrupt stop (setting the corresponding stop mode select bit in the TIMER_STOP_CFG register). Using TIMER_STOP_CFG_SET to set stop mode bits permits configuring specific timers without influencing the stop mode configuration of other timers.

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. Writing a zero has no effect. Because TIMER_STOP_CFG_CLR 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 returns 0x0000.

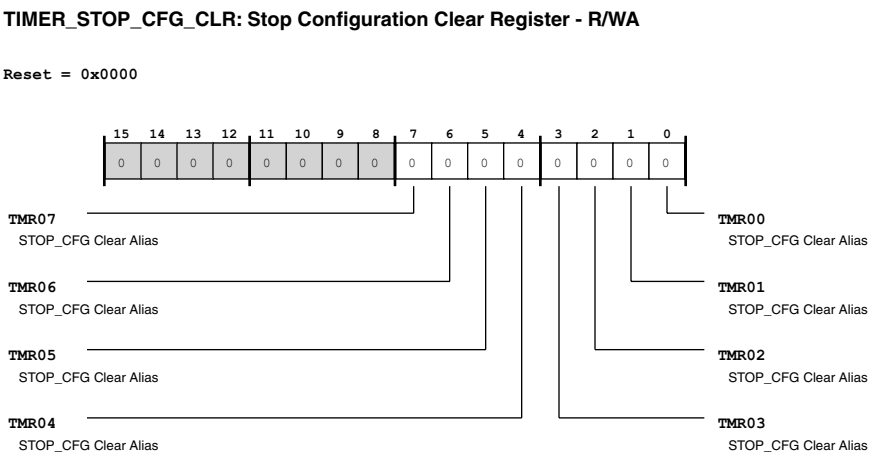


Figure 15-15: TIMER_STOP_CFG_CLR Register Diagram

Table 15-14: TIMER_STOP_CFG_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W1C)	TMRnn	STOP_CFG Clear Alias. For all TIMER_STOP_CFG_CLR.TMRnn bits, write =0 has no effect, and write =1 for graceful stop in PWMOUT modes (clearing the corresponding stop mode select bit in the TIMER_STOP_CFG register). Using TIMER_STOP_CFG_CLR to clear stop mode bits permits configuring specific timers without influencing the stop mode configuration of other timers.

Data Interrupt Mask Register

Each timer may generate a unique processor data interrupt request signal. The TIMER_DATA_IMSK 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.

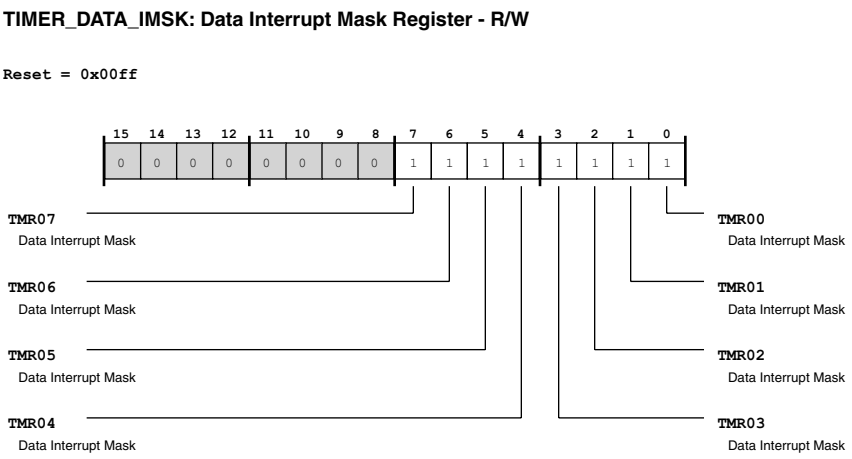


Figure 15-16: TIMER_DATA_IMSK Register Diagram

Table 15-15: TIMER_DATA_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMRnn	Data Interrupt Mask. For all TIMER_DATA_IMSK.TMRnn bits, write =0 unmasks (enables) the corresponding data interrupt request, and write =1 masks (disables) the corresponding data interrupt request.

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 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.

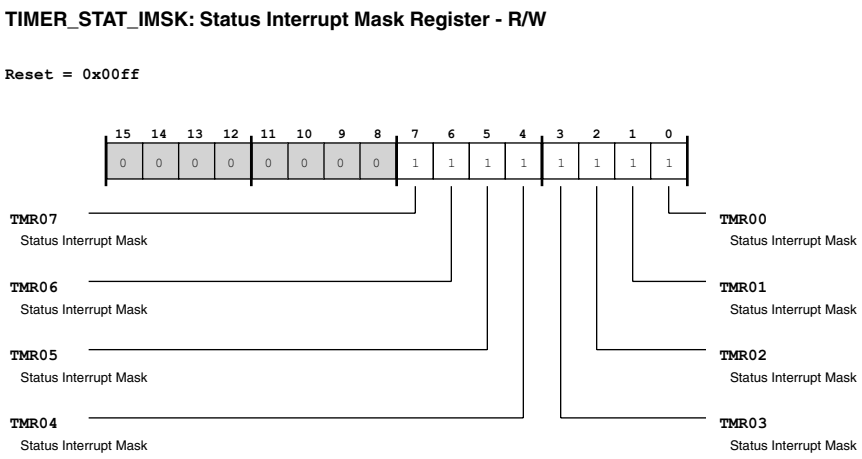


Figure 15-17: TIMER_STAT_IMSK Register Diagram

Table 15-16: TIMER_STAT_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMRnn	Status Interrupt Mask. For all TIMER_STAT_IMSK.TMRnn bits, write =0 unmasks (enables) the corresponding status interrupt request, and write =1 masks (disables) the corresponding status interrupt request.

Trigger Master Mask Register

As a trigger master, each timer can generate a unique data trigger pulse signal. The TIMER_TRG_MSK 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.

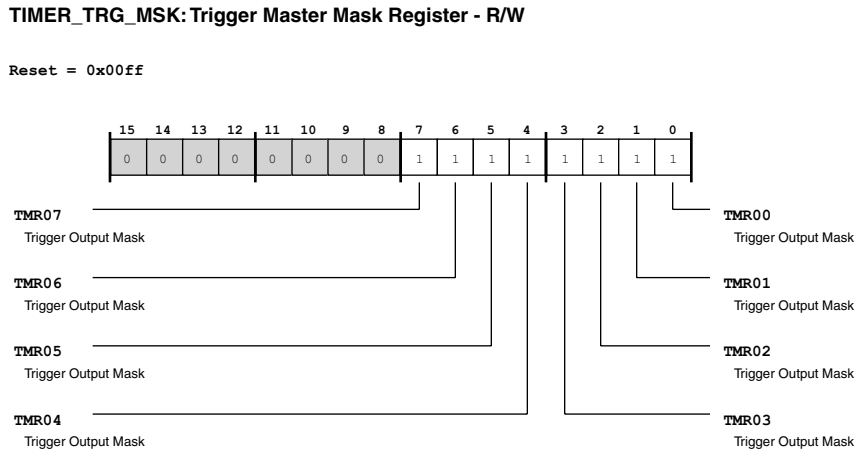


Figure 15-18: TIMER_TRG_MSK Register Diagram

Table 15-17: TIMER_TRG_MSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMRnn	Trigger Output Mask. For all TIMER_TRG_MSK.TMRnn bits, write =0 unmasks (enables) the corresponding data trigger output, and write =1 masks (disables) the corresponding data trigger output.

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.

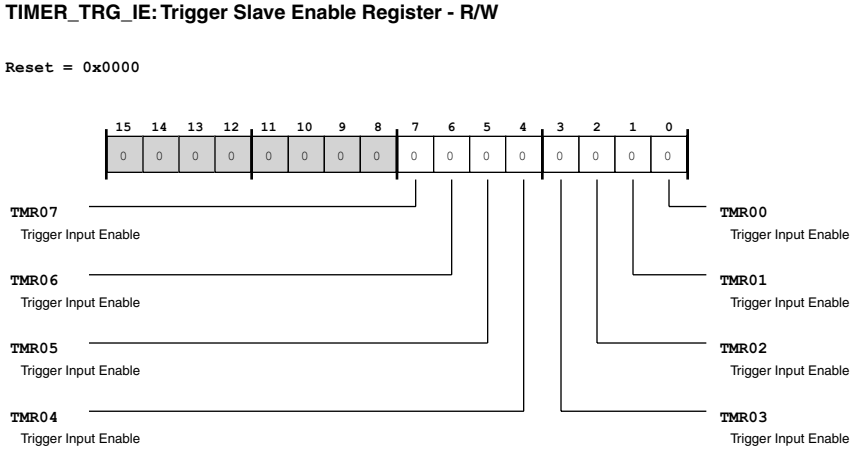


Figure 15-19: TIMER_TRG_IE Register Diagram

Table 15-18: TIMER_TRG_IE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMRnn	Trigger Input Enable. For all TIMER_TRG_IE.TMRnn bits, write =0 disables the corresponding trigger input, and write =1 enables the corresponding trigger input.

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_TMRn_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 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.

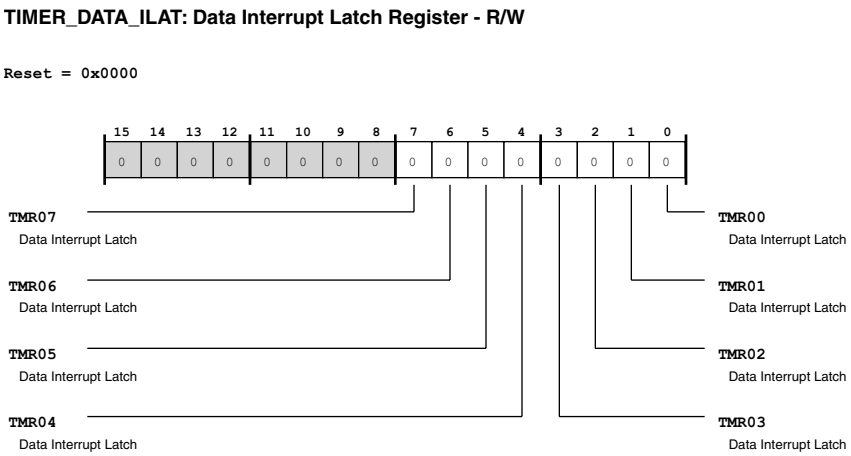


Figure 15-20: TIMER_DATA_ILAT Register Diagram

Table 15-19: TIMER_DATA_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W1C)	TMRnn	Data Interrupt Latch. For all TIMER_DATA_ILAT.TMRnn 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 TIMER_TMRn_CFG.IRQMODE bit has occurred).

Status Interrupt Latch Register

The TIMER_STAT_ILAT holds the latched interrupt status for error interrupts, 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 source.

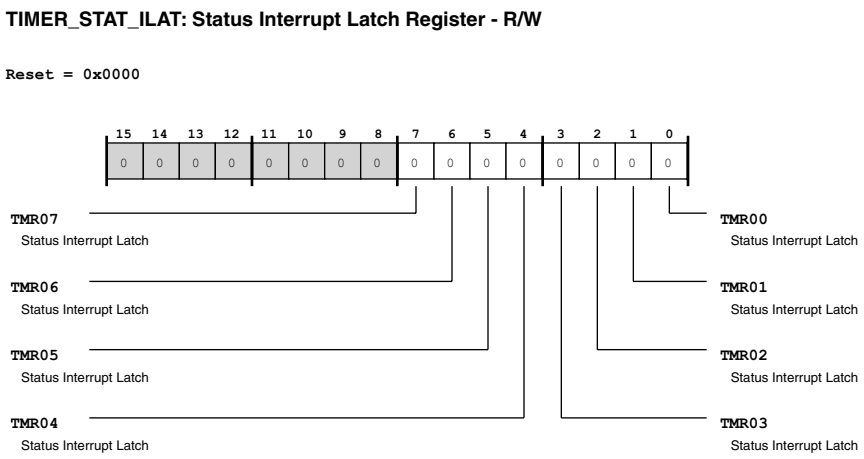


Figure 15-21: TIMER_STAT_ILAT Register Diagram

Table 15-20: TIMER_STAT_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W1C)	TMRnn	Status Interrupt Latch. For all TIMER_STAT_ILAT.TMRnn bits, status of 0 indicates no error interrupt is latched, and status of 1 indicates a timer counter overflow or programming error interrupt is latched.

Error Type Status Register

The TIMER_ERR_TYPE 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 interrupt is latched in TIMER_STAT_ILAT, the corresponding TERRx bits in TIMER_ERR_TYPE 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 TIMER_ERR_TYPE.

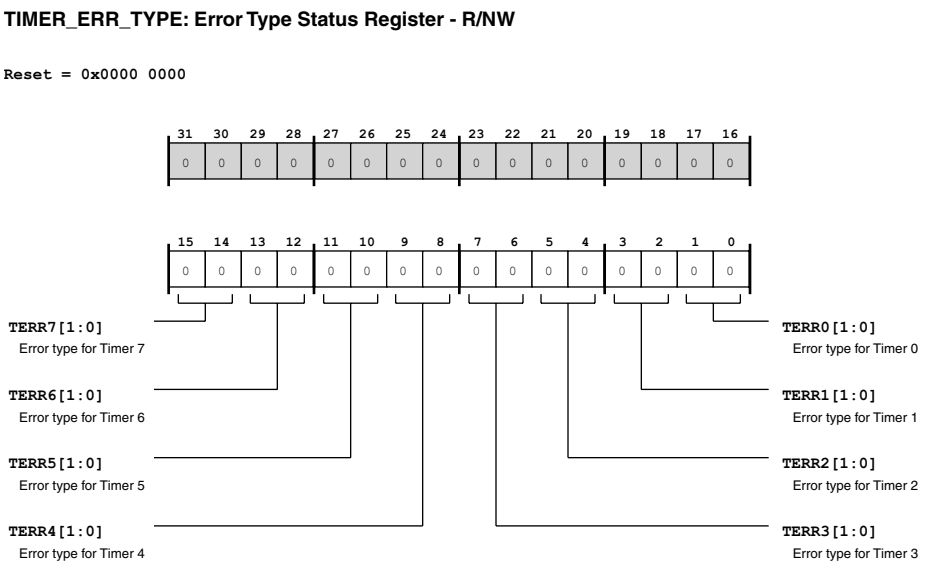


Figure 15-22: TIMER_ERR_TYPE Register Diagram

Table 15-21: TIMER_ERR_TYPE Register Fields

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

Table 15-21: TIMER_ERR_TYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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

Broadcast Period Register

For timers with `TIMER_TMRn_CFG.BPEREN` enabled, a write to `TIMER_BCAST_PER` concurrently updates the period (`TIMER_TMRn_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_TMRn_PER` register.

TIMER_BCAST_PER: Broadcast Period Register - R/WA

Reset = 0x0000 0000

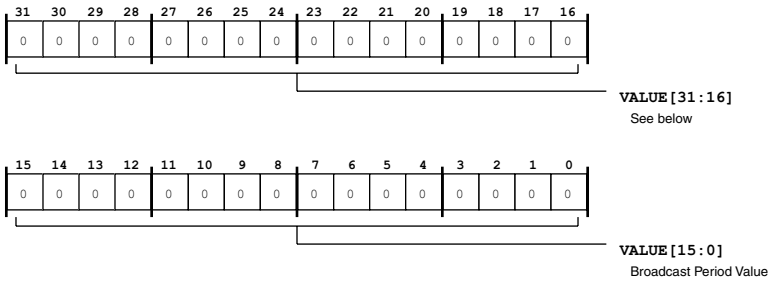


Figure 15-23: TIMER_BCAST_PER Register Diagram

Table 15-22: TIMER_BCAST_PER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	Broadcast Period Value.

Broadcast Width Register

For timers with `TIMER_TMRn_CFG.BWIDEN` enabled, a write to `TIMER_BCAST_WID` concurrently updates the' width (`TIMER_TMRn_WID`) registers of only those timers. A read of `TIMER_BCAST_WID` returns 0x00000000, and no bus error is generated. To read back a written value, read that TMR's `TIMER_TMRn_WID` register.

TIMER_BCAST_WID: Broadcast Width Register - R/WA

Reset = 0x0000 0000

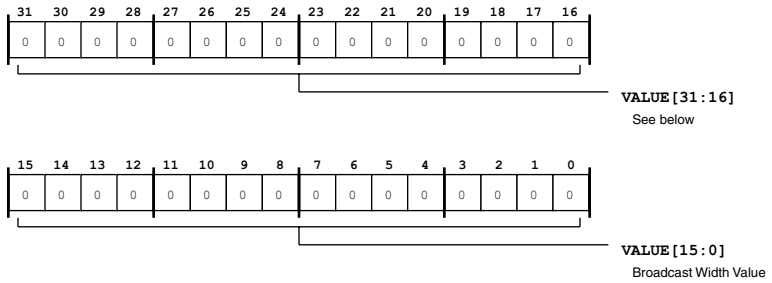


Figure 15-24: TIMER_BCAST_WID Register Diagram

Table 15-23: TIMER_BCAST_WID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	Broadcast Width Value.

Broadcast Delay Register

For timers with TIMER_TMRn_CFG.BDLYEN enabled, a write to TIMER_BCAST_DLY concurrently updates the delay (TIMER_TMRn_DLY) registers of only those timers. A read of TIMER_BCAST_DLY returns 0x00000000, and no bus error is generated. To read back a written value, read that TMR's TIMER_TMRn_DLY register.

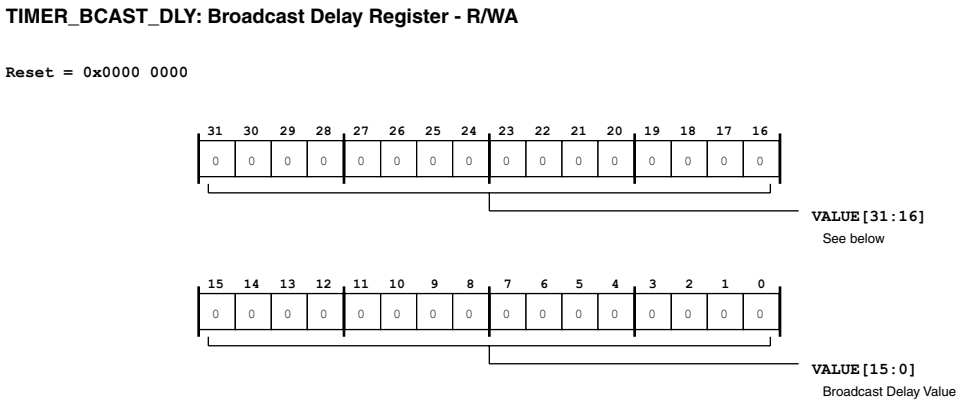


Figure 15-25: TIMER_BCAST_DLY Register Diagram

Table 15-24: TIMER_BCAST_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	Broadcast Delay Value.

Timer n Configuration Register

Each timer has a TIMER_TMRn_CFG register that specifies its operating mode. Only write to a TIMER_TMRn_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_TMRn_CFG register.

Note that a timer's TIMER_TMRn_CFG register may be read at any time.

TIMER_TMRn_CFG: Timer n Configuration Register - R/W

Reset = 0x0000

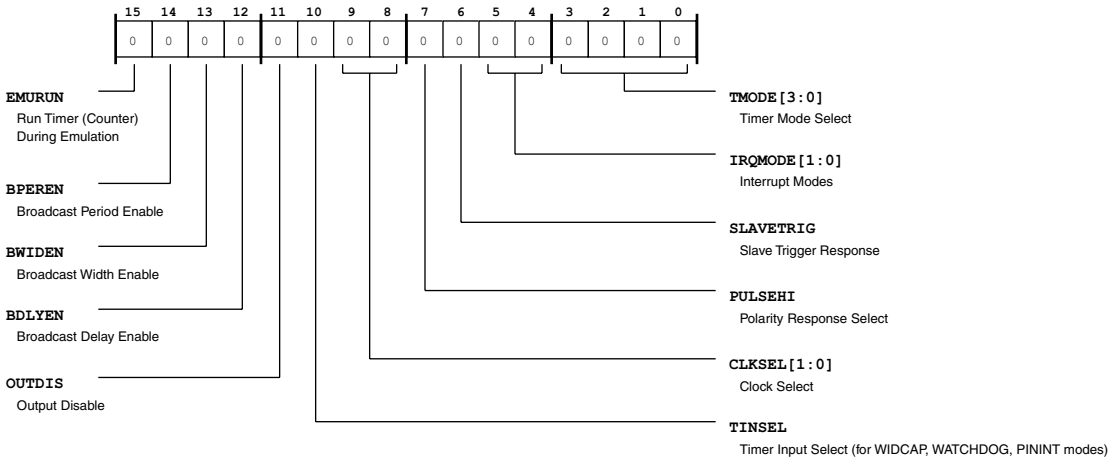


Figure 15-26: TIMER_TMRn_CFG Register Diagram

Table 15-25: TIMER_TMRn_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	EMURUN	Run Timer (Counter) During Emulation.	
		0	Stop Timer During Emulation
		1	Run Timer During Emulation
14 (R/W)	BPEREN	Broadcast Period Enable.	
		0	Disable Broadcast to PER Register
		1	Enable Broadcast to PER Register
13 (R/W)	BWIDEN	Broadcast Width Enable.	
		0	Disable Broadcast to WID Register
		1	Enable Broadcast to WID Register
12 (R/W)	BDLYEN	Broadcast Delay Enable.	
		0	Disable Broadcast to DLY Register
		1	Enable Broadcast to DLY Register
11 (R/W)	OUTDIS	Output Disable.	
		0	Enable TMR pin output buffer
		1	Disable TMR pin output buffer

Table 15-25: TIMER_TMRn_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	TINSEL	Timer Input Select (for WIDCAP, WATCHDOG, PININT modes).	
		0	Use TMR pin input
		1	Use TMR Alternate Capture Input
9:8 (R/W)	CLKSEL	Clock Select.	
		0	Use SCLK
		1	Use TMR_ALT_CLK0 as the TMR clock
		3	Use TMR_ALT_CLK1 as the TMR clock
7 (R/W)	PULSEHI	Polarity Response Select.	
		0	Negative Response/Pulse Negative Edge Response on TMR pin or Alternate Capture pin causes interrupt (PININT, EXTCLK modes) Negative Action Pulse on TMR pin or Alternate Capture pin causes interrupt (PWMOUT, WATCHDOG, WIDCAP modes)
		1	Positive Response/Pulse Positive Edge Response on TMR pin or Alternate Capture pin causes interrupt (PININT, EXTCLK modes) Positive Action Pulse on TMR pin or Alternate Capture pin causes interrupt (PWMOUT, WATCHDOG, WIDCAP modes)
6 (R/W)	SLAVETRIG	Slave Trigger Response. Note that 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 15-25: TIMER_TMRn_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W)	IRQMODE	Interrupt Modes. The TIMER_TMRn_CFG.IRQMODE selects the interrupt request mode. Note that any mismatched combination of TIMER_TMRn_CFG.IRQMODE and TIMER_TMRn_CFG.TMODE results in no interrupt being generated. Also note that in WIDCAP modes, the position of the interrupt is controlled with the TIMER_TMRn_CFG.TMODE bit, and the TIMER_TMRn_CFG.IRQMODE bit is ignored.	
		0	Active Edge Mode The timer generates an interrupt at every active edge. The active edge polarity depends on the state of the TIMER_TMRn_CFG.PULSEHI bit). Valid for PININT mode only.
		1	Delay Expired Mode The timer generates an interrupt when the TIMER_TMRn_CNT value reaches the value in the TIMER_TMRn_DLY register. This mode is valid for all PWMOUT modes.
		2	Width Plus Delay Expired Mode The timer generates an interrupt when the TIMER_TMRn_CNT value reaches the value in the TIMER_TMRn_WID register plus the value in the TIMER_TMRn_DLY register. (PWMOUT modes only). The timer generates an interrupt if the de-asserting edge is within the specified window. (WATCHDOG modes only.)
		3	Period Expired Mode The timer generates an interrupt when the TIMER_TMRn_CNT value reaches the value in the TIMER_TMRn_PER register. (Continuous PWMOUT and EXTCLK modes only.) The timer generates an interrupt if the de-asserting edge is outside the specified window.(WATCHDOG modes only.)

Table 15-25: TIMER_TMRn_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/W)	TMODE	Timer Mode Select.	
		0000 - 0111	Idle Mode
		8	Period Watchdog Mode
		9	Width Watchdog Mode
		10	Width Capture Asserted Mode Measurement report at asserting edge of waveform
		11	Width Capture Deasserted Mode Measurement report at de-asserting 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_TMRn_CNT 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_TMRn_CNT is read only and may be read at any time (whether the timer is running or stopped). Reading TIMER_TMRn_CNT 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, TMR_ALT_CLK0, or TMR_ALT_CLK1. The counter retains its value after the timer is disabled.

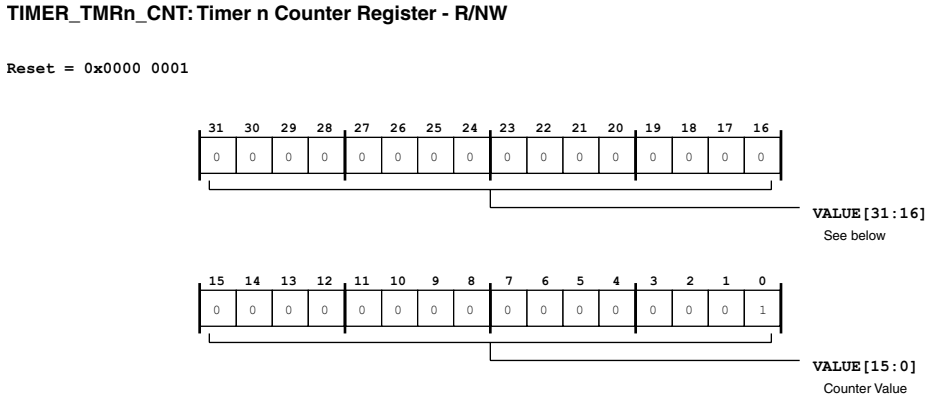


Figure 15-27: TIMER_TMRn_CNT Register Diagram

Table 15-26: TIMER_TMRn_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Counter Value.

Timer n Period Register

The TIMER_TMRn_PER holds the period value for the corresponding timer. Has different uses depending on the timer mode.

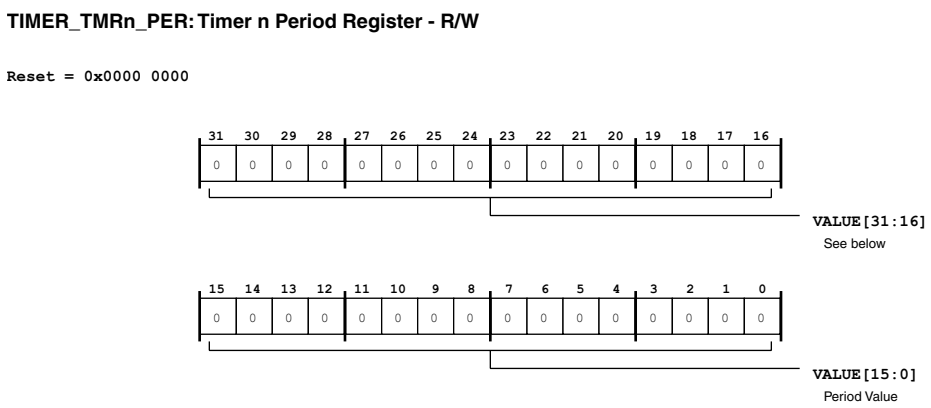


Figure 15-28: TIMER_TMRn_PER Register Diagram

Table 15-27: TIMER_TMRn_PER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Period Value.

Timer n Width Register

The TIMER_TMRn_WID holds the width value for the corresponding timer. Has different uses depending on the timer mode.

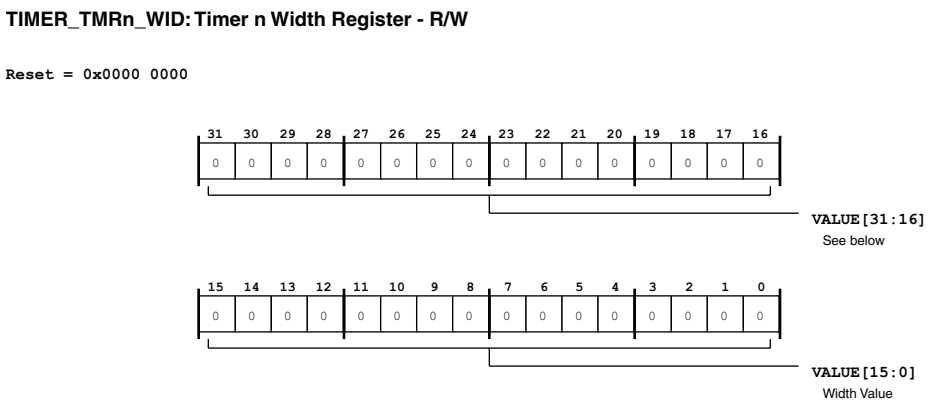


Figure 15-29: TIMER_TMRn_WID Register Diagram

Table 15-28: TIMER_TMRn_WID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Width Value.

Timer n Delay Register

The TIMER_TMRn_DLY holds the delay value for the corresponding timer. Has different uses depending on the timer mode.

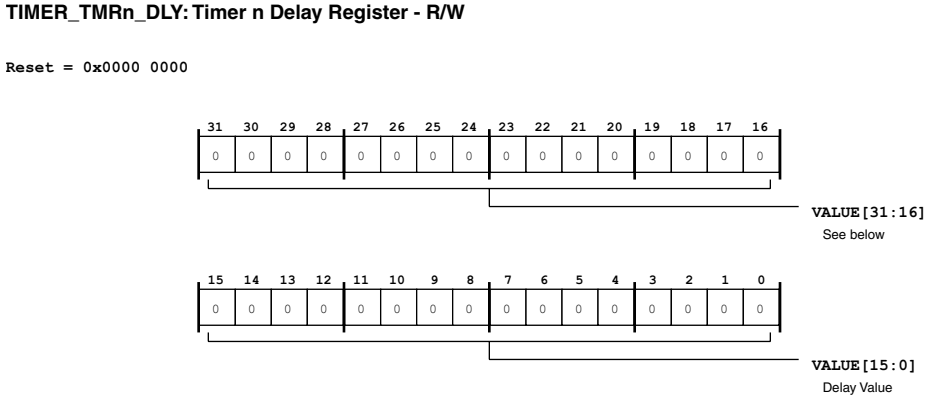


Figure 15-30: TIMER_TMRn_DLY Register Diagram

Table 15-29: TIMER_TMRn_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Delay Value.

16 Watchdog Timer (WDOG)

The processor includes a 32-bit timer for each core that can be used to implement a software watchdog function. A software watchdog can improve system reliability by generating an event to the processor core if the watchdog expires before being updated by software. The watchdog timers are clocked by the system clock (*SCLK*).

WDOG Features

The watchdog timer has the following features.

- Two identical 32-bit watchdog timers
- 8-bit disable bit pattern
- Can generate a general-purpose event for the core

Typically, the watchdog timer is used to supervise stability of the system software. When used in this way, software reloads the watchdog timer in a regular manner so that the downward counting timer never expires (never becomes 0). An expiring timer then indicates that system software might be out of control. At this point, based on the GP event generated by the WDOG, it is often better to reset and reboot the system using the Reset Control Unit.

For easier debugging, the watchdog timer does not decrement (even if enabled) when the processor is in emulation mode.

NOTE: The emulation event is controlled by the SDU (System Debug Unit). Please refer to the SDU chapter for more details.

Watchdog Timer Functional Description

When enabled, the 32-bit watchdog timer counts downward every `SCLK` cycle. When the count becomes 0, the expiry event is generated. This generates an GP event to the processor. When the event is generated, the core and the peripherals need to be reset using the software. The counter value can be read through the 32-bit `WDOG_STAT` register. The `WDOG_STAT` register cannot, however, be written directly. Rather, software writes the watchdog period value into the 32-bit `WDOG_CNT` register before the watchdog is enabled. Once the watchdog is started, the period value cannot be altered.

ADSP-BF60x WDOG Register List

Table 16-1: ADSP-BF60x WDOG Register List

Name	Description
WDOG_CTL	Control Register
WDOG_CNT	Count Register
WDOG_STAT	Watchdog Timer Status Register

ADSP-BF60x WDOG Interrupt List

Table 16-2: ADSP-BF60x WDOG Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
WDOG0 Expiration	2		LEVEL
WDOG1 Expiration	3		LEVEL

WDOG Block Diagram

The following figure shows the detailed watchdog timer block diagram.

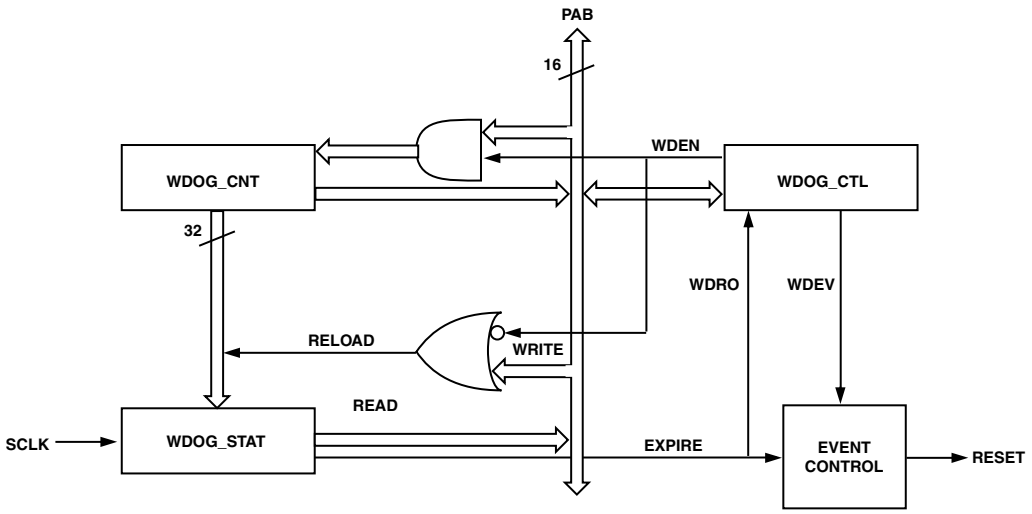


Figure 16-1: Watchdog Timer Block Diagram

Internal Interface

The watchdog timer does not directly interact with any pins of the chip.

External Interface

The watchdog timer is clocked by the system clock (*SCLK*) and its registers are accessed through the 16-bit peripheral MMR access bus. The 32-bit WDOG_CNT and WDOG_STAT registers must always be accessed by 32-bit read/write operations. Hardware ensures that those accesses are atomic. When the counter expires, the GP expiration event is generated.

WDOG Configuration

PREREQUISITE:

To start the watchdog timer, use the following procedure.

1. Set the count value for the watchdog timer by writing the count value into the watchdog count register (WDOG_CNT). Note that loading the WDOG_CNT register while the timer is not enabled also pre-loads the WDOG_STAT register.
2. Enable the watchdog timer by writing to the WDOG_CTL.WDEN bit field. The watchdog timer then begins counting down, decrementing the value in the WDOG_STAT register. When the WDOG_STAT reaches 0, the expiration event is generated.

   ADDITIONAL INFORMATION: To prevent the event from being generated, software must reload the count value from the WDOG_CNT register to the WDOG_STAT register by executing a write (of any value) to the WDOG_STAT register, or must disable the watchdog timer in the WDOG_CTL register before the watchdog timer expires.

   a. If software does not serve the watchdog in time, the WDOG_STAT register continues decrementing until it reaches 0 at which point it generates a GP interrupt (if enabled) and the software can perform a core and/or a system reset.

   ADDITIONAL INFORMATION: Once the counter reaches 0, it stops decrementing and remains at 0. Additionally, the WDOG_CTL.WDRO bit is set.

   b. If the watchdog is enabled with a zero value loaded to the counter and the WDOG_CTL.WDRO bit was cleared, the WDOG_CTL.WDRO bit of the watchdog control register is set immediately and the counter remains at zero without further decrements. If, however, the WDOG_CTL.WDRO bit was set by the time the watchdog is enabled, the counter decrements to 0xFFFF FFFF and continues operation.

   ADDITIONAL INFORMATION: Software can disable the watchdog timer only by writing a 0xAD value to the WDOG_CTL.WDEN bit field.

ADSP-BF60x WDOG Register Descriptions

Watch Dog Timer Unit (WDOG) contains the following registers.

Table 16-3: ADSP-BF60x WDOG Register List

Name	Description
WDOG_CTL	Control Register
WDOG_CNT	Count Register
WDOG_STAT	Watchdog Timer Status Register

Control Register

The WDOG_CTL register controls the watch dog timer. This register supports enabling/disabling the watch dog timer and supports checking the timer rollover status. Note that when the processor is in emulation mode, the watch dog timer counter will not decrement even if it is enabled.

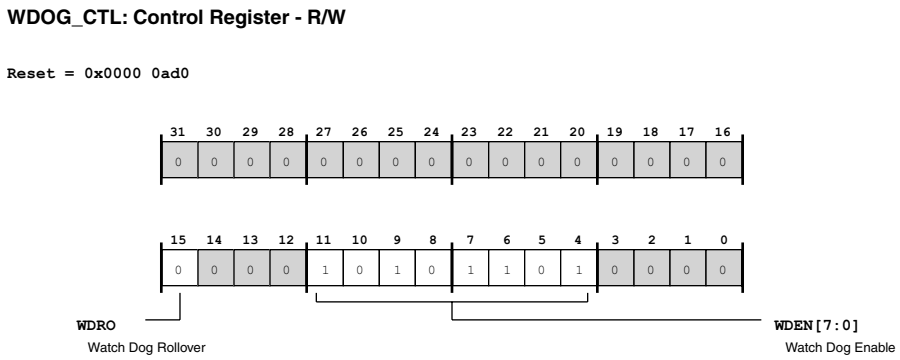


Figure 16-2: WDOG_CTL Register Diagram

Table 16-4: WDOG_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	WDRO	Watch Dog Rollover. Software can determine whether the timer has rolled over by interrogating the WDOG_CTL.WDRO 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

Table 16-4: WDOG_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11:4 (R/W)	WDEN	Watch Dog Enable. The WDOG_CTL.WDEN field is used to enable and disable the watch dog timer. Writing any value other than the disable value into this field enables the watch dog timer. This multi-bit disable key minimizes the chance of inadvertently disabling the watch dog timer.	
		173	Counter Disabled All other values - counter enabled

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.

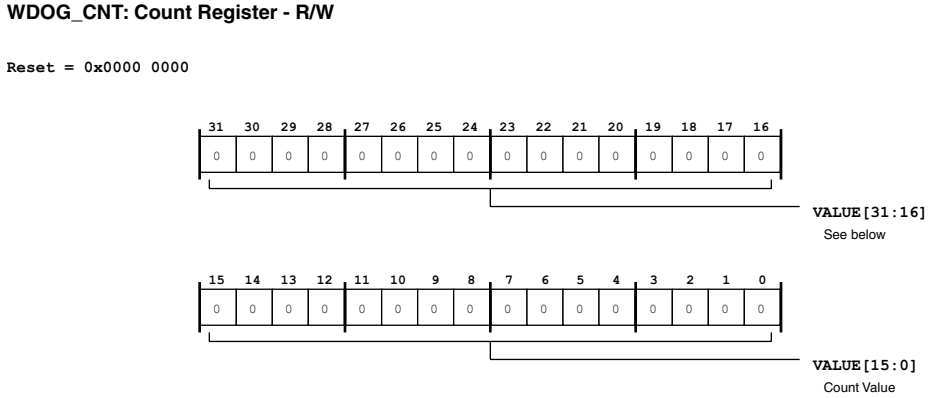


Figure 16-3: WDOG_CNT Register Diagram

Table 16-5: WDOG_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Count Value.

Watchdog Timer Status Register

The WDOG_STAT contains the current count value of the watch dog timer. Reads of this register return the current count value. When the watch dog timer is enabled, WDOG_STAT is decremented by 1 on each SCLK

cycle. When count value reaches 0, the watch dog timer stops counting, and the expiry event is generated. WDOG_STAT is a 32-bit unsigned system MMR that must be accessed with 32-bit reads and writes.

Values cannot be stored directly in WDOG_STAT, but are instead copied from the WDOG_CNT register. This copy process can happen in two ways:

- While the watch dog timer is disabled, writing the WDOG_CNT register pre-loads the WDOG_STAT register.
- While the watch dog timer is enabled, writing the WDOG_STAT register loads it with the value in WDOG_CNT.

When the processor executes a write (of an arbitrary value) to WDOG_STAT, the value in WDOG_CNT is copied into WDOG_STAT. Typically, software sets the value of WDOG_CNT at initialization, then periodically writes to WDOG_STAT before the watch dog timer expires. This reloads the watch dog timer with the value from WDOG_CNT and prevents generation of the expiry event.

If the user does not reload the counter before SCLK*Count register cycles, an expiry event is generated, and the WDOG_CTL.WDRO bit is set. When this happens, the counter will stop decrementing and will remain 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.

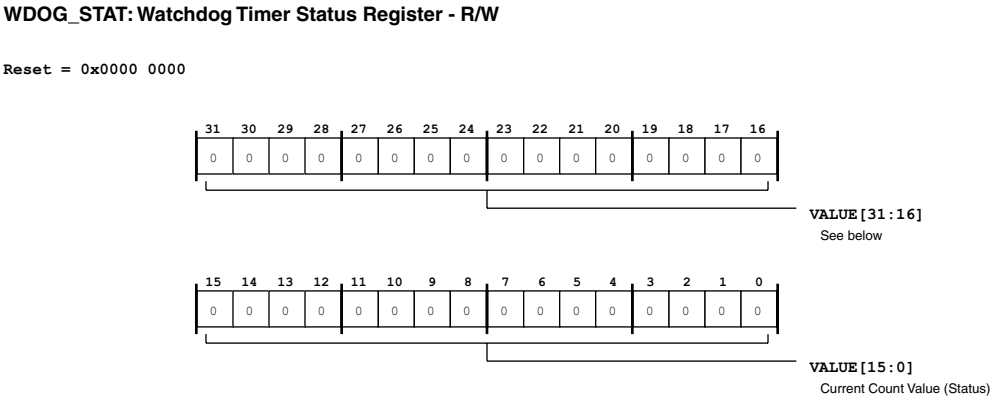


Figure 16-4: WDOG_STAT Register Diagram

Table 16-6: WDOG_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Current Count Value (Status).

17 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 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) that can be used to establish a reference point to verify that the acquired position does not drift over time. In addition, the incremental position information can be used to determine speed, if the time intervals are measured.

The GP counter provides flexible ways to establish position information. When used in conjunction with the GP timer block, the GP counter may allow for the acquisition of coherent position/time-stamp information that enables speed calculation.

GP Counter Features

The GP Counter includes the following features:

- 32-bit up/down counter
- Quadrature encode mode (Gray code)
- Binary encoder mode
- Alternative frequency-direction mode
- Timed direction and up/down counting modes
- Zero marker/push button support
- Capture event timing in association with GP Timer
- Boundary comparison and boundary setting features

CNT Functional Description

A block diagram of the GP counter is shown below. The `CNT_UD` and `CNT_DG` pins accept various forms of incremental inputs and are processed by the 32-bit counter, while the `CNT_ZM` pin can be used 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 may be filtered (debounced) before being evaluated by the GP counter.

The GP counter also features a flexible boundary comparison. In all of the operating modes, the counter can be compared to an upper and lower limit. A variety of actions can be taken when these limits are reached.

The module can optionally generate an interrupt request to the system through its IRQ line. On many processors, there is also an output that can be used by a GP timer module to generate timestamps on certain events.

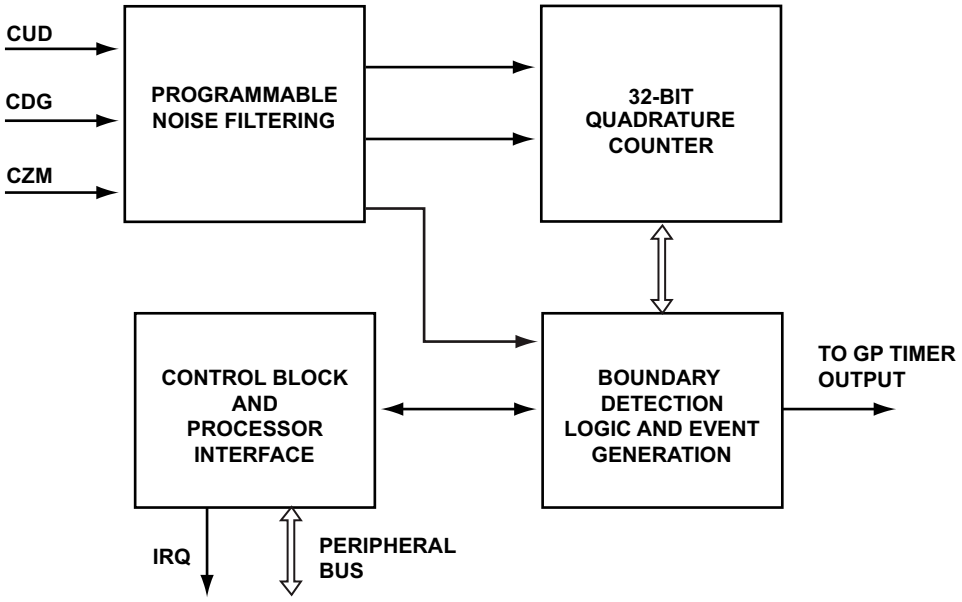


Figure 17-1: GP Timer Block Diagram

ADSP-BF60x CNT Register List

The 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 (counting) pulses on one or two inputs. Because integration provides relative position, some devices also feature a zero position input (zero marker) 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 time intervals are measured. The CNT provides flexible ways to establish position information. When used in with the 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 17-1: ADSP-BF60x CNT Register List

Name	Description
CNT_CFG	Configuration Register
CNT_IMSK	Interrupt Mask Register
CNT_STAT	Status Register
CNT_CMD	Command Register
CNT_DEBNCE	Debounce Register
CNT_CNTR	Counter Register
CNT_MAX	Maximum Count Register
CNT_MIN	Minimum Count Register

ADSP-BF60x CNT Interrupt List

Table 17-2: ADSP-BF60x CNT Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
CNT0 Status	27		LEVEL

ADSP-BF60x CNT Trigger List

Table 17-3: ADSP-BF60x CNT Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
CNT0 Status	16	LEVEL

Table 17-4: ADSP-BF60x CNT Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
None		

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 table below. Optionally, an interrupt is generated if both inputs change within one SCLK cycle. Such transitions are not allowed by Gray coding. Therefore, the CNT_CNTR register remains unchanged, and an error condition is signaled.

Table 17-5: 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. This feature can be enabled with the CNT_CFG.CDGINV and CNT_CFG.CUDINV bits.

As an example, if the CNT_DG and CNT_UD inputs are 00 and the next transition is to 01. This normally increments the counter as is shown in the table. If the CNT_UD polarity is inverted, this generates a received input of 01 followed by 00. This results in 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 tran-

sitions that have occurred as shown in the following table. Optionally, an interrupt is generated if the detected code steps by more than 1 (in binary arithmetic) within one SCLK cycle. Such transitions are considered erroneous. Therefore, the CNT_CNTR register remains unchanged, and an error condition is signaled.

Table 17-6: 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 is incremented or decremented at every active edge of the input pins. The active edge can be selected by the CNT_CFG.CUDINV bit and has the following results.

- If an active edge is detected at the CNT_UD input, the counter increments.
- If an active edge is detected 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 and the polarity is selected by the CNT_CFG.CUDINV bit.

If an active edge is detected at the CNT_DGand the active edge is selected by the CNT_CFG.CDGINV

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 polarity can be selected by the CNT_CFG.CUDINV bit. The CNT_DG pin can be used to gate the clock. The polarity can be selected by the CNT_CFG.CDGINV bit.

CNT Event Control

Eleven events can be signaled to the processor using status information and optional interrupt requests. The interrupts are enabled by the respective bits in the CNT_IMSK register. Dedicated bits in the CNT_STAT register report events. When an interrupt from the GP counter is acknowledged, 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 interrupts.

Interrupt requests are cleared by write-one-to-clear (W1C) operations to the CNT_STAT register. 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/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, an interrupt request is generated. The CNT_STAT.IC bit should only be set in the quadrature encoder or binary encoder modes.

Up/Down Count Events

The CNT_STAT.UC bit indicates whether the counter has been incremented. Similarly, the CNT_STAT.DC bit reports decrements. The two events are independent. For instance, if the counter first 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 may 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 may be enabled through the CNT_IMSK.UC and CNT_IMSK.DC bits. This feature should be used carefully when the counter is clocked at high rates. This 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 additional push buttons, if GP counter features are not needed. When up/down counter mode is enabled, these count events can be used 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, an interrupt request is generated.

Overflow Events

There are two status bits that indicate whether the signed counter register has overflowed from a positive to a negative value or vice versa. 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 0xXXXX 7FFF to 0xXXXX 8000, or from 0xXXXX 8000 to 0xXXXX 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 CNT_CNTR, CNT_MAX, or CNT_MIN registers are updated by software or the CNT_CMD register is written to. The CNT_IMSK.MINC and CNT_IMSK.MAXC bits enable interrupt 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, if enabled by the corresponding CNT_IMSK.CZM, CNT_IMSK.CZME and CNT_IMSK.CZMZ bits.

GP Counter Programming Model

The following sections provide information used to assist in programming the interface.

CNT 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 via the CNT_CFG register.
2. Usually, events of interest are processed using interrupts rather than by polling status bits. If this is the 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 signalling to the system event controller.

4. If timing information is required, set up the relevant GP Timer in width capture mode.
5. Finally, enable interrupts and the GP Counter itself using the CNT_IMSK and CNT_CFG registers, respectively.

CNT Mode Configuration

The CNT_ZM input pin can be used 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, and zero-once mode.

Configuring GP Counter Push-Button Operation

Use the following procedure to configure push-button operation

1. Set CNT_IMSK.CZME to enable (unmask) the zero marker error 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.

RESULT:

An active edge at the CNT_ZM input sets the CNT_STAT.CZME 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 to be zeroed by a Zero Marker interrupt.
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.

RESULT:

This 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

This mode is used 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.

RESULT:

When an active edge is detected at the CNT_ZM input pin, the four LSBs of the CNT_CNTR register are compared to zero. If they are not zero, a mismatch is signaled using the CNT_STAT.CZME bit.

Configuring Zero-Once Mode

This mode is used to perform an initial reset of the counter value when an active zero marker is detected. 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.

RESULT:

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, the boundary registers (CNT_MIN and CNT_MAX) are modified by hardware whenever the CNT_CNTR value reaches either of them. This mode can be used to keep track of 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.

RESULT:

The CNT_MAX register is loaded with the current CNT_CNTR value if the latter increments beyond the CNT_MAX value. Similarly, the CNT_MIN register is loaded with the CNT_CNTR value if 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.

RESULT:

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.

RESULT:

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. Note that the CNT_STAT.MAXC and CNT_STAT.MINC bits are not set if the CNT_MAX and/or CNT_MIN registers are updated by software.

Configuring GP Counter Push-Button Operation

Use the following procedure to configure push-button operation

1. Set CNT_IMSK.CZME to enable (unmask) the zero marker error 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.

RESULT:

An active edge at the CNT_ZM input sets the CNT_STAT.CZME 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. This filtering can be enabled or disabled by the CNT_CFG.DEBEN bit. The following figure shows the filtering operation for the CNT_UD pin.

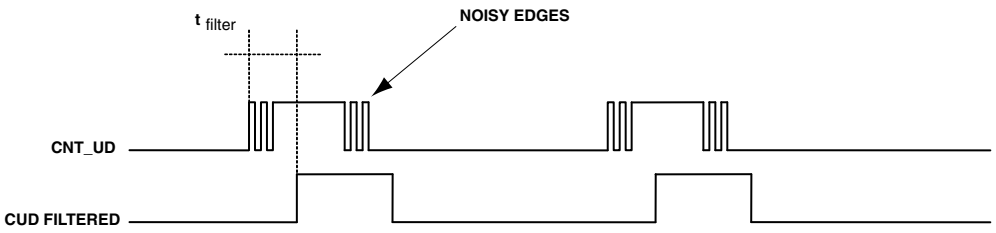


Figure 17-2: Programmable Noise Filtering

The filtering mechanism is implemented 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 time t_{filter} is determined, given SCLK and the CNT_DEBNCE.DPRESCALE value, by the following formula, where lower values of CNT_DEBNCE.DPRESCALE result in shorter debounce delays:

$$t_{filter} = 128 \times (2^{DPRESCALE} \times SCLK)$$

Capturing Counter Interval and CNT_CNTR Read Timing

When the count speed is very low, it is often useful to capture the time elapsed since the last count event. In order to do this, program the associated GP Timer's TIMER_TMRn_CFG register in a width capture mode with the following bit settings.

- TIMER_TMRn_CFG.PULSEHI = 0
- TIMER_TMRn_CFG.TMODE = b#1011
- TIMER_TMRn_CFG.TINSEL = 1

The following figure shows and the following list describes operation of the GP counter and the GP timer in this mode.

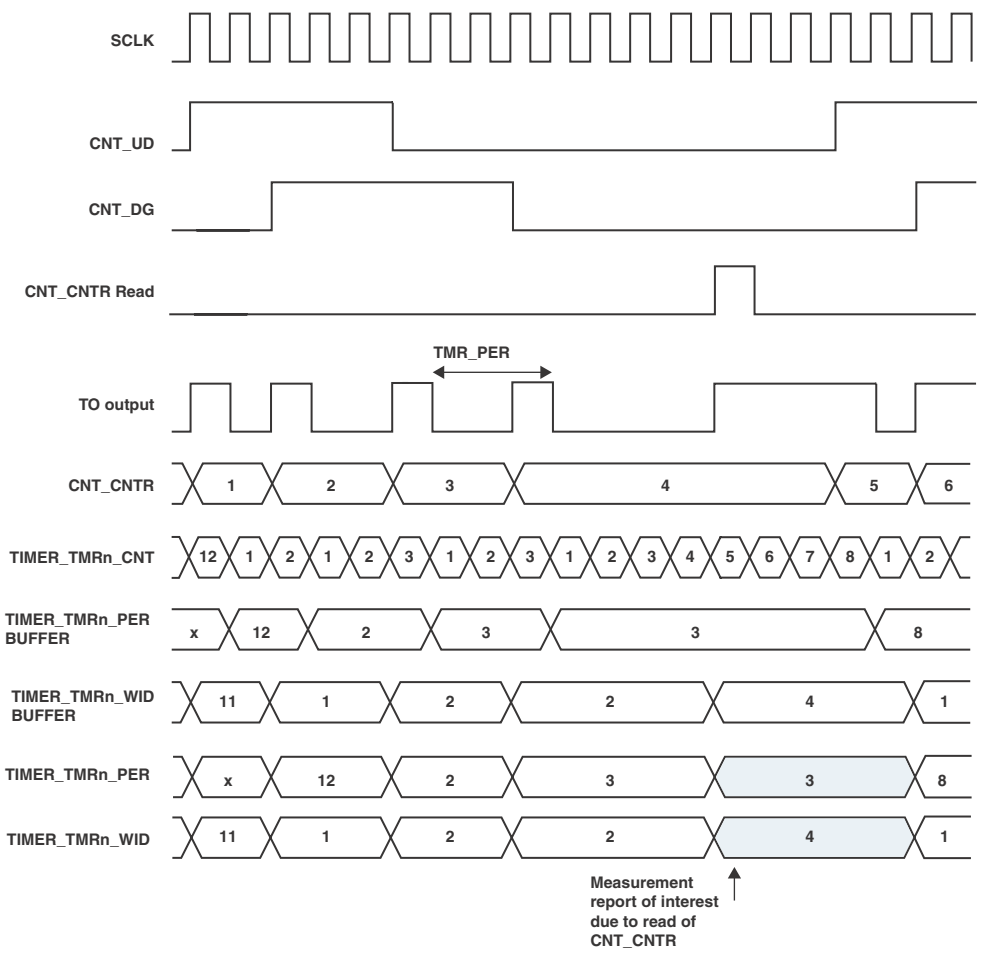


Figure 17-3: Capture Intervals

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_TMRn_PER register with the period (measured from falling edge to falling edge, because TIMER_TMRn_CFG.PULSEHI = 0) of the CNT_TO signal.
3. The TIMER_TMRn_WID register is updated with the pulse width (the portion where CNT_TO is low, again because TIMER_TMRn_CFG.PULSEHI = 0).
4. Both registers are updated at every rising edge of the CNT_TO signal (because TIMER_TMRn_CFG.TMODE= b#011).

Therefore, the TIMER_TMRn_PER register contains the period between the last two count events, and the TIMER_TMRn_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 very high count event rates. For instance, if CNT_CNTR is incremented/decremented every SCLK cycle (timed direction mode), the CNT_TO signal will not be valid.

Capturing Time Interval Between Successive Counter Events

When the required timing information is the interval between successive count events, the associated timer should be programmed in a width capture mode with TIMER_TMRn_CFG bit settings of TIMER_TMRn_CFG.PULSEHI = 1, TIMER_TMRn_CFG.TMODE = b#1010 and TIMER_TMRn_CFG.TINSEL = 1. Typically, this information is sufficient if the speed of GP counter events does not to reach very low values.

The following figure shows the operation of the GP counter and the GP timer in this mode.

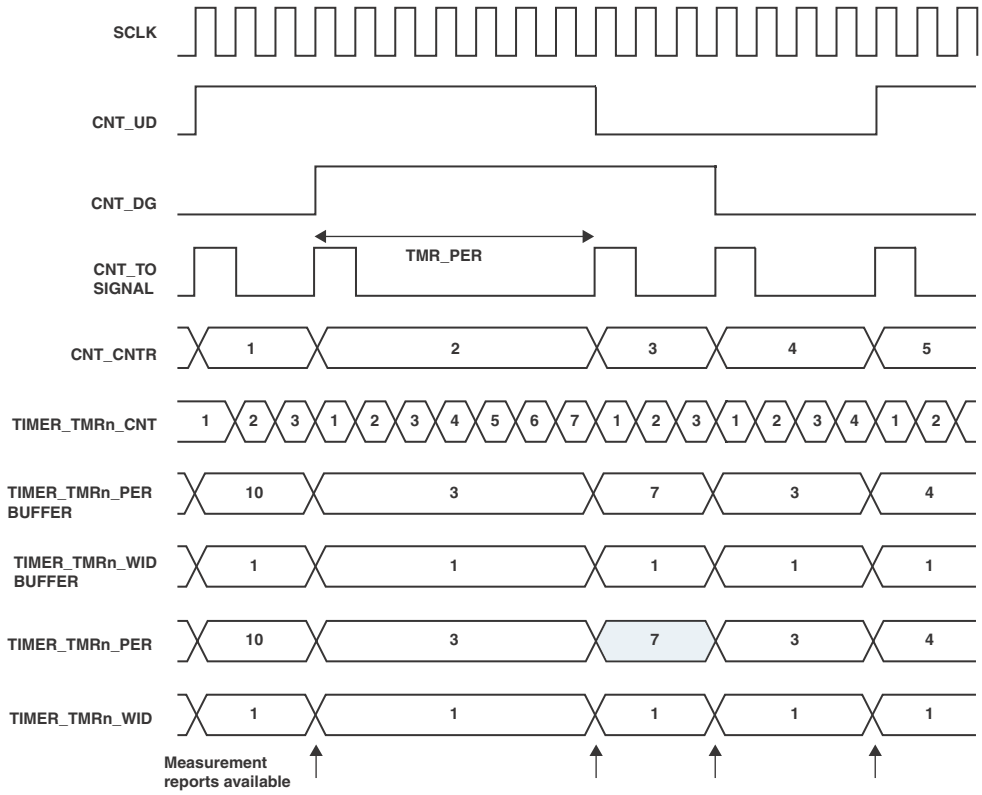


Figure 17-4: Period Register Timing

The CNT_TO signal generates a pulse every time a count event occurs. The GP timer updates its TIMER_TMRn_PER register with the period (measured from rising edge to rising edge) of the CNT_TO signal. The TIMER_TMRn_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_TMRn_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_TMRn_WID register only contains the width of the CNT_TO pulse. If a read of CNT_CNTR has occurred between events, the TIMER_TMRn_WID register contains the time between the read of CNT_CNTR and the next event.

This mode can also be used with TIMER_TMRn_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.

ADSP-BF60x CNT Register Descriptions

CNT (CNT) contains the following registers.

Table 17-7: ADSP-BF60x CNT Register List

Name	Description
CNT_CFG	Configuration Register
CNT_IMSK	Interrupt Mask Register
CNT_STAT	Status Register
CNT_CMD	Command Register
CNT_DEBNCE	Debounce Register
CNT_CNTR	Counter Register
CNT_MAX	Maximum Count Register
CNT_MIN	Minimum Count Register

Configuration Register

The CNT_CFG register configures counter modes, configures input pins, and enable the CNT.

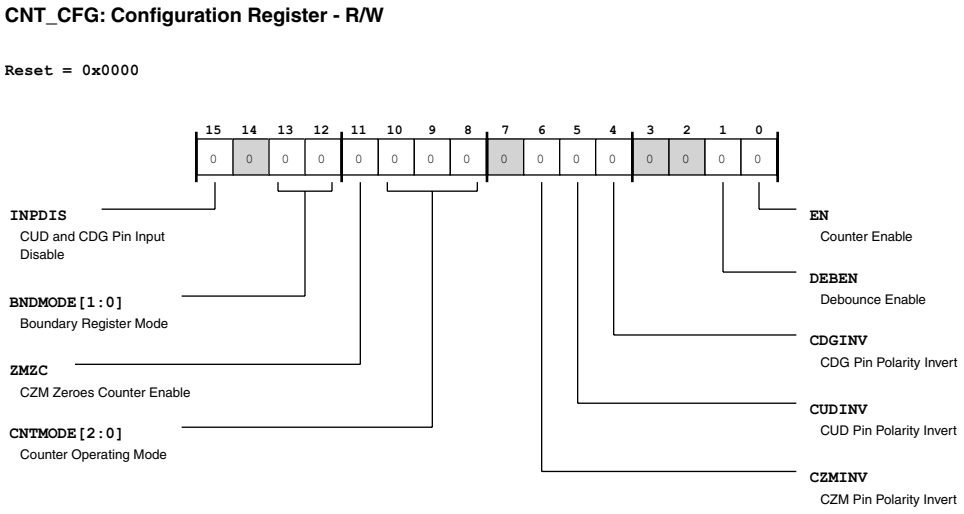


Figure 17-5: CNT_CFG Register Diagram

Table 17-8: CNT_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	INPDIS	CUD and CDG Pin Input Disable. The CNT_CFG.INPDIS disables or enables the CNT_UD input pin and the CNT_DG pin.	
		0	Enable
		1	Pin Input Disable
13:12 (R/W)	BNDMODE	Boundary Register Mode. The CNT_CFG.BNDMODE selects the mode for the CNT_MIN and CNT_MAX boundary registers.	
		0	BND_COMP Boundary compare mode
		1	BIN_ENC Binary encoder mode
		2	BND_CAPT Boundary capture mode
		3	BND_AEXT Boundary auto-extend mode
11 (R/W)	ZMZC	CZM Zeroes Counter Enable. The CNT_CFG.ZMZC enables or disables level sensitive - Active CNT_ZM pin operation to zero the CNT_CNTR.	
		0	Disable
		1	Enable

Table 17-8: CNT_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10:8 (R/W)	CNTMODE	Counter Operating Mode. The CNT_CFG.CNTMODE 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 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 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 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 17-8: CNT_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	DEBEN	Debounce Enable. The CNT_CFG.DEBEN 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 enables or disables CNT operation.	
		0	Counter Disable
		1	Counter Enable

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 (if bit cleared) or enable/unmask an interrupt (if bit set).

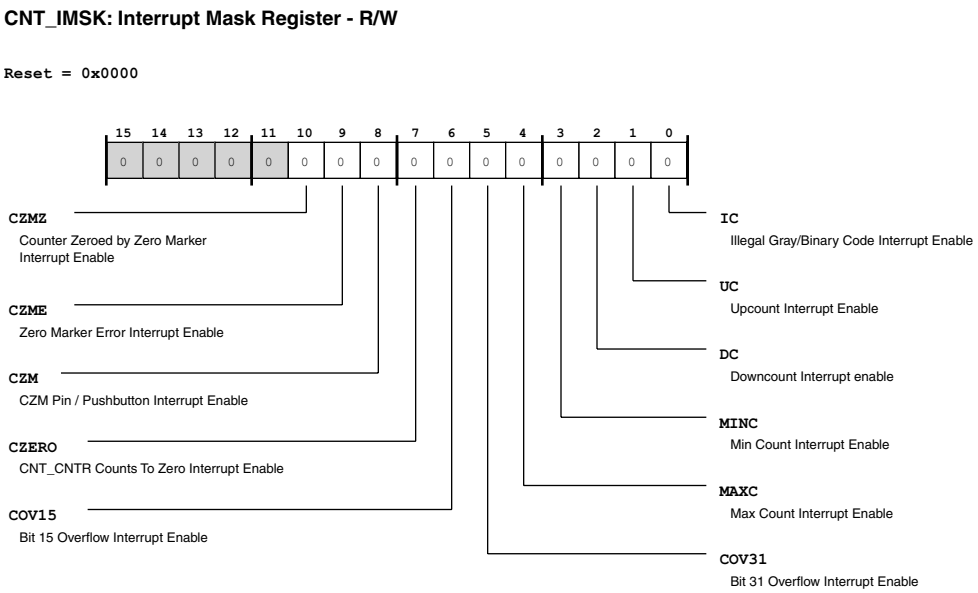


Figure 17-6: CNT_IMSK Register Diagram

Table 17-9: CNT_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	CZMZ	Counter Zeroed by Zero Marker Interrupt Enable.	
		0	Mask Interrupt
		1	Unmask Interrupt
9 (R/W)	CZME	Zero Marker Error Interrupt Enable.	
		0	Mask Interrupt
		1	Unmask Interrupt
8 (R/W)	CZM	CZM Pin / Pushbutton Interrupt Enable.	
		0	Mask Interrupt
		1	Unmask Interrupt
7 (R/W)	CZERO	CNT_CNTR Counts To Zero Interrupt Enable.	
		0	Mask Interrupt
		1	Unmask Interrupt
6 (R/W)	COV15	Bit 15 Overflow Interrupt Enable.	
		0	Mask Interrupt
		1	Unmask Interrupt
5 (R/W)	COV31	Bit 31 Overflow Interrupt Enable.	
		0	Mask Interrupt
		1	Unmask Interrupt
4 (R/W)	MAXC	Max Count Interrupt Enable.	
		0	Mask Interrupt
		1	Unmask Interrupt
3 (R/W)	MINC	Min Count Interrupt Enable.	
		0	Mask Interrupt
		1	Unmask Interrupt
2 (R/W)	DC	Downcount Interrupt enable.	
		0	Mask Interrupt
		1	Unmask Interrupt
1 (R/W)	UC	Upcount Interrupt Enable.	
		0	Mask Interrupt
		1	Unmask Interrupt

Table 17-9: CNT_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	IC	Illegal Gray/Binary Code Interrupt Enable.	
		0	Mask Interrupt
		1	Unmask Interrupt

Status Register

The CNT_STAT register provides status information for each of the CNT events. 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 CNT_STAT indicate either no interrupt pending (if bit cleared) or an interrupt pending (if bit set).

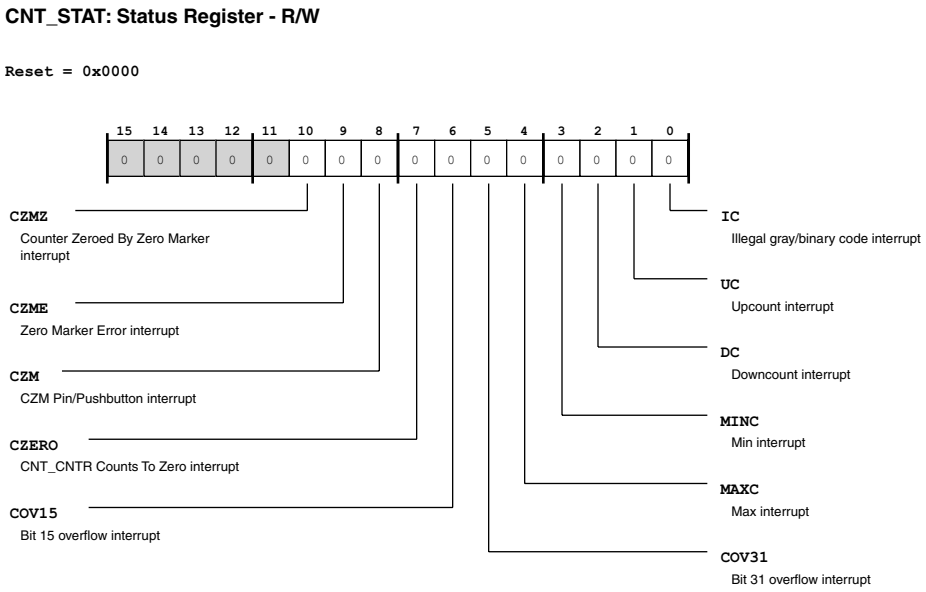


Figure 17-7: CNT_STAT Register Diagram

Table 17-10: CNT_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/W1C)	CZMZ	Counter Zeroed By Zero Marker interrupt.
9 (R/W1C)	CZME	Zero Marker Error interrupt.

Table 17-10: CNT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W1C)	CZM	CZM Pin/Pushbutton interrupt.
7 (R/W1C)	CZERO	CNT_CNTR Counts To Zero interrupt.
6 (R/W1C)	COV15	Bit 15 overflow interrupt.
5 (R/W1C)	COV31	Bit 31 overflow interrupt.
4 (R/W1C)	MAXC	Max interrupt.
3 (R/W1C)	MINC	Min interrupt.
2 (R/W1C)	DC	Downcount interrupt.
1 (R/W1C)	UC	Upcount interrupt.
0 (R/W1C)	IC	Illegal gray/binary code interrupt.

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

no zero marker event has 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, CNT_CMD permits modifying the CNT_MIN and CNT_MAX boundary registers in a number of ways. The current counter value in CNT_CNTR 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 vice versa) 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, but 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 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.

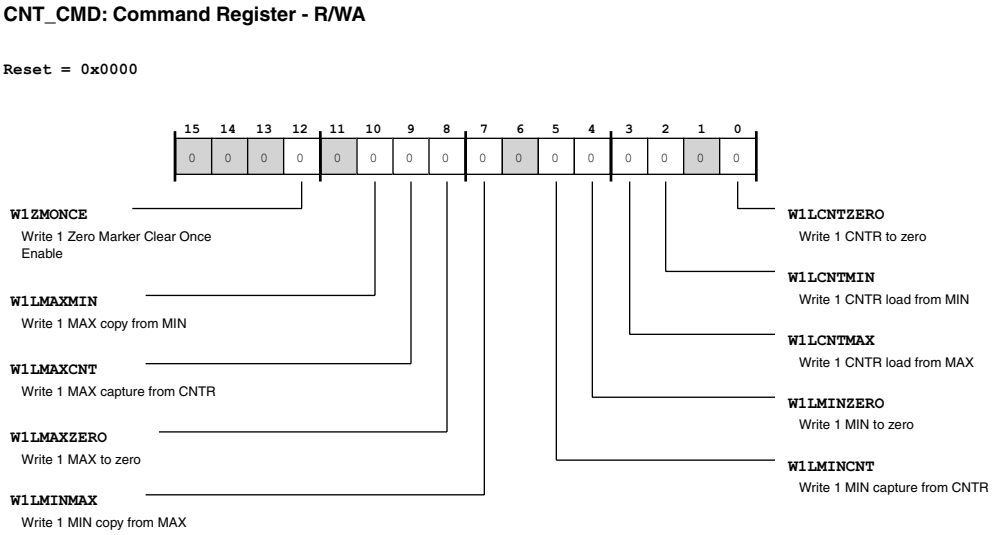


Figure 17-8: CNT_CMD Register Diagram

Table 17-11: CNT_CMD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W1A)	W1ZMONCE	Write 1 Zero Marker Clear Once Enable. The CNT_CMD.W1ZMONCE enables a single zero marker clear of CNT_CNTR. 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/W1A)	W1LMAXMIN	Write 1 MAX copy from MIN.
9 (R0/W1A)	W1LMAXCNT	Write 1 MAX capture from CNTR.

Table 17-11: CNT_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R0/W1A)	W1LMAXZERO	Write 1 MAX to zero.
7 (R0/W1A)	W1LMINMAX	Write 1 MIN copy from MAX.
5 (R0/W1A)	W1LMINCNT	Write 1 MIN capture from CNTR.
4 (R0/W1A)	W1LMINZERO	Write 1 MIN to zero.
3 (R0/W1A)	W1LCNTMAX	Write 1 CNTR load from MAX.
2 (R0/W1A)	W1LCNTMIN	Write 1 CNTR load from MIN.
0 (R0/W1A)	W1LCNTZERO	Write 1 CNTR to zero.

Debounce Register

The CNT_DEBNCE register selects the noise filtering characteristic of the three input pins according to the formula:

$t_{filter} = 128 \times (2^{DPRESCALE} / SCLK)$

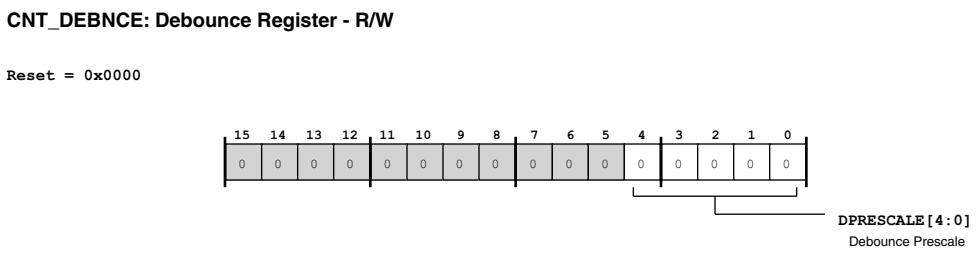


Figure 17-9: CNT_DEBNCE Register Diagram

Table 17-12: CNT_DEBNCE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4:0 (R/W)	DPRESCALE	Debounce Prescale. The CNT_DEBNCE.DPRESCALE 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
		17	131072x cycles
		18	Reserved from this value 10010 - 11111: Reserved
		31	Reserved till this value

Counter Register

The CNT_CNTR register holds the 32-bit, twos-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 CNTas a 16-bit counter if sufficient for the application.

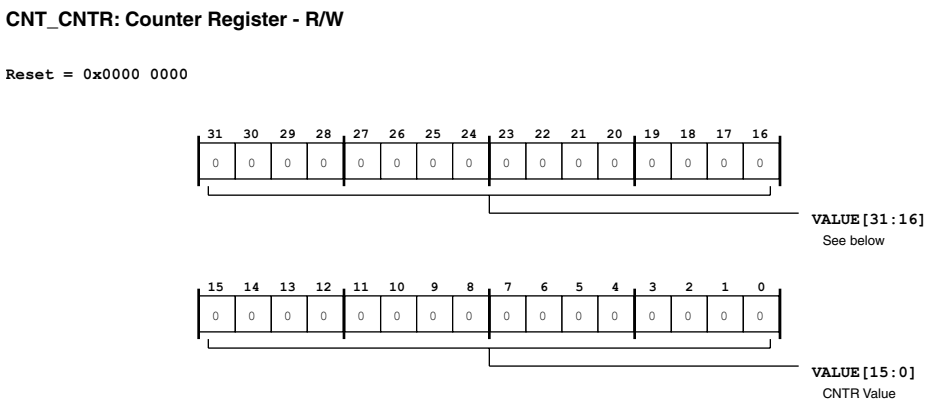


Figure 17-10: CNT_CNTR Register Diagram

Table 17-13: CNT_CNTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	CNTR Value.

Maximum Count Register

The CNT_MAX register holds the 32-bit, twos-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.

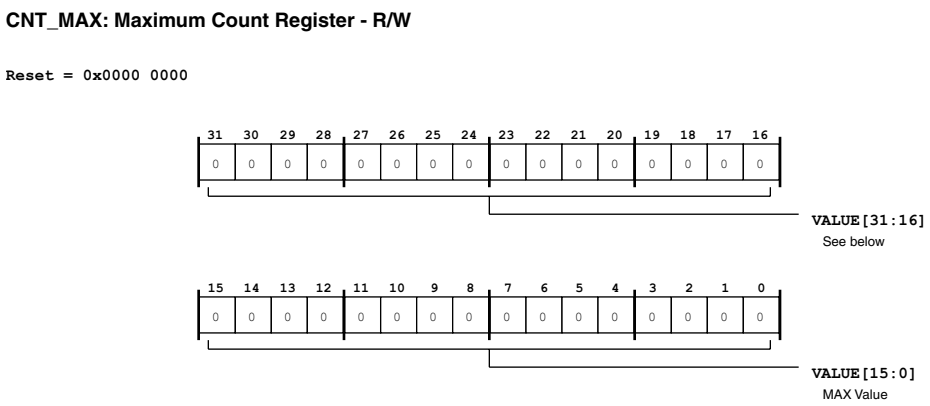


Figure 17-11: CNT_MAX Register Diagram

Table 17-14: CNT_MAX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MAX Value.

Minimum Count Register

The CNT_MIN register holds the 32-bit, twos-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.

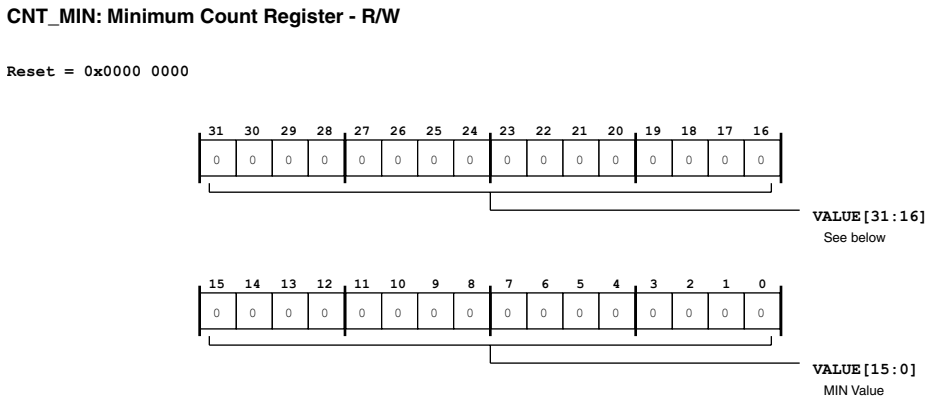


Figure 17-12: CNT_MIN Register Diagram

Table 17-15: CNT_MIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MIN Value.

18 Pulse-Width Modulator (PWM)

The Pulse Width Modulator (PWM) module is a flexible and programmable waveform generator. With minimal CPU intervention the PWM peripheral is capable of generating complex waveforms for motor control, Pulse Coded Modulation (PCM), Digital to Analog Conversion (DAC) functionality, power switching and power conversion. The PWM module has 4 PWM pairs capable of 3-phase PWM generation for source inverters for AC induction and DC brush less motors.

PWM Features

The two 3-phase PWM generation units each feature:

- 16-bit center-based PWM generation unit
- Programmable PWM pulse width
- Single/double update modes
- Programmable dead time and switching frequency
- Twos-complement implementation which permits smooth transition to full ON and full OFF states
- Dedicated asynchronous PWM shutdown signal

Functional Description

Each PWM Module has 4 PWM pairs with 2 outputs: a high and a low side signal.

During initialization, the `PWM_TM0` register is written to define the PWM period, and the Channel Pulse Duty registers are written to define the initial channel pulse widths. The `PWM_CTL` and `PWM_CHANCFG` registers are written, depending on the system configuration and modes. The `PWM_STAT` register can be read to determine polarity, and whether switched reluctance (SR) mode (PWM_SR bit) is enabled, and whether an external trip situation is preventing the correct start-up of the PWM Controller. An active external trip event must be resolved prior to PWM startup. The `PWM_CTL`register is then written to define the major operating mode and to enable the PWM outputs and PWM sync pulse.

During the PWM_SYNCINT interrupt driven control loop, only the `PWM_AH0.DUTY` values are updated typically.

Typically the PWMx_TMR interrupt is used to periodically execute an Interrupt Service Routine (ISR), to update the PWM channel control and duty registers according to a control algorithm based on expected

operation and sampled existing operation. The PWMx_TMR interrupt also can trigger an ADC to sample data for use during the ISR.

During processor boot the PWM is initialized and the program flow enters a wait loop. When a PWM_SYNC interrupt occurs, the ADC samples data, the data is algorithmically interpreted, and new PWM channel duties are calculated and written to the PWM. More sophisticated implementations include different startup, runtime, and shutdown algorithms to determine PWM channel duties based on expected behavior and further features.

During initialization, the PWM_TMO registers are written to define the PWM periods of different timers. The PWM_DLYA registers are written to define the initial phase difference between the main timer and the other timers. The PWM_CHANCFG and PWM_ACTL registers are written to define the initial pulse configuration. The Channel x High/Low Duty registers are programmed to define the initial duty values. The requisite dead-time must be written into the PWM_DT register. The PWM_CTL register is written in the end with the final configurations and enable bit for the entire PWM (PWM_CTL.GLOBEN).

The PWM_SYNC interrupt is assigned to one of the core's user interrupts. During the PWM_SYNC interrupt driven control loop, only the PWM_DLYA, the duty registers and channel C high pulse duty register values are typically updated. To see programming limitations on the PWM registers, see the Register Descriptions section.

During any external trip event (if not disabled), the PWM outputs will be turned off. When a PWM output is turned off, it means that the output level is held at a polarity opposite that given in the PWM_CHANCFG.POLDH bits in the PWM_CHANCFG register. The PWM sync pulse will continue to operate if it is already enabled. A PWMTRIP interrupt occurs if unmasked, to notify the software of this event.

Note that even if the clock to the PWM is damaged, an external trip event will turn off the PWM outputs, but the PWMTRIP interrupt may not occur.

ADSP-BF60x PWM Register List

The pulse-width modulator (PWM) includes multiple timers (providing period flexibility) and channels (provide mode, interrupt, and pulse shape flexibility), permitting a wide variety of PWM output options for motor control and other applications. A set of registers govern PWM operations. For more information on PWM functionality, see the PWM register descriptions.

Table 18-1: ADSP-BF60x PWM Register List

Name	Description
PWM_CTL	Control Register
PWM_CHANCFG	Channel Config Register
PWM_TRIPCFG	Trip Config Register

Table 18-1: ADSP-BF60x PWM Register List (Continued)

Name	Description
PWM_STAT	Status Register
PWM_IMSK	Interrupt Mask Register
PWM_ILAT	Interrupt Latch Register
PWM_CHOPCFG	Chop Configuration Register
PWM_DT	Dead Time Register
PWM_SYNC_WID	Sync Pulse Width Register
PWM_TM0	Timer 0 Period Register
PWM_TM1	Timer 1 Period Register
PWM_TM2	Timer 2 Period Register
PWM_TM3	Timer 3 Period Register
PWM_TM4	Timer 4 Period Register
PWM_DLYA	Channel A Delay Register
PWM_DLYB	Channel B Delay Register
PWM_DLYC	Channel C Delay Register
PWM_DLYD	Channel D Delay Register
PWM_ACTL	Channel A Control Register
PWM_AH0	Channel A-High Duty-0 Register
PWM_AH1	Channel A-High Duty-1 Register
PWM_AL0	Channel A-Low Duty-0 Register
PWM_AL1	Channel A-Low Duty-1 Register
PWM_BCTL	Channel B Control Register
PWM_BH0	Channel B-High Duty-0 Register

Table 18-1: ADSP-BF60x PWM Register List (Continued)

Name	Description
PWM_BH1	Channel B-High Duty-1 Register
PWM_BL0	Channel B-Low Duty-0 Register
PWM_BL1	Channel B-Low Duty-1 Register
PWM_CCTL	Channel C Control Register
PWM_CH0	Channel C-High Pulse Duty Register 0
PWM_CH1	Channel C-High Pulse Duty Register 1
PWM_CL0	Channel C-Low Pulse Duty Register 0
PWM_CL1	Channel C-Low Duty-1 Register
PWM_DCTL	Channel D Control Register
PWM_DH0	Channel D-High Duty-0 Register
PWM_DH1	Channel D-High Pulse Duty Register 1
PWM_DL0	Channel D-Low Pulse Duty Register 0
PWM_DL1	Channel D-Low Pulse Duty Register 1

ADSP-BF60x PWM Interrupt List

Table 18-2: ADSP-BF60x PWM Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
PWM0 PWMTMR Group	28		LEVEL
PWM0 Trip	29		LEVEL
PWM1 PWMTMR Group	30		LEVEL
PWM1 Trip	31		LEVEL

ADSP-BF60x PWM Trigger List

Table 18-3: ADSP-BF60x PWM Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
PWM0 PWMTMR Group	17	LEVEL
PWM1 PWMTMR Group	18	LEVEL

Table 18-4: ADSP-BF60x PWM Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
None		

Architectural Concepts

The PWM Controller is driven by a clock, whose period is t_{SCLK}. The PWM generator produces four pairs (four high-side and four low-side) of PWM signals on the eight PWM output pins. Each high and low pair constitutes a channel. For example PWM_AL/PWM_AH make up channel A, and PWM_BLPWM_BH make up channel B and so on. Each pair of channel outputs can be produced with reference to either a main timer or to an independent timer. These timers operate on a switching frequency determined by the PWM_TM0 registers. There are 2 duty registers for every PWM output, which enable generation of symmetrical or asymmetrical waveforms that produce lower harmonic distortion in three-phase PWM inverters, with minimal CPU intervention.

Block Diagram

The following figure shows a block diagram that represents the main functional blocks of the PWM controller.

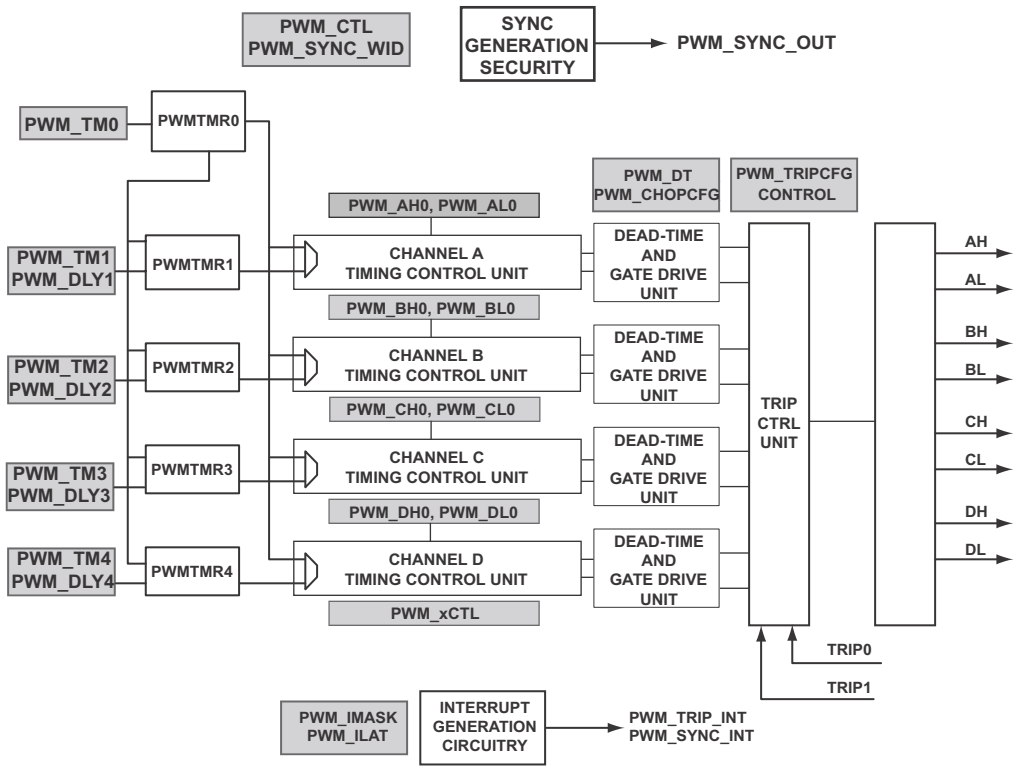


Figure 18-1: PWM Block Diagram

The primary blocks are described below.

- Each pair of PWM signals is referenced either to the main timer or to the independent timer.
- PWMTMR0 is the main timer and can trigger the delayed start of the other timers.
- Timing Control Units, one for each channel, which together form the core of the PWM, generate the required complex waveforms on the high side and low side outputs for the respective channel.
- Dead Time insertion is done after the ideal PWM output pair is generated.
- The Gate Drive Unit generates the high-frequency chopping signal and subsequently mixes it with the requisite PWM output signals.
- The PWM Shutdown and Interrupt Controller manages the various PWM shutdown modes for the timing unit and generates the requisite interrupt signals.
- The PWM Sync Pulse Control Unit generates the internal PWM_SYNC pulse and also controls whether the external PWM_SYNC input pulse is used.

Timer Units

Five timers make up the time base for the PWM module. The main timer, PWMTMR0 operates at a switching frequency determined by the period register `PWM_TM0`. The four remaining timers PWMTMR1,

PWMTMR2, PWMTMR3 and PWMTMR4 can operate at independent switching frequencies determined by their respective registers.

These four timers can be programmed to work at a multiple of the main timer's frequency by programming respective PWM_TMx appropriately. In this case, the PWM_DLYA registers can be used to provide for lead-lag phase control of a given timer with respect to the main timer PWMTMR0.

NOTE: When delayed operation of a timer is enabled, its register value must either be equal to the PWM_TM0 register value or PWM_TM0 must be an integer multiple of each timers register to ensure proper function. Non-integer multiples are not allowed.

PWM Switching Frequency (PWM_TM) Register

The 16-bit read/write PWM period register controls the PWM switching frequency. The fundamental timing unit of the PWM Controller is t_{SCLK}. Therefore, for a 100 MHz system clock (SCLK), f_{SCLK}, the fundamental time increment (t_{SCLK}) is 10 ns. The value written to a timer's register is effectively the number of t_{SCLK} clock increments in half a PWM period. The required timer register value as a function of the desired PWM switching frequency (f_{PWM}) is given by:

$$PWM\_TM = f_{SCLK}/2 \times f_{PWM}$$

Therefore, the PWM switching period (T_s) can be written as:

$$T_s = 2 \times PWM\_TM \times t_{SCLK}$$

For example, for an f_{SCLK} of 100 MHz and a desired PWM switching frequency (f_{PWM}) of 10 kHz (T_s = 100 ms), the correct value to load into the timer register is:

$$PWM\_TM = 100 \times 10^6 \div 2 \times 10 \times 10^3 = 5000$$

The largest value that can be written to the 16-bit timer register is 0xFFFF = 65,535, which, at an f_{SCLK} of 100 MHz, corresponds to a minimum PWM switching frequency of:

$$f_{PWM(min)} = 100 \times 10^6 \div 2 \text{ x } 65535 = 762 \text{ Hz}$$

NOTE: Timer register values of 0 and 1 are not defined and should not be used when the PWM outputs or PWM is enabled.

Timer Unit Operation

The PWM timer is an up-down counter operates on the peripheral clock of period t_{CK}. The period of the PWM timer is divided into two halves. In the first half, the timer roughly counts down from PWM_TMx/2 to –PWM_TMx/2. During this half, the PWM_STAT.TMROPHASE register is held at 0 In the second half of the period, the timer roughly counts up from –PWM_TMx/2 to PWM_TMx/2. The PWM_STAT.TMROPHASE bit indicates a 1 during this half.

The actual partition of the periods varies slightly between odd and even values of the half-period, PWM_TMx.

When the timer register value is odd, for example 11, then that timer loads +5 at the beginning of the period, counts down from +5 to –5 in the first half, reloads –5 at the midpoint and counts up from –5 to +5 in the second half. The reload values at the period and mid-period boundaries are the same as the previous count. It totally counts 2 × 11 = 22 counts in the entire period. This is shown in the figure below. Note that both half-periods have a count of 11 each.

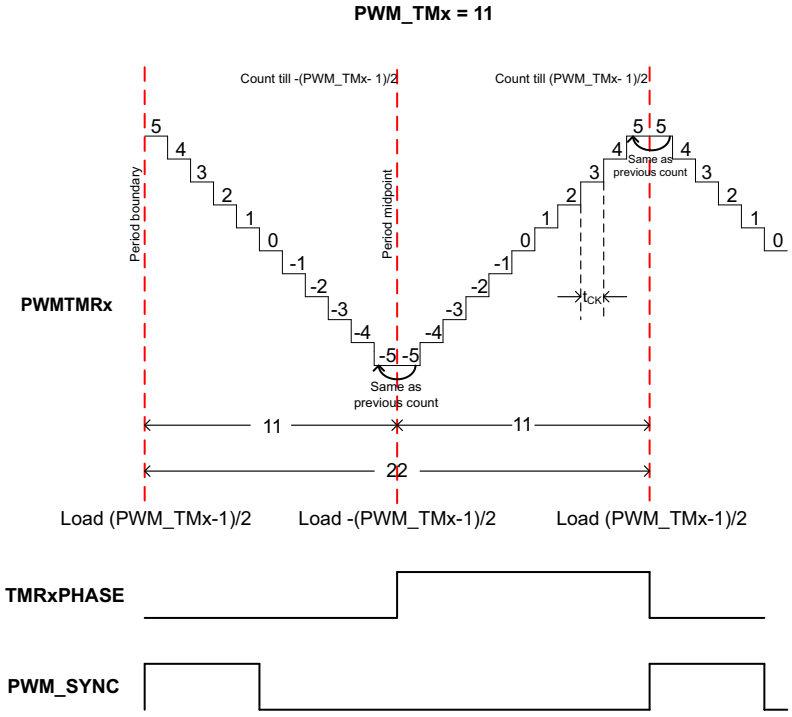


Figure 18-2: Operation of Timer for Odd Value of PWM_TM

When the timer register value is even, for example 12, then that timer loads +5 at the beginning of the period, counts from +5 to –6 in the first half, reloads –5 at the midpoint and counts up from –5 to +6 in the second half. The reload values at the period and mid-period boundaries are different from the previous count. It totally counts 2 × 12 = 24 counts in the entire period. This is shown in the figure below. Note that both half-periods have a count of 12 each.

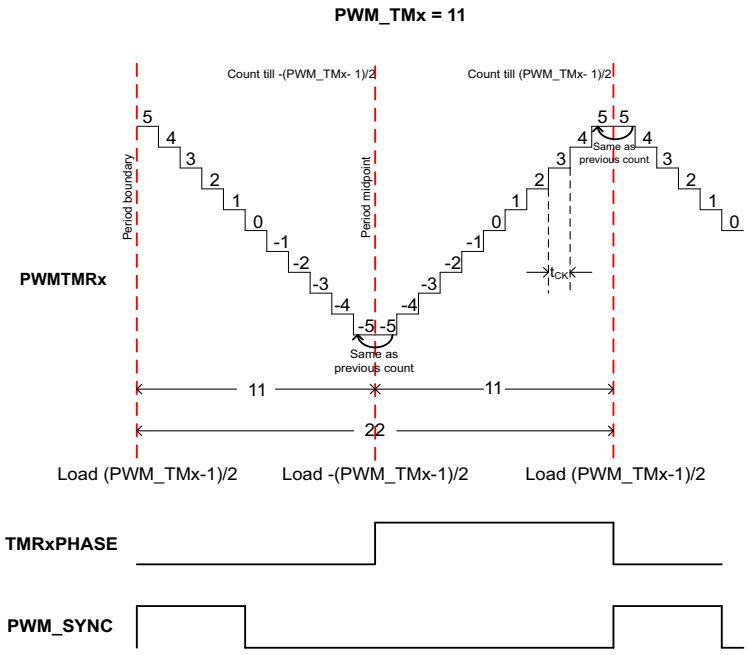


Figure 18-3: Operation of Timer for Even Value of PWM_TM

Note that in the operation discussed in this section, double buffering of all channel registers and the timer registers take place at the period boundary of the respective timers.

Phase Offset Control

The PWM timers (PWMTMR1 through PWMTMR4) can operate with a programmable phase lag with respect to the main timer, PWMTMR0. Using DLY x delay counter in conjunction with PWMTMR0 and setting the `PWM_CTL.DLYAEN` bit implements this feature for a given channel.

If this feature is enabled for channel A (and channel A is using PWMTMR1 for generating duty cycle), when PWMTMR0 reaches its period boundary, it triggers, DLY A, which counts out SCLK cycles equal in number to the value programmed in the `PWM_DLYA` register. At the end of this count, DLY x sends out a trigger to PWMTMR1. Thus PWMTMR1 receives a synchronization pulse in every period of PWMTMR0 at a point delayed from its period boundary by the value in `PWM_DLYA` register. For more information on how channels can reference different timers for their outputs, see "Channel Timing Control Unit"

NOTE: The following conditions must be satisfied when this feature is used on PWMTMRy for channel Y with respect to PWMTMRx.

- The `PWM_DLYA` register must be programmed to a value less than 2 × TMy
- PWM_TM0 = N × PWM_TMy where N is an integer.

The function of PWMTMRy (PWMTMR1 in the example above) differs in cases where PWM_TM0 = PWM_TM1 from cases where PWM_TM0 = N × PWM_TM1. Both cases are examined below.

Case 1: PWM_TM0 = PWM_TMy

In this case every period of PWMTMRy starts its period again, on reception of the synchronization pulse from DLY-x. If the trigger from DLY-x is late, it holds its count till the trigger comes. If the trigger is a bit early, it reloads without regard to whether it has completed its current period. Thus PWMTMRy is re-synced with PWMTMR0 with a phase lag as programmed in PWM_DLYA register, in every one of its periods.

Note that in this case the expiration of the DLY x counter is the period boundary of PWMTMRy. Therefore, this is the point of update of all the double buffered registers related to the given channel (except the delay registers which are double buffered at the period boundary of PWMTMR0).

Phase Offset Control Using DELAY shows an example where:

- PWM_TM0, PWM_TM1 and PWM_TM2 are programmed with the same value.
- PWM_DLYA and PWM_DLYB are programmed values DELAY1 and DELAY2 respectively, such that DELAY2 > DELAY1.
- Channel A's outputs are referenced to TMR1 and Channel B's outputs to PWMTMR2.

The delay registers are double buffered and the new value of delay is reloaded at the period boundary of PWMTMR0. The two options described below exist if the new value is different from the older one. The behavior of PWMTMRy in both these cases is discussed, in conjunction with *Impact of New DELAY Value on Timer Count for Equal TM*. It is assumed that channel B references its outputs to PWMTMR0 and channel A references its outputs to PWMTMR1.

1. The new delay value is higher than the previous value. Therefore, the corresponding PWMTMRy is allowed more than one period's time between consecutive triggers from the DLY-x counter. In this case, after reaching its period boundary, PWMTMRy holds its count at the period boundary and waits for the trigger from DLY-x. This is shown in CASE A in *Impact of New DELAY Value on Timer Count for Equal TM*.
2. The next delay value programmed is smaller than the previous value. In this case, the corresponding PWMTMRy is allowed only less than one period's time between consecutive triggers from the DLY-x counter. Though the trigger comes earlier in this case, before PWMTMRy has counted out one full period, it reloads and starts its period again. This is shown in CASE B in *Impact of New DELAY Value on Timer Count for Equal TM*.

Therefore, PWMTMR1 waits and obeys a synchronization pulse from DLY A in every one of its periods.

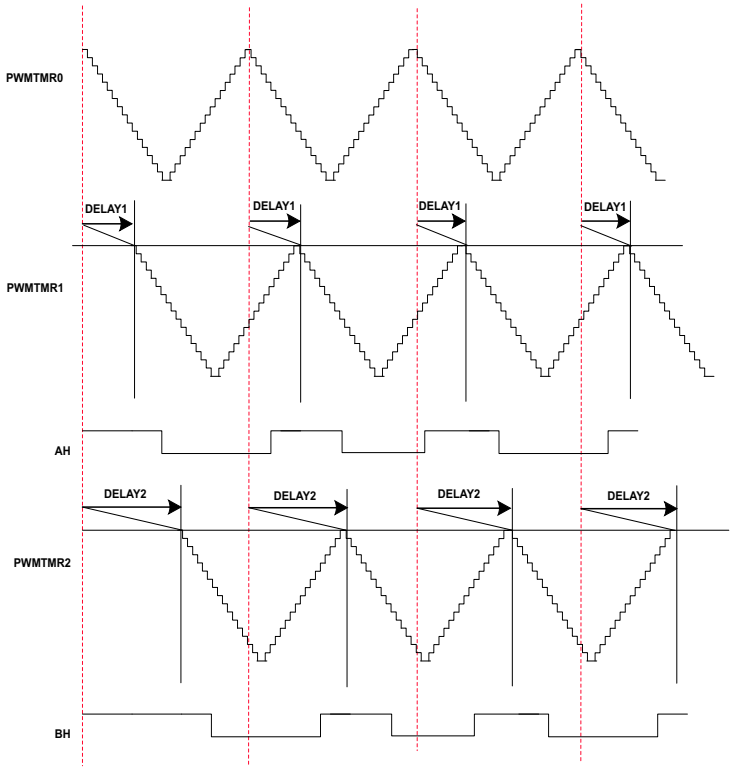


Figure 18-4: Phase Offset Control Using DELAY

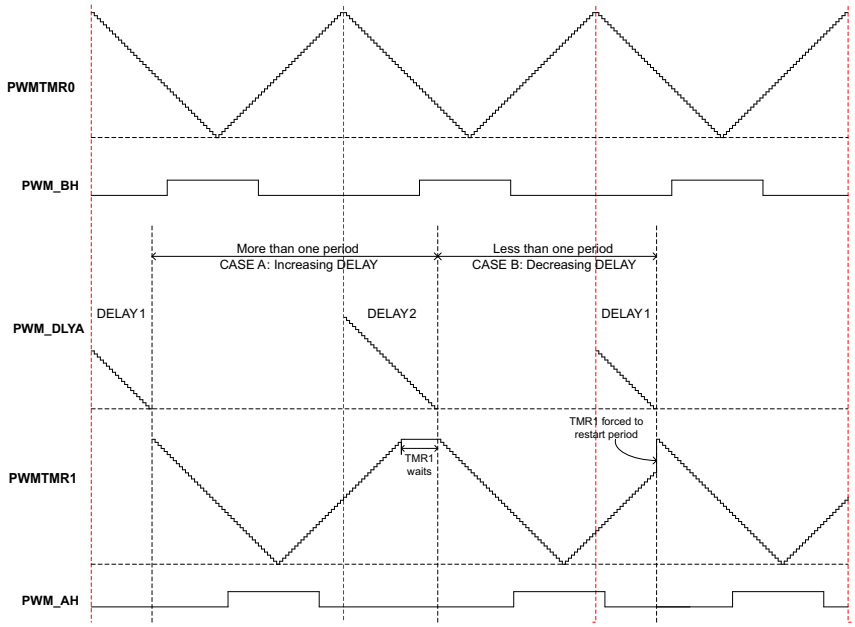


Figure 18-5: Impact of New DELAY Value on Timer Count for Equal TM

Case 2: PWM_TM0 = N x PWM_TMy

In this case, within a single period of PWMTMR0 a program can fit multiple periods (N) of PWMTMRy. Additionally, the DLY-x counter is triggered only once every N periods of PWMTMRy.

The operation is as follows: Every Nth period of PWMTMRy, it expects a synchronization pulse from DLY-x counter. When it counts out that period if the trigger hasn't yet arrived yet, it waits at the end of the period for the trigger, and starts counting down once it arrives. If the trigger comes earlier than that, it restarts immediately without waiting to complete its period count.

In the intervening periods, PWMTMRy operates independently. As the period ends, it reloads and starts the next period without intervention from the delay counter.

Impact of DELAY Value Change for the Multiple TM gives an example with N = 2. Note that PWMTMRy syncs up with PWMTMR0 every 2nd period, and is free running across every odd period boundary.

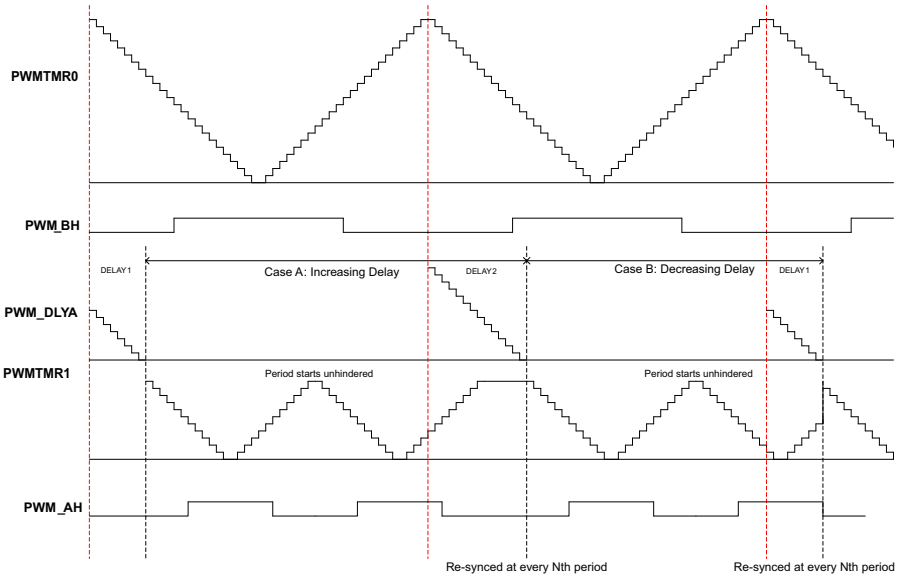


Figure 18-6: Impact of DELAY Value Change for the Multiple TM

Channel Timing Control Unit

The channel timing control unit is the core of the PWM. There are four separate channels, each channel controlling a pair of output signals – the high side output and the low side output.

Channel Control

The PWM_CHANCFG register controls the static configuration of all the channels and is to be initialized once before the beginning of the operation.

NOTE: The PWM_CHANCFG register is not double buffered and the contents of it must not be changed once the PWM is enabled.

Each channel works off a reference timer base. The time base can be either the main timer PWMTMR0 or the appropriate PWMTMRx as given below. This can be configured with the PWM_CHANCFG.REFTMRA bit field.

- Channel A can work off PWMTMR0 or PWMTMR1
- Channel B can work off PWMTMR0 or PWMTMR2
- Channel C can work off PWMTMR0 or PWMTMR3
- Channel D can work off PWMTMR0 or PWMTMR4

The PWM_xCTL registers contain bits which control the dynamic pulse behavior of the channel outputs and this register is double buffered. This register has bits which control enable/disable and pulse positioning of the outputs. The impact of these is explained in the following sections.

Pulse Positioning and Duty Cycle Registers

The PWM_ACTL.PULSEMODEHI[1:0] and PWM_ACTL.PULSEMODELO[1:0] fields define the region within the timer period where the output pulses should be positioned. When the PWM_CHANCFG.MODELSC bit is 0, the PWM_ACTL.PULSEMODEHIfield specifies the pulse positioning for both the high-side and low-side outputs of the channel. When the bit is 1, PWM_ACTL.PULSEMODELO defines the pulse positioning for the low side channel output, while PWM_ACTL.PULSEMODEHI continues to do serve this purpose for the high side channel output.

Two Duty-Cycle registers are provided for each channel output: PWM_AH0 and PWM_AH1 for the high side output, and PWM_AL0 and PWM_AL1 for the low side output. These registers determine the width of the output pulses. When the PWM_CHANCFG.MODELSC bit is 0, the high side Duty-Cycle registers are used for the low side output pulse width determination as well. The duty cycle range that can be programmed into these registers is between –PWMTMx/2 and +PWMTMx/2 when dead-time is not considered.

When dead-time is considered, for PULSEMODE 00 and 01, the programmed duty is modified for use by the PWM in such a way that the range is limited between the values [–PWMTMx/2 + PWM_DT] to [+PWMTMx/2 + PWMDT] considering the high-side output. Dead-time is explained in detail in a later section. For PULSEMODEs 10 and 11, the high-side Duty-Cycle registers are limited between [PWM_ TMx/2 + PWM_DT] and [–PWM_TMx/2 – PWM_DT].

NOTE: Values programmed into these registers that fall outside these limits result in over/under modulation.

Duty Cycle and Pulse Positioning Control

The PWM_ACTL.PULSEMODEHI and PWM_ACTL.PULSEMODELO fields control how the Duty cycle registers modify the waveform of the high and low side outputs. (ThePWM_ACTL.PULSEMODEHI and PWM_ACTL. PULSEMODELO fields are referred to as *PULSEMODE* in the subsequent discussion.)

- PULSEMODE = 00 – Allows symmetrical pulse waveform around the center of the PWM period. In this mode, only one of the Duty Cycle registers is used for an output—for example, for the AH output, only the PWM_AH0 register is used. Note that in this mode, the values in the duty cycle registers are scaled such that a value of 0 produces 50% duty.
- PULSEMODE = 01 – Allows asymmetrical pulse waveform around the center of the PWM period. In this mode both the Duty Cycle registers are used. For example for the AH output, PWM_AH0 and PWM_ AH1 registers are used. In this mode, if PWM_AH1 is programmed with the same value as PWM_AH0, the obtained output is identical to that in the PULSEMODE = 00 case.
- PULSEMODE = 10 or 11 – Allow pulse waveforms to be produced either on the first half or the second half of the PWM period respectively, and both PWM_AH0 and PWM_AH1 registers are used.

NOTE: In PULSEMODE = 10 the condition PWM_AH0> PWM_AH1. If the low side is working off the low side Duty-Cycle registers, the condition PWM_AL0> PWM_AL1 should be strictly adhered to.

NOTE: In PULSEMODE = 11 the condition PWM_AH0< PWM_AH1. If the low side is working off the low side Duty-Cycle registers, the condition PWM_AL0< PWM_AL1 should be strictly adhered to.

The following figure shows the pulse positioning modes as described above for PWM_AH. In the figure, DUTY0 is assumed to be the value in PWM_AL0 register and DUTY1, the value in PWM_AH1 register. The step signal, count, indicates the output of the timer that Channel A is working off. Also, in the example, the signal is assumed to be configured as active high, and dead-time is assumed to be zero.

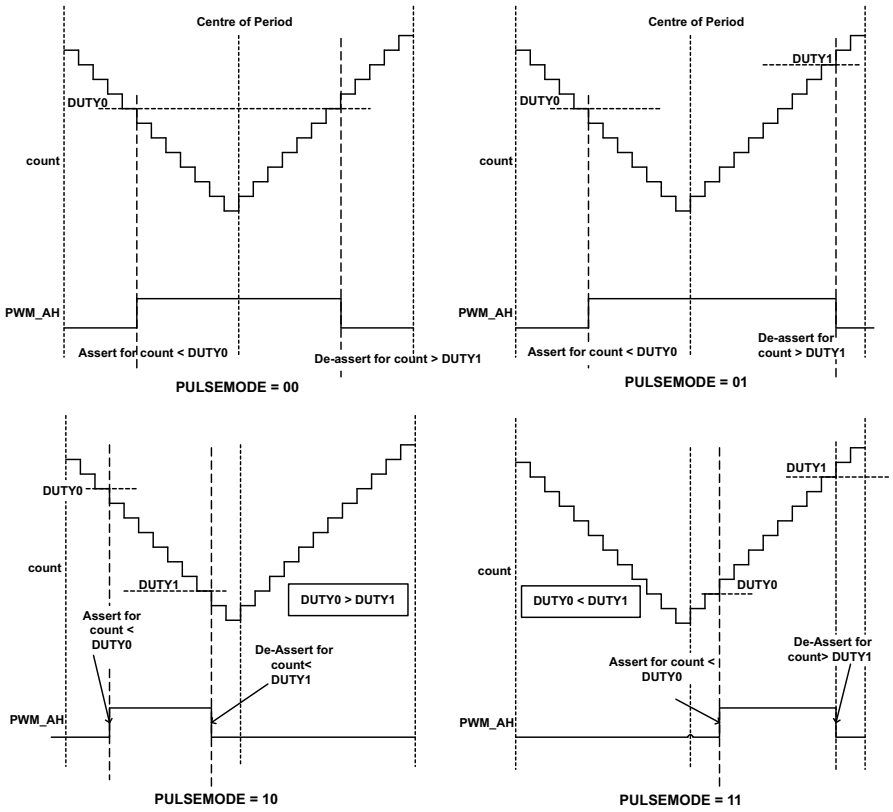


Figure 18-7: Pulse Positioning Modes

Channel Low Side Output Dependent Operation Mode and Dead-Time

The low-side output waveform can be programmed to be dependent on the high side output waveform or be totally independent. This is controlled using the PWM_CHANCFG.MODELSC bit.

For example, the channel A produces the high side output PWM_AH and the low side output PWM_AL. When PWM_CHANCFG.MODELSC = 0, the low-side output is also generated using the high-side duty-cycle registers for pulse width, PWM_ACTL.PULSEMODEHI bits for pulse positioning and the PWM_CHANCFG.POLAH bit for polarity. If PWM_DT is programmed to 0, the low-side output is an inverted version of the high-side output.

When PWM_DT is programmed with a non-zero value, both the high-side and low-side outputs are shrunk symmetrically about the points of transition in the zero dead-time case by an amount PWM_DT.

The high and low-side outputs for the case with zero and non-zero dead-time for PWM_ACTL.PULSEMODEHI = 00 and 01 are shown in the following figure. In the figure, DUTY0 is the value programmed into the PWM_

AH0 register and DUTY1 is the value programmed into the PWM_AH1 register. PWM_CHANCFG.POLAH is 1 indicating that both signals are active high. PWM_DT holds the value DT.

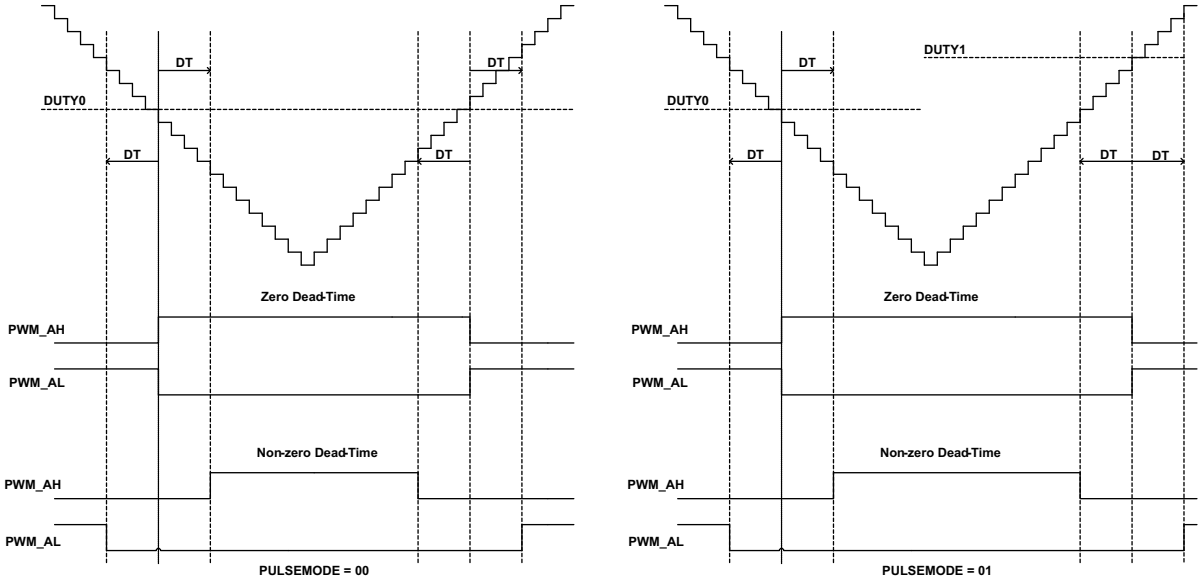


Figure 18-8: Channel Outputs in Dependent Mode for PULSEMODE = 00, 01

The high and low-side outputs for the case with zero and non-zero dead-time for PWM_ACTL.PULSEMODEHI = 10 and 11 are shown in the following figure. In the figure, DUTY0 is the value programmed into PWM_AH0 register and DUTY1 is the value programmed into the PWM_AH1 register. PWM_CHANCFG.POLAH is 1 indicating that both signals are active high. PWM_DT holds the value DT.

NOTE: Bringing dead-time into the picture, the guidelines for programming the Duty-Cycle registers in PULSEMODEs 10 and 11 given in "PWM Duty Cycle and Pulse Positioning Control" are modified as follows:

PULSEMODE 10: PWM_xH0 – DT > PWM_xH1 + DT

PULSEMODE 11: PWM_xH0 + DT < PWM_xH1 – DT

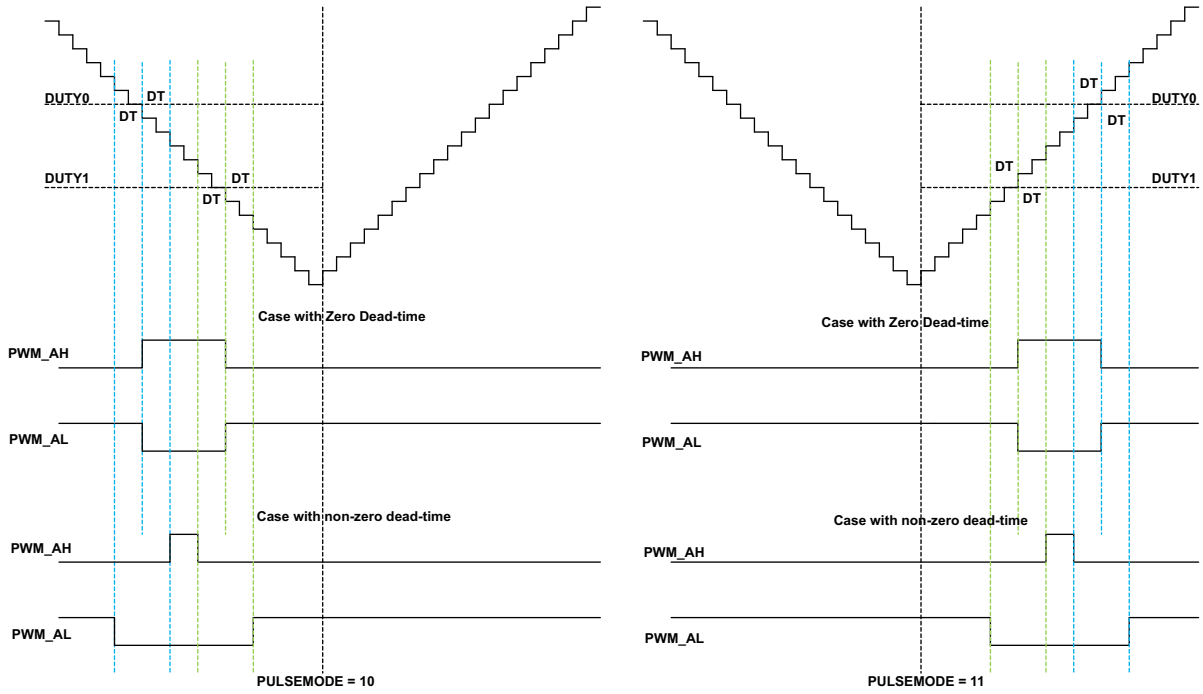


Figure 18-9: Channel Outputs in Dependent Mode for PULSEMODE = 10, 11

Channel High Side and Low Side Outputs, Independent Operation Mode

Independent control of the channel outputs, PWM_AH0 and PWM_AL0 is possible by setting the PWM_CHANCFG.MODELSA bit to 1. In this case, PWM_AH is generated using PWM_AH0, PWM_AH1 for pulse width, PWM_ACTL.PULSEMODEHI for pulse position, and PWM_CHANCFG.POLAH for polarity. PWM_AL is generated using PWM_AL0, PWM_AL1 for pulse width, PWM_ACTL.PULSEMODELO for pulse position and PWM_CHANCFG.POLAL for polarity.

NOTE: In the independent mode, the dead-time insertion is not applicable and the dead-time is forced to zero by the hardware.

The following figure shows an example of the independent mode of operation where PWM_AH and PWM_AL work off different register bits.

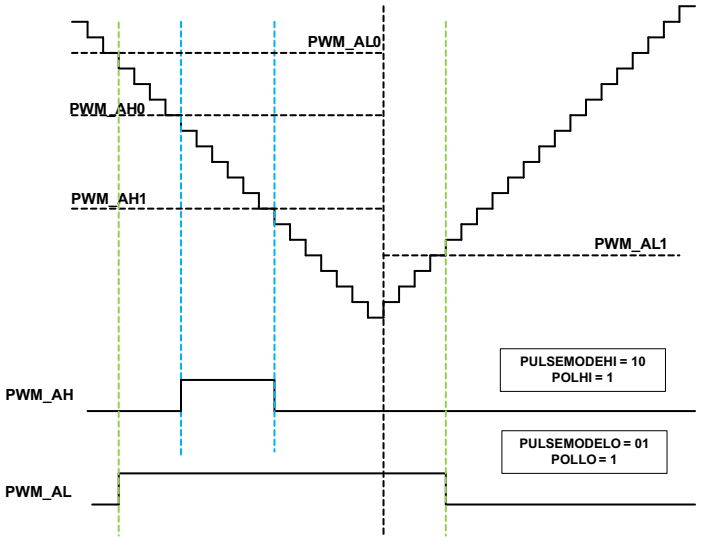


Figure 18-10: PWM_AH and PWM_AL in Independent Operation Mode

Note that PWM_AH and PWM_AL can be positioned in the timer period with a given phase difference between them. This is achieved by having PULSEMODEHI and PULSEMODELO bits programmed to different values. This is shown in the next figure.

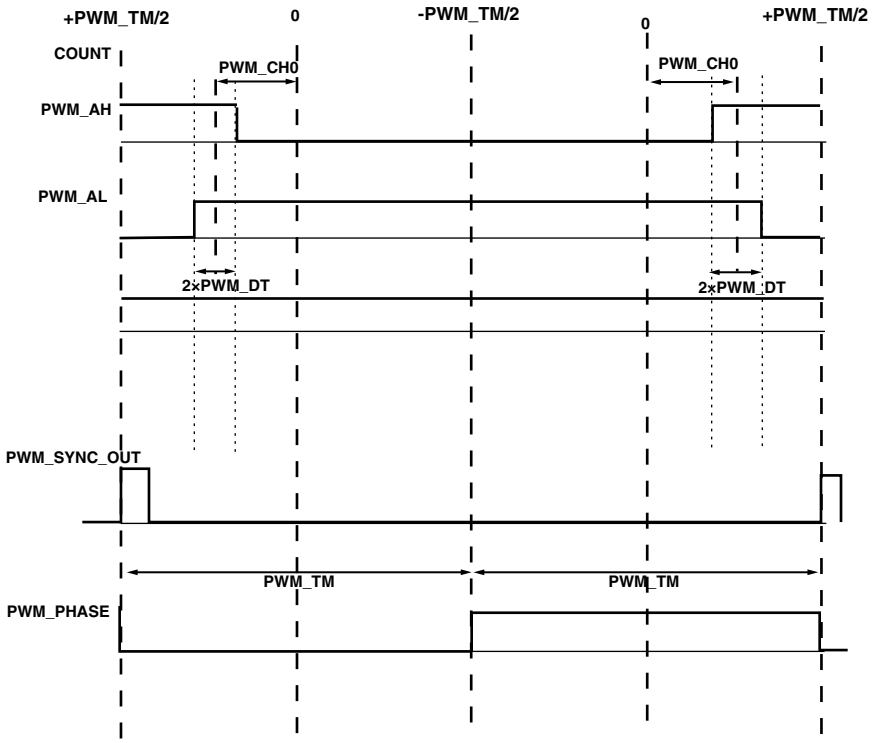


Figure 18-11: Channel Outputs Controlled Independently

Switched Reluctance Motors Application

In the typical power converter configuration for switched or variable reluctance motors, the motor winding is connected between the two power switches of a given inverter leg. Therefore, to allow for a complete circuit in the motor winding, it is necessary to turn on both switches at the same time.

Switched reluctance motors are used in the following configurations: Hard Chop, Alternate Chop, Soft Chop—Bottom On, and Soft Chop—Top On.

The following figure shows the four SR mode types as active high PWM output signals.

Hard Chop mode contains independently programmed rising edges of a channel's high and low signals in the same PWM half cycle and both contain independently programmed falling edges in the next PWM half cycle. The PWM_CHANCFG.POLAH and PWM_CHANCFG.POLAL are programmed to same values.

Alternate Chop mode is similar to normal PWM operation but the PWM channel high and low signal edges are always opposite and are independently programmed. The PWM_CHANCFG.POLAH and PWM_CHANCFG.POLAL are programmed to opposite values. The Low Side Invert is the only difference between Hard Chop mode and Alternate Chop mode.

Soft Chop—Bottom On uses a 100% duty on the low side of the channel and Soft Chop—Top On uses a 100% duty on the high side of the channel. Similar to Hard Chop mode the PWM_AH0 duty register is used for the high channel and PWM_AL0duty register is used for the low channel.

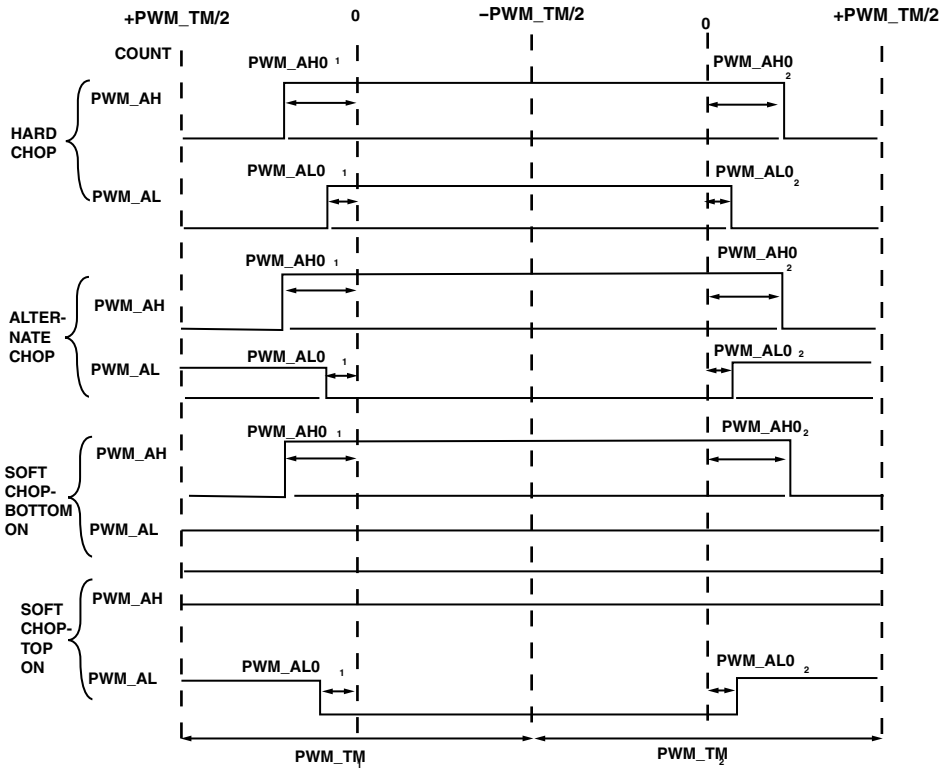


Figure 18-12: Four SR Mode Types, Active High PWM Output Signals

Switching Dead Time (PWM_DT) Register

The second important parameter that must be set up in the initial configuration of the PWM Controller is the switching dead time. This is a short delay introduced between turning off one PWM signal (for example, AH) and turning on the complementary signal (for example, AL). This short time delay permits the power switch being turned off (AH in this case) to completely recover its blocking capability before the complementary switch is turned on. This time delay prevents a potentially destructive short-circuit condition from developing across the dc link capacitor of a typical voltage source inverter.

The 10-bit, read/write PWM_DT register controls the dead time. This register controls the dead time inserted into the three pairs of PWM output signals. Dead time (T_d) is related to the value in the PWM_DT register by:

$$T_d = PWM\_DT \times 2 \times t_{SCLK}$$

Therefore, a PWM_DT value of 0x00A introduces a 200 ns delay (for a SCLK of 100 MHz) between turning off any PWM signal (for example, AH) and then turning on its complementary signal (for example, AL). The length of the dead time can therefore be programmed in increments of $2 \times t_{SCLK}$ (or 20 ns for an SCLK of 100 MHz). The PWM_DT register is a 10-bit register whose maximum value of 0x3FF (1023 decimal) corresponds to a maximum programmed dead time of:

$$T_{d(max)} = 1023 \times 2 \times t_{SCLK} = 1023 \times 2 \times 10 \times 10^{-9} = 20.5\ \mu s$$

for an f_{SCLK} rate of 100 MHz. The dead time can be programmed to be zero by writing 0 to the PWM_DT register.

Duty Cycle with Dead-Time Control: Calculations for PULSEMODE 00

The duty cycle registers are scaled such that a value of 0 represents a 50% PWM duty, cycle. The switching signals produced are also adjusted to incorporate the programmed dead time value in the PWM_DT register. The unit in this case produces active low signals so that a low level corresponds to a command to turn on the associated power device.

A typical pair of PWM outputs, PWM_AH and PWM_AL, is shown in the following figure. The time values in the figure indicate the integer value in the associated register and can be converted to time by multiplying by the fundamental time increment, t_{CK}. In the example channel A is working off of PWMTMR0.

Because PULSEMODE is set to 00, the switching patterns are perfectly symmetrical about the mid-point of the switching period. The dead time is incorporated by moving the switching instants of both PWM signals away from the instant set by the PWM_AH0 register. Both switching edges are moved by an equal amount ($PWMDT \times t_{CK}$) to preserve the symmetrical output patterns. Also shown is the PWM_SYNC_OUT pulse whose rising edge denotes the beginning of the switching period, and the PWM_STAT.TMR0PHASE bit.

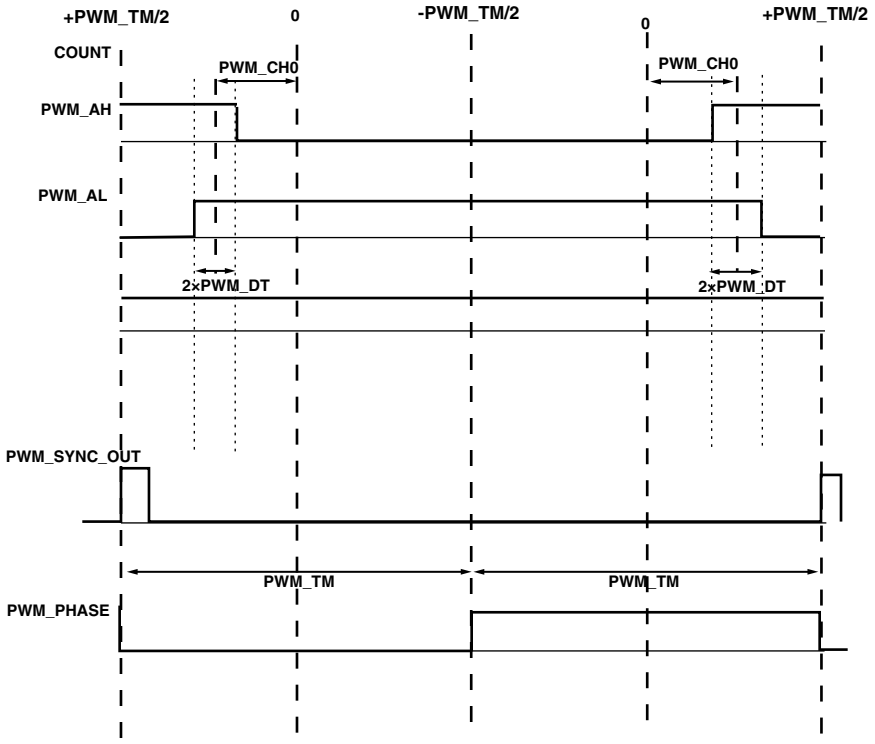


Figure 18-13: Dead-Time Between Outputs in Dependent Mode

The resulting on-times (active low) of the PWM signals over the full PWM period (two half-periods) produced by the PWM timing unit and illustrated in the figure above, may be written as shown in the following equation.

T_{AH} = (PWM_TM0 + 2 × (PWM_AH0 - PWM_DT)) × $t_{CK;}$

Range of T_{AH} is [0:2 × PWM_TM0 × t_{CK}]

T_{AL} = (PWM_TM0 - 2 × (PWM_AH0 + PWM_DT)) × $t_{CK;}$

Range of T_{AL} is [0:2 × PWM_TM0 × t_{CK}]

The corresponding duty cycles are shown in the following equations.

$$d_{AH} = \frac{T_{AH}}{Ts} = \frac{1}{2} + \frac{PWM\_AH0 - PWM\_DT}{PWM\_TM}$$

$$d_{AL} = \frac{T_{AL}}{Ts} = \frac{1}{2} - \frac{PWM\_AH0 + PWM\_DT}{PWM\_TM0}$$

The negative values of T_{AH} and T_{AL} are not permitted and the minimum permissible value is zero, corresponding to a 0% duty cycle. In a similar fashion, the maximum value is T_s, the PWM switching period, corresponding to a 100% duty cycle. Calculation of duty for other PULSEMODEs can be similarly carried out.

Special Consideration for PWM Operation in Over-Modulation

The PWM Timing Unit is capable of producing PWM signals with variable duty cycle values at the PWM output pins. In PULSEMODEs 00 and 01, at the extremities of the modulation process, we have 0% and 100%. IN PULSEMODEs 01 and 10, at the extremities the modulation process, we have 0% and 50%. The modulation is called FULL OFF when the lower extremity of modulation is set for any PWM timer period for the corresponding channel. The modulation is called FULL ON when the higher extremity of modulation is set for any PWM timer period for the corresponding channel. In between, for other duty cycle values, the operation is termed NORMAL MODULATION.

- Full On Modulation. In PULSEMODEs 00 and 01, a PWM channel is said to be in FULL ON modulation if the high-side output of that channel is asserted for the whole duration of the period of the PWM timer that channel is referencing. Condition for FULL ON MODULATION: PWM_xH0 - DT > PWM_TMy/2 for PULSEMODE 00 and the additional condition that PWM_xH1 - DT > PWM_TMy/2 as well.

  In PULSEMODE 10, a PWM channel is said to be in FULL ON modulation if the high-side output of that channel is asserted for the whole duration of the first half period of the PWM timer that the channel is referencing. Condition for FULL ON MODULATION: PWM_xH0 - DT > PWM_TMy/2 and PWM_xH1 + DT < -PWM_TMy/2.

  In PULSEMODE 11, a PWM channel is said to be in FULL ON modulation if the high-side output of that channel is asserted for the whole duration of the second half period of the PWM timer that the channel is referencing. Condition for FULL_ON MODULATION: PWM_xH0 + DT < -PWM_TMy/2 and PWM_xH1 - DT > PWM_TMy/2.

- Full Off Modulation. In PULSEMODEs 00 and 01, a PWM channel is said to be in FULL OFF modulation if the high-side output of that channel is de-asserted for the whole duration of the period of the PWM timer that channel is referencing. Condition for FULL OFF MODULATION: PWM_xH0 - DT < -PWM_TMy/2 for PULSEMODE 00 and the additional condition that PWM_xH1 - DT < -PWM_TMy/2 as well.

  In PULSEMODE 10, a PWM channel is said to be in FULL OFF modulation if the high-side output of that channel is de-asserted for the whole duration of the first half period of the PWM timer that the channel is referencing (In the second half-period it is anyway de-asserted). Condition for FULL OFF MODULATION: PWM_xH0 - DT < -PWM_TMy/2 and PWM_xH1 + DT < PWM_xH0 - DT.

  In PULSEMODE 11, a PWM channel is said to be in FULL OFF modulation if the high-side output of that channel is de-asserted for the whole duration of the second half period of the PWM timer that the channel is referencing (In the first half of the period it is anyway de-asserted). Condition for FULL OFF MODULATION: PWM_xH0 + DT > PWM_TMy/2 and PWM_xH1 - DT > PWM_xH0 + DT.

- Normal Modulation. All other cases of modulation fall under this category.

There are certain situations, on transition either into or out of either FULL ON or FULL OFF modulation, where it is necessary to insert additional emergency dead time delays to prevent potential shoot through conditions in the inverter. Disable/Enable usage (related to DISHI and DISLO bits in `PWM_CTL` register) also can potentially cause outputs to violate shoot through condition criteria. Another case is when the

phase delay of a PWM timer is varied by large values. These transitions are detected automatically and if appropriate for safety, an emergency dead-time is inserted to prevent shoot through conditions.

The insertion of the additional emergency dead time into one of the PWM signals of a given pair during these transitions is only needed if otherwise both PWM signals would be required not to toggle within a dead time of each other. The additional emergency dead time delay is inserted into the PWM signal that is toggling into the ON state. In effect the turn ON of this signal is delayed by an amount (2 x PWMDT x t_{CK}) from the rising edge of the opposite output. After this delay, the PWM signal is allowed to turn ON provided the desired output is still scheduled to be in the ON state after the emergency dead time delay.

The following figure illustrates two example of such a transition. Here POLHI is kept at 1. PWM_AH has been in FULL ON modulation for sometime, and during the current period, its PULSEMODE is changed to 10 keeping the FULL ON condition. At the half-period boundary it is forced to transition to a de-asserted state due to this being PULSEMODE 10. It can be seen that an emergency dead-time is inserted on the low-side output.

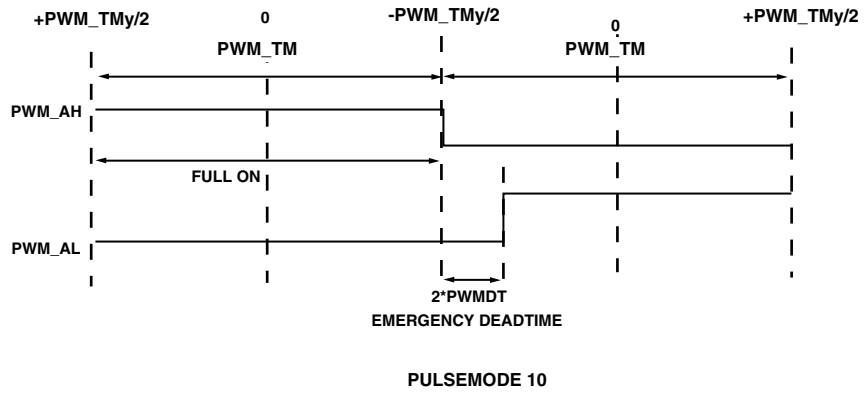


Figure 18-14: Over Modulation Transition Scenarios

Output Disable and Cross-Over Functions

Each PWM_ACTL channel control register contains separate enable bits for the high and low side signals. For example, the PWM_ACTL.DISHI and PWM_ACTL.DISLO bits in the channel A control register control the enable/disable of the AH and AL outputs respectively. If the disable bit is set (= 1), then the corresponding PWM output is disabled, irrespective of the value of the corresponding duty cycle register. This PWM output signal remains in the OFF state as long as the corresponding enable/disable bit is set.

The PWM_ACTL.XOVR bit, or the cross-over bit allows programs to send the low-side output through the high-side output pin and the high-side output through the low side output pin.

For example, PWM_AH0 is programmed to zero with PWM_CHANCFG.MODELSA = 0. With PWM_ACTL.DISLO = 1 and PWM_ACTL.XOVR = 1 the low-side is being toggled even though the output is disabled, but nothing is toggling on the high-side pins. What is happening is that the high-side output is now coming out through the low-side (and vice versa). What is toggling on the low-side pin is the 50% duty high side output pulse (less the dead-time if any).

The following figure shows this example. In case 1 PWM_ACTL.XOVR = 0 and in case 2 PWM_ACTL.XOVR = 1.

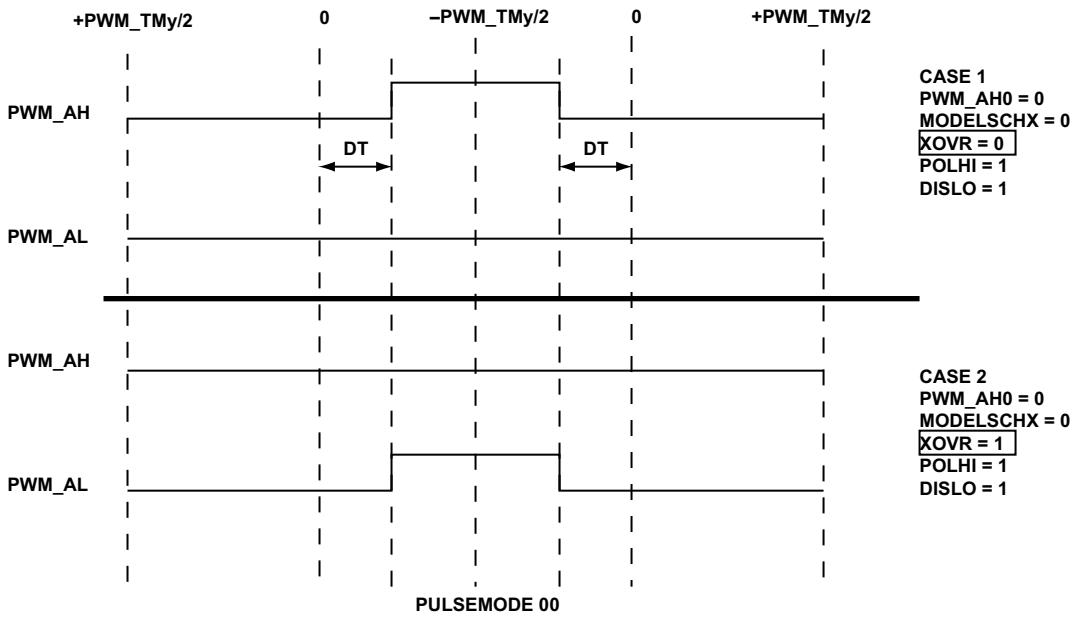


Figure 18-15: XOVR and DISHI/DISLO Functionality

Brushless DC Motor (Electronically Commutated Motor) Control

In the control of an electronically commutated motor (ECM), only two inverter legs are switched at any time. Often, the high-side device in one leg must be switched on at the same time as the low-side driver in a second leg. Therefore, by programming identical duty cycles values for two PWM channels (for example, PWM_CHA = PWM_CHB) and setting the `PWM_BCTL.XOVR` bit to crossover the BH/BL pair if PWM signals, it is possible to turn on the high-side switch of phase A and the low-side switch of phase B at the same time.

To control ECM, normally the third inverter leg (phase C in this example) is disabled for a number of PWM cycles. To implement this function, both the `PWM_CH` and `PWM_CL` outputs are disabled by setting the `PWM_CCTL.DISHI` and `PWM_CCTL.DISLO` bits.

In normal ECM operation, each inverter leg is disabled for certain time periods so that the PWM channel registers change based on the position of the rotor shaft (motor commutation).

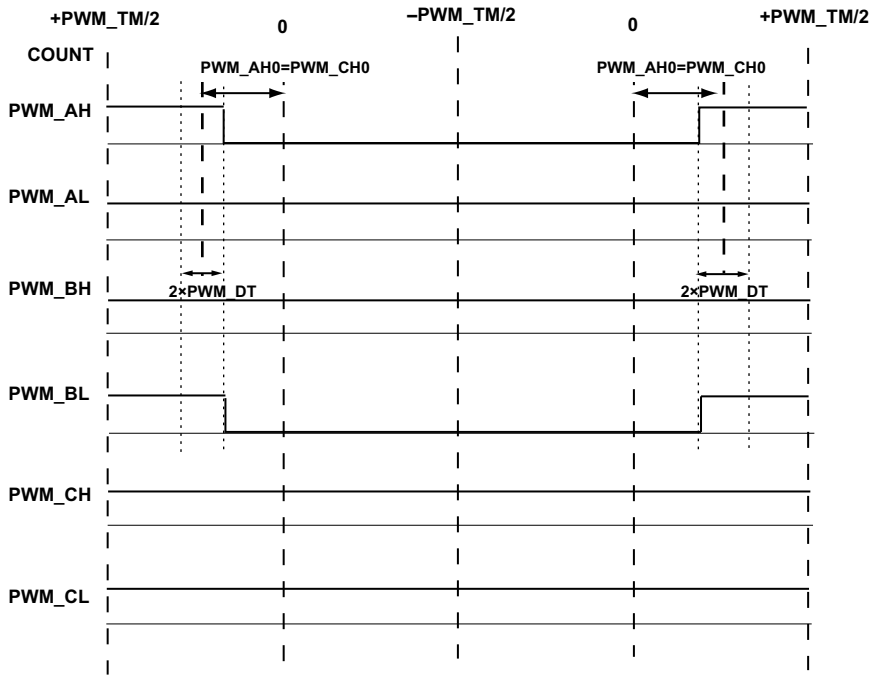


Figure 18-16: ECM Control

For the situation illustrated in the figure, an appropriate value for the PWM_SEG register is 0x00A7. In normal ECM operation, each inverter leg is disabled for certain lengths of time, such that the PWM_SEG register is changed, based upon the position of the rotor shaft (motor commutation).

Gate Drive Unit

The Gate Drive Unit of the PWM adds features that simplify the design of isolated gate drive circuits for PWM inverters. If a transformer coupled power device gate drive amplifier is used then the active PWM signal must be chopped at a high frequency. The PWM_CHOPCFG register allows the programming of this high frequency chopping mode. The chopped active PWM signals may be required for the high-side drivers only, for the low-side drivers only or for both the high-side and low-side switches. Therefore, independent control of this mode for both high and low-side switches is included with two separate control bits in the PWM_CHANCFG register.

Typical PWM output signals with high-frequency chopping enabled on both high-side and low-side signals are shown in the figure below. Chopping of the PWM outputs is enabled by setting bits in PWM_CHANCFG register. The high frequency chopping frequency is controlled by the 8-bit PWM_CHOPCFG.VALUE value in the PWM_CHOPCFG register. The period of this high frequency carrier is then given by,

$T_{chop} = [4 \times (CHOPDIV + 1)] \times t_{CK}$

and the chopping frequency is therefore an integral subdivision of the peripheral clock frequency:

$f_{chop} = f_{CK}/[4 \times (CHOPDIV + 1)]$

The PWM_CHOPCFG.VALUE value may range from 0 to 255, corresponding to a programmable chopping frequency rate from 122 kHz to 31.25 MHz for a 125 MHz, f_{CK} rate. The gate drive features must be programmed before operation of the PWM controller and are not changed during normal operation of the PWM controller. Following a reset, all bits of the PWM_CHANCFG register are cleared so that high frequency chopping is disabled, by default.

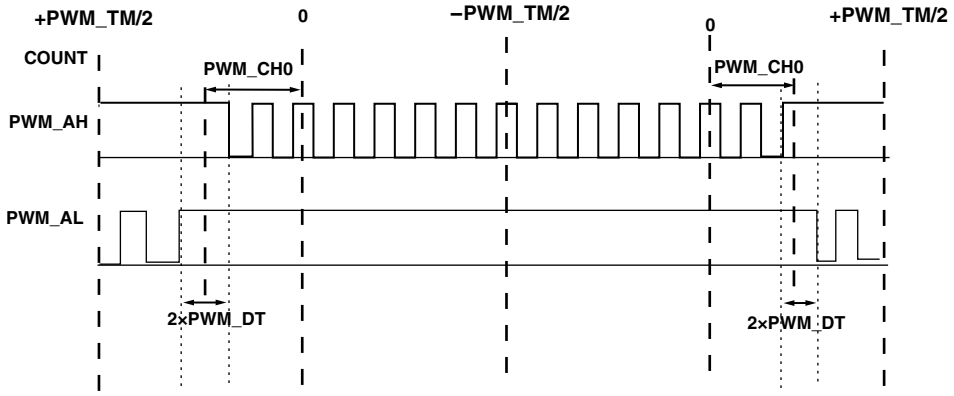


Figure 18-17: Hi-Side and Lo-Side Outputs With Gate Chop Enabled

Output Control Feature Precedence

It is important to understand the order in which output control features are applied to the PWM signal. The following hierarchy indicates the order (from most important to least important) in which signal features are applied to the PWM output signal.

1. Channel duty generation
2. Channel Crossover
3. High-side/Low-side disable
4. Emergency dead time insertion
5. Gate drive chopping
6. Polarity

Sync Operation

The PWM_SYNC signal can be internally generated as a function of PWM_TM0.VALUE and PWM_SYNC_WID.VALUE or can be input externally. Multiple PWM configurations can be established with each PWM operating with its own independent PWM_SYNC or from its own or shared external PWM_SYNC signal. The external PWM_SYNC can be synchronous to the internal clock as in the case of a primary PWM generating an internal

PWM_SYNC signal which drives the secondary PWM_SYNC_IN pin. The external PWM_SYNC can also be asynchronous to the internal clock as is typically the case of an off-chip PWM_SYNC signal used to drive each PWM's PWM_SYNC_IN pin.

Internal PWM SYNC Generation

The PWM controller produces an output PWM synchronization pulse at a rate equal to period of a selected PWM timer. Programming the PWM_CTL.INTSYNCREF field controls this selection.

Further, if the other timers are running with a non-zero DELAY offset in relation to PWMTMR0 and the PWM_SYNC pulse is referenced to any of these timers, then the PWM_SYNC pulses are generated at their respective period boundaries which has the lag-lead offset compared to PWMTMR0.

This pulse is available for external use at the PWM_SYNC_OUT pin. The width of the PWM_SYNC pulse is programmable by the 10-bit read/write PWM_SYNC_WID register. The width of the PWM_SYNC pulse, TPWM_SYNC, is given by the following equation.

$t_{PWMSYNC} = t_{SCLK} \times (PWMSYNCWT + 1)$

The width of the pulse is programmable from t_{CK} to $1024t_{CK}$ (corresponding to 8 ns to 8.19 µs for a f_{CK} rate of 125 MHz). Following a reset, the PWM_SYNC_WID register contains 0x3FF (1023 decimal) so that the default PWM_SYNC width is 8.19 µs, for a 125 MHz f_{CK}.

External PWM SYNC Generation

By setting the PWM_CTL.EXTSYNC bit, the PWM is set up in a mode to expect an external PWM_SYNC on the PWM_SYNC_IN pin. The external PWM_SYNC only determines the operation of the main timer PWMTMR0.

The external sync should be synchronized by setting the PWM_CTL.EXTSYNCSEL bit to 0 (assumes the external PWM_SYNC selected is asynchronous).

The external PWM_SYNC period is expected to be an integer multiple of the value of the PWM_TM0 period register. When the rising edge of the external PWM_SYNC is detected, the PWMTMR0 timer is restarted at the beginning of its period. If the external PWM_SYNC period is not exactly an integer multiple of the internal PWM_SYNC, the behavior of the PWM channel outputs which are referenced to PWMTMR0 are clipped.

The effect latency from PWM_SYNC_IN to the PWM outputs is about three clock cycles in synchronous mode, and five clock cycles in asynchronous mode.

Event Control

The PWM output signals can be shut-off in a number of different ways. Two trip inputs pwm_trip0b, and pwm_trip1b are provided, each of which can be mapped to provide either a temporary or permanent shutdown on any pair of channel outputs. This shutdown mechanism is asynchronous so that the associated PWM output disable circuitry does not go through any clocked logic, ensuring correct PWM shutdown

even in the event of a loss of the processor clock. In addition to the hardware shutdown features, the PWM system may be shutdown in software by means of the PWM_CTL.SWTRIP bit.

Status information about the PWM is available to the user in the PWM_STAT register, which stores all status bits, including raw interrupt status bits. In particular, the period boundary of each timer is available, as well as status bits that indicate whether the operation is in the first half or the second half of the timer. Additionally the TRIP status is also available.

The PWM_IMSK and PWM_ILAT registers allow masking and show masked interrupt status bits respectively. The interrupt bits are latched and held on the interrupt event and the software must write a 1 to clear the interrupt bit, usually during the Interrupt Service routine.

Trip Control Unit

The PWM Trip unit processes hardware or software fault conditions and shuts down the PWM channel outputs immediately on the occurrence of these conditions. This shut down mechanism can be enabled separately for each channel. The design also allows for a self-restart mechanism to be enabled on a channel. Self-restart re-enables the channel outputs following the fault condition (allowed only on hardware trips) when the PWMTMR-y that the channel is using reaches its period boundary.

There are 2 external hardware sources that can indicate a hardware fault condition:

1. $\overline{\text{PWMTRIP0}}$ input pin
2. $\overline{\text{PWMTRIP1}}$ input pin

These are active low inputs where a falling edge on either of these pins indicates a fault condition.

To enable the trip unit to shut down a particular channel's output in response to the fault event on either of these $\overline{\text{PWM\_TRIPn}}$ pins, program the PWM_TRIPCFG.ENOA bit corresponding to that channel.

The PWM_TRIPCFG.MODEOA bits must be programmed to specify the restart mechanism for a channel that has been tripped.

1. If PWM_TRIPCFG.MODEOA = 0, once tripped, a trip condition is registered on this channel in the PWM_STAT.FLTTRIPA bit and the outputs of that channel are immediately shut down. This is called a FAULT TRIP condition. To resume channel output when a FAULT TRIP occurs, clear the PWM_STAT.FLTTRIPA bit by writing a 1 to it. Note that the bit cannot be cleared by a processor write if the trip condition is still active. The RAW trip status is available for both the pins in the PWM_STAT.RAWTRIPO register bits.
2. If PWM_TRIPCFG.MODEOA = 1, once tripped, a trip condition is registered on this channel in the PWM_STAT.SRTRIPA bit and the outputs of that channel are immediately shut down. This is called a self-restart trip condition. At the next period boundary of the PWMTMR-y that the channel is using, if the tripping condition is not active, the design clears the status register bit and restores the outputs.

The trip input pins should have an external pull-down resistor on the chip pin, so that if the pin becomes unconnected the PWM will be disabled.

In addition to the hardware trip conditions, a global software trip bit in the PWM_CTL register allows for a software forced FAULT TRIP condition. When the global software trip bit is set to 1, irrespective of the values in the PWM_TRIPCFG register, it sets all the PWM_STAT.FLTTRIPA bits in the PWM_STAT register and also gates the channel outputs. To remove the trip condition from the channel, a W1C should be performed on the particular channel's PWM_STAT.FLTTRIPA bit.

If the PWM_TRIPCFG.ENOA bit is set to 1 to, for any channel, then the occurrence of a fault condition on the PWMTRIPy bit is logged in the PWM_STAT.FLTTRIPA register bit. If the corresponding PWM_IMSK.TRIP0 bit = 1, then an interrupt is generated. Note that tripping a channel output doesn't interfere with PWM_SYNC generation.

The following figure shows a scenario where PWMTRIP0 is enabled on channel A as SELF RESTART TRIP. Channel A works with PWM_CHANCFG.POLAH = 1. Note that in Period 2, PWM_AH is full ON modulated, and tries to rise at the period boundary itself where the self-restart occurs for the channel. However, since the low-side output of the channel was only recently removed due to a trip, the rise edge on PWM_AH is delayed until the emergency dead-time period is over. PWMTRIP1 is enabled on channel B as FAULT TRIP. Channel B works with PWM_CHANCFG.POLAH = 0. PWMTRIP1 stays low for a long time, and because of this the first processor write to re-enable the channel output fails. The second one passes since the fault-condition has gone away.

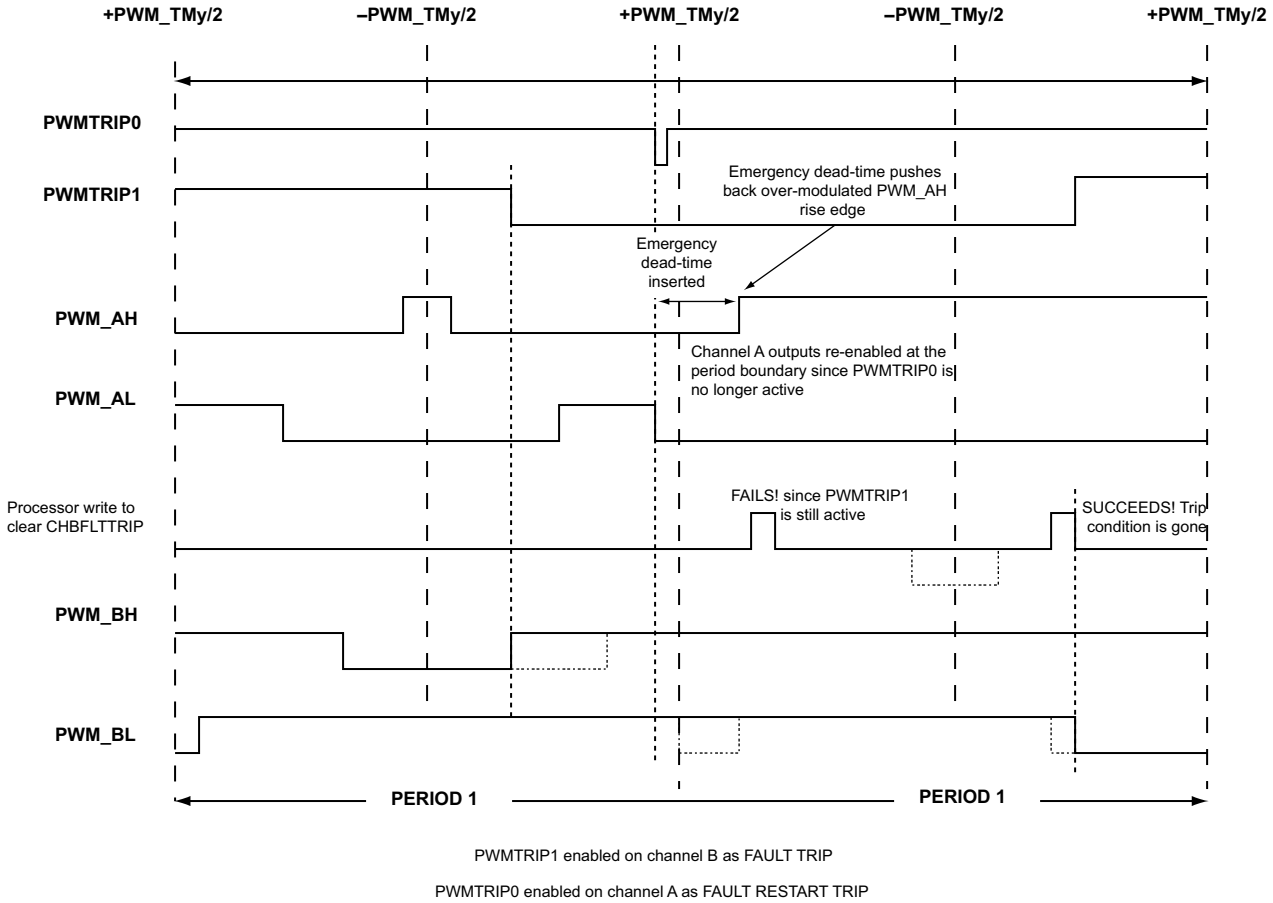


Figure 18-18: Operation Under Hardware Fault Conditions

NOTE:

Dead-time is ensured on re-enabling the channel outputs after trip.

NOTE:

Programs should not allow changes in the configuration/enable bits of `PWM_TRIPCFG` register (which select between trip enable and disable) within ± 10 clock cycles of the toggling of external trip pulse. If this time frame is not followed, then unexpected behavior occurs.

Programming Model

The following sections provide general (and some application specific) programming steps for configuring and using the PWM module.

- *Programming Model for 3-Phase AC Motor Control*

Programming Model for 3-Phase AC Motor Control

The *PWM Module and Interaction with System* figure shows how the PWM unit (green) interfaces to both software (blue) and hardware (yellow). The software configures the unit, calculates duty cycles (Duty A, Duty B, Duty C), and services the interrupts generated by the module (PWM Sync IRQ, TRIP IRQ). The hardware applies the gate signals (AH, AL, BH, BL, CH, CL) to the inverter and provides an overcurrent trip signal back to the unit (TRIP0).

The typical 3-phase AC motor configuration shown in the *PWM Module and Interaction with System* figure applies for both permanent magnet and induction motor types.

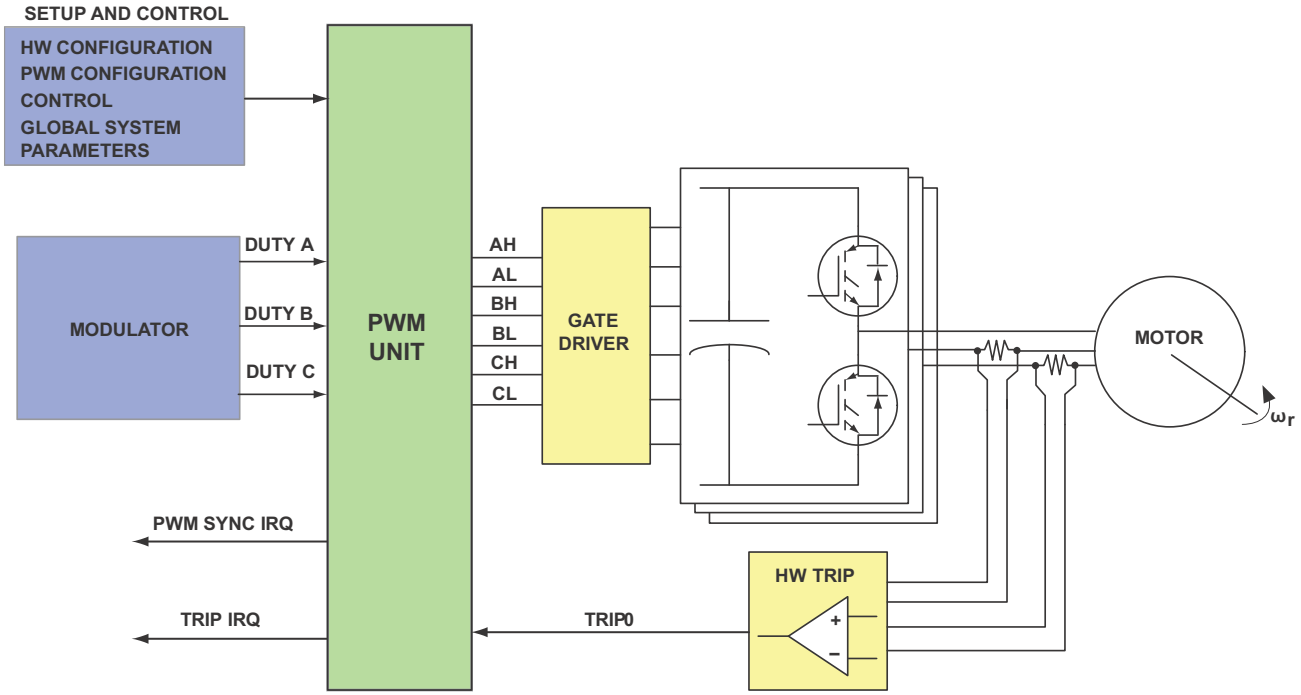


Figure 18-19: PWM Module and Interaction with System

System Parameters

The following system parameters (characteristics) influence the module configuration for this application. This example system has/uses:

- One 3-phase AC machine
- B6 inverter
- SVPWM, including both linear- and over-modulation
- Switching frequency of 20 kHz
- Dead time of 1us
- Trip signal generated by hardware
- Active high level gate drive
- Core frequency of 200 MHz
- Peripheral clock of 100 MHz

System State Sequencing

Managing the system state and sequence of states is critically important when programming the PWM module. The *PWM System States* figure provides an overview of these states.

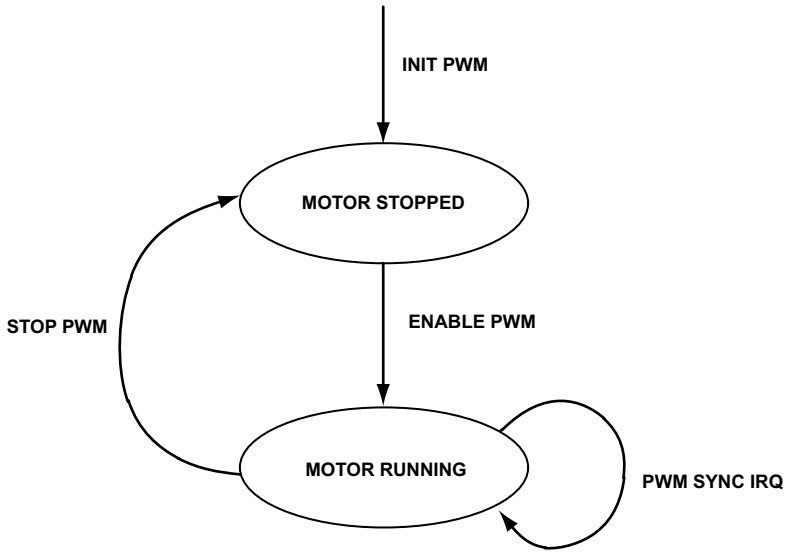


Figure 18-20: PWM System States

As shown in this state diagram, the module configuration is updated on state transitions (indicated by the arrows). The transitions are initialization, motor start, PWM sync interrupt (on each), and motor stop. These transitions are discussed in detail in the following sections.

- *PWM Initialization for Motor Control*
- *PWM Enable for Motor Control*
- *PWM Response to Sync Interrupt for Motor Control*
- *PWM Disable (and Stop the Motor) for Motor Control*

PWM Initialization for Motor Control

The processor should do the following programming at power up and repeat this programming whenever the PWM and system must be brought into a known (safe) state.

1. Place the PWM module in a safe state and set up synchronization of the module.

   ADDITIONAL INFORMATION: The following bitwise operations on the `PWM_CTL` and `PWM_CHANCFG` register accomplish this PWM state:

   ```
   PWM_CTL &= 0xFFE0FF08
   PWM_CTL |= 0x20000
   ```

```
PWM_CHANCFG &= 0x80808080
PWM_CHANCFG |= 0x24242424
```

ADDITIONAL INFORMATION: These operations achieve the following bit settings:

- Disable PWM (PWM_CTL.GLOBEN = 0)
- All phases must run with same phase, disable delay for channels A, B, C, D (PWM_CTL.DLYAEN through PWM_CTL.DLYDEN =0)
- Use internal synchronization by timer TMR0 (PWM_CTL.EXTSYNC =0, PWM_CTL.EXTSYNCSEL =1)
- All phases must be synchronized by the same timer, TMR0 (PWM_CTL.INTSYNCREF = b#000)
- Low side is always the inverse of High side (PWM_CHANCFG.POLAL through PWM_CHANCFG.POLDL = 1)
- System uses active high gate driver (PWM_CHANCFG.ENCHOPAH throughPWM_CHANCFG.ENCHOPDH =1)
- Pulse transformer is not used: disable gate chopping (PWM_CHANCFG.ENCHOPAL through PWM_CHANCFG.ENCHOPDL =0)

2. Set up the trip and associated interrupts.

   ADDITIONAL INFORMATION: The following bitwise operations on the PWM_TRIPCFG and PWM_ILAT register accomplish this PWM state:

```
PWM_TRIPCFG &= 0xF0F0F0F0
PWM_TRIPCFG |= 0x1010101
```

```
PWM_ILAT &= 0xFFE0FFFC
PWM_ILAT |= 0x1
```

ADDITIONAL INFORMATION: These operations achieve the following bit settings:

- All phases must shut down simultaneously in case of fault: (PWM_TRIPCFG.ENOA through PWM_TRIPCFG.ENOD =0, PWM_TRIPCFG.MODE0A through PWM_TRIPCFG.MODE0D =0, PWM_TRIPCFG.EN1A through PWM_TRIPCFG.EN1D =0, PWM_TRIPCFG.MODE1A through PWM_TRIPCFG.MODE1D =0)
- Enable TRIP0 as fault trigger for all channels. (PWM_TRIPCFG.ENOA through PWM_TRIPCFG.MODE1D =1)
- For thermal control and synchronization, SW intervention is needed at trip. Do not use automatic restart of any channels
- Generate an interrupt at trip on TRIP0. (PWM_ILAT.TMROPER = 1)

3. Configure the PWM channels.

ADDITIONAL INFORMATION:

ADDITIONAL INFORMATION: The following bitwise operations on the PWM_TRIPCFG and PWM_ILAT register accomplish this PWM state:

```
PWM_DT = 0x32
PWM_TM0 =0x9C4
PWM_ACTL &= 0xFFFFF0000
PWM_BCTL &= 0xFFFFF0000
PWM_CCTL& = 0xFFFFF0000
PWM_AH0  = 0x0
```

```
PWM_BH0  = 0x0
PWM_CH0  = 0x0
```

ADDITIONAL INFORMATION: These operations achieve the following bit settings:

- Configure a dead time of 1 µs (PWM_DT = 0x32).
- Configure a PWM frequency of 20 kHz (PWM_TM0 = 0x9C4).
- Disable all outputs (PWM_ACTL.DISHI- PWM_CCTL.DISHI = 0, PWM_ACTL.DISLO through PWM_CCTL.DISLO = 0)
- Use conventional PWM, disable crossover (PWM_ACTL.XOVR through PWM_CCTL.XOVR = 0)
- Use symmetrical pulse position on all outputs (PWM_ACTL.PULSEMODEHI through PWM_CCTL.PULSEMODEHI = 0, PWM_ACTL.PULSEMODELO through PWM_CCTL.PULSEMODELO = 0)
- Set an initial duty-cycle of 50% (PWM_AH0 through PWM_CH0 = 0x0)

PWM Enable for Motor Control

The processor must do the following programming to enable the PWM before starting the motor.

1. Start the PWM timer TMR0.

   ADDITIONAL INFORMATION: The following bitwise operation on the PWM_CTL register accomplishes this PWM state:

   ```
   PWM_CTL |= 0x1
   ```

   ADDITIONAL INFORMATION: This operation achieves the following bit setting:

   - Ensable PWM (PWM_CTL.GLOBEN =1)

2. Enable six PWM outputs.

   ADDITIONAL INFORMATION: The following bitwise operations on the PWM_ACTL through PWM_CCTL registers accomplish this PWM state:

   ```
   PWM_ACTL| = 0x3
   PWM_BCTL| = 0x3
   PWM_CCTL| = 0x3
   ```

   ADDITIONAL INFORMATION: These operations achieve the following bit settings:

   - Enable high and low side channel outputs (PWM_ACTL.DISHI through PWM_CCTL.DISHI = 1, PWM_ACTL.DISLO through PWM_CCTL.DISLO = 1)

3. Enable the PWM TRIP0 interrupt.

   ADDITIONAL INFORMATION: The following bitwise operation on the PWM_ILAT register accomplishes this PWM state:

   ```
   PWM_ILAT |= 0x1
   ```

ADDITIONAL INFORMATION: These operations achieve the following bit settings:

- Enable PWM TRIP0 interrupt (`PWM_ILAT.TRIP0` =1)

PWM Response to Sync Interrupt for Motor Control

When the PWM sync interrupt occurs, the processor may need to update to the PWM duty cycle with a value calculated by the motor control algorithm. This application uses symmetric pulses position and uses dependent High and Low side outputs, so only one register needs to be updated for each phase.

1. Write new duty cycle value (calculated by motor control algorithm) to the timer when the sync interrupt occurs.

   ADDITIONAL INFORMATION: The following bitwise operations on the `PWM_AH0` through `PWM_CH0` registers accomplish this PWM state:

   ```
   PWM_AH0 = Duty_A_mc_algorithm_current_value
   ```

   ADDITIONAL INFORMATION:

   ```
   PWM_BH0 = Duty_B_mc_algorithm_current_value
   ```

   ADDITIONAL INFORMATION:

   ```
   PWM_CH0 = Duty_C_mc_algorithm_current_value
   ```

   ADDITIONAL INFORMATION:

   ADDITIONAL INFORMATION: These operations achieve the following bit settings:

PWM Disable (and Stop the Motor) for Motor Control

The processor should do the following programming to stop the motor, disable the PWM, and disable PWM interrupts. These actions place the PWM and system in a safe, passive state.

1. Disable the PWM timer.

   ADDITIONAL INFORMATION: The following bitwise operation on the PWM_CTL register accomplish this PWM state:

   ```
   PWM_CTL &= 0xFFFFFFFE
   ```

   ADDITIONAL INFORMATION: This operation achieves the following bit setting:

   - Disable PWM (PWM_CTL.GLOBEN =0)

2. Disable all PWM outputs.

   ADDITIONAL INFORMATION: The following bitwise operations on the PWM_ACTL through PWM_CCTL registers accomplish this PWM state:

   ```
   PWM_ACTL &= 0xFFFFFFFC
   PWM_BCTL &= 0xFFFFFFFC
   PWM_CCTL &= 0xFFFFFFFC
   ```

   ADDITIONAL INFORMATION: These operations achieve the following bit settings:

   - Disable PWM outputs PWM_ACTL.DISHI through PWM_CCTL.DISHI = 1, PWM_ACTL.DISLO through PWM_CCTL.DISLO = 0)

3. Set the PWM duty-cycle to 50%.

   ADDITIONAL INFORMATION: The following bitwise operations on the PWM_AH0 through PWM_CH0 registers accomplish this PWM state:

   ```
   PWM_AH0  = 0x0
   PWM_BH0  = 0x0
   PWM_CH0  = 0x0
   ```

   ADDITIONAL INFORMATION: These operations achieve the following bit settings:

   - Set PWM duty cycle to 50%. (PWM_AH0.DUTY through PWM_CH0.DUTY = 0)

4. Disable the PWM TRIP0 interrupt.

   ADDITIONAL INFORMATION: The following bitwise operation on the PWM_ILAT register accomplish this PWM state:

   ```
   PWM_ILAT &= 0xFFFFFFFE
   ```

   ADDITIONAL INFORMATION: These operations achieve the following bit settings:

   - Disable PWM TRIP0 interrupt (PWM_ILAT.TRIP0 = 0)

ADSP-BF60x PWM Register Descriptions

Pulse-Width Modulator (PWM) contains the following registers.

Table 18-5: ADSP-BF60x PWM Register List

Name	Description
PWM_CTL	Control Register
PWM_CHANCFG	Channel Config Register
PWM_TRIPCFG	Trip Config Register
PWM_STAT	Status Register
PWM_IMSK	Interrupt Mask Register
PWM_ILAT	Interrupt Latch Register
PWM_CHOPCFG	Chop Configuration Register
PWM_DT	Dead Time Register
PWM_SYNC_WID	Sync Pulse Width Register
PWM_TM0	Timer 0 Period Register
PWM_TM1	Timer 1 Period Register
PWM_TM2	Timer 2 Period Register
PWM_TM3	Timer 3 Period Register
PWM_TM4	Timer 4 Period Register
PWM_DLYA	Channel A Delay Register
PWM_DLYB	Channel B Delay Register
PWM_DLYC	Channel C Delay Register
PWM_DLYD	Channel D Delay Register
PWM_ACTL	Channel A Control Register
PWM_AH0	Channel A-High Duty-0 Register

Table 18-5: ADSP-BF60x PWM Register List (Continued)

Name	Description
PWM_AH1	Channel A-High Duty-1 Register
PWM_AL0	Channel A-Low Duty-0 Register
PWM_AL1	Channel A-Low Duty-1 Register
PWM_BCTL	Channel B Control Register
PWM_BH0	Channel B-High Duty-0 Register
PWM_BH1	Channel B-High Duty-1 Register
PWM_BL0	Channel B-Low Duty-0 Register
PWM_BL1	Channel B-Low Duty-1 Register
PWM_CCTL	Channel C Control Register
PWM_CH0	Channel C-High Pulse Duty Register 0
PWM_CH1	Channel C-High Pulse Duty Register 1
PWM_CL0	Channel C-Low Pulse Duty Register 0
PWM_CL1	Channel C-Low Duty-1 Register
PWM_DCTL	Channel D Control Register
PWM_DH0	Channel D-High Duty-0 Register
PWM_DH1	Channel D-High Pulse Duty Register 1
PWM_DL0	Channel D-Low Pulse Duty Register 0
PWM_DL1	Channel D-Low Pulse Duty Register 1

Control Register

The PWM_CTL register enables the PWM, enables delay counters for the channels, and configures sync features. This register also provides support for tripping a PWM fault condition through software.

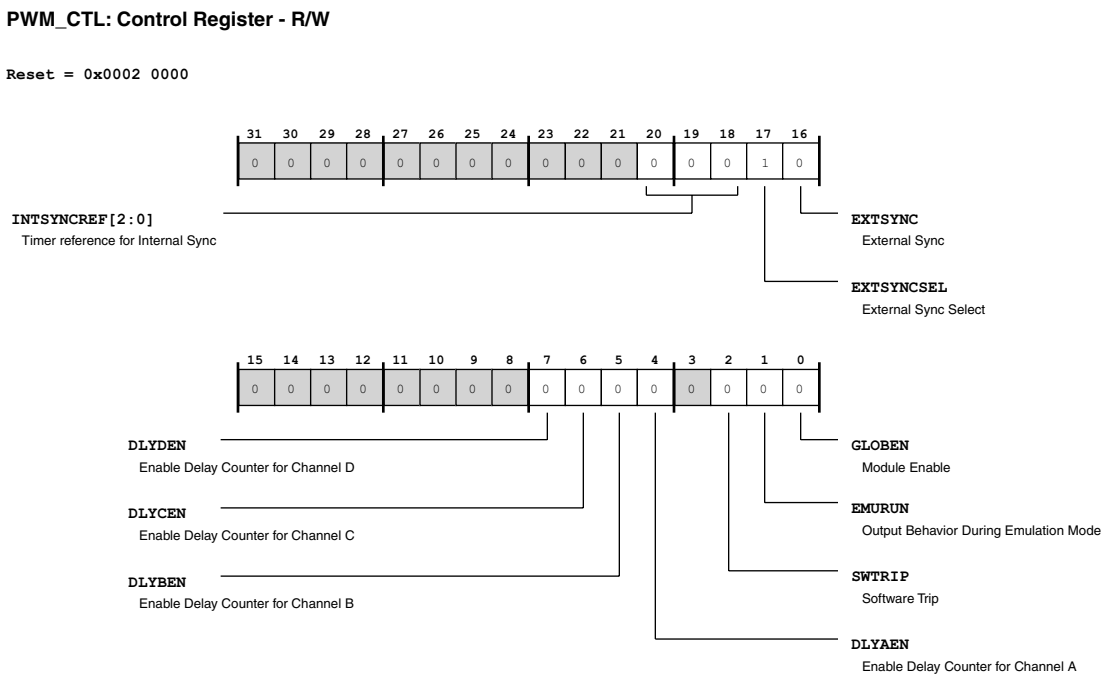


Figure 18-21: PWM_CTL Register Diagram

Table 18-6: PWM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20:18 (R/W)	INTSYNCREF	Timer reference for Internal Sync. The PWM_CTL.INTSYNCREF bits select the timer reference for the internal sync. Note that all other combinations reserved.	
		0	PWMTMR0 provides sync reference
		1	PWMTMR1 provides sync reference
		2	PWMTMR2 provides sync reference
		3	PWMTMR3 provides sync reference
		4	PWMTMR4 provides sync reference
17 (R/W)	EXTSYNCSEL	External Sync Select. The PWM_CTL.EXTSYNCSEL bit selects whether the external sync signal is synchronous or asynchronous. Note that latency in PWM sync response differs between asynchronous and synchronous external sync modes. For more information, see the PWM functional description.	
		0	Asynchronous External Sync
		1	Synchronous External Sync

Table 18-6: PWM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	EXTSYNC	External Sync. The PWM_CTL.EXTSYNC bit selects whether the PWM uses an external or internal sync signal for the main timer (PWMTMR0). Do not change the value of the PWM_CTL.EXTSYNC bit while the PWM is enabled (PWM_CTL.GLOBEN =1).	
		0	Internal sync used
		1	External sync used
7 (R/W)	DLYDEN	Enable Delay Counter for Channel D. The PWM_CTL.DLYDEN bit enables the Channel D delay counter, supporting phase-offset control. Do not change the value of the PWM_CTL.DLYDEN bit while the PWM is enabled (PWM_CTL.GLOBEN =1).	
		0	Disable
		1	Enable
6 (R/W)	DLYCEN	Enable Delay Counter for Channel C. The PWM_CTL.DLYCEN bit enables the Channel C delay counter, supporting phase-offset control. Do not change the value of the PWM_CTL.DLYCEN bit while the PWM is enabled (PWM_CTL.GLOBEN =1).	
		0	Disable
		1	Enable
5 (R/W)	DLYBEN	Enable Delay Counter for Channel B. The PWM_CTL.DLYBEN bit enables the Channel B delay counter, supporting phase-offset control. Do not change the value of the PWM_CTL.DLYBEN bit while the PWM is enabled (PWM_CTL.GLOBEN =1).	
		0	Disable
		1	Enable
4 (R/W)	DLYAEN	Enable Delay Counter for Channel A. The PWM_CTL.DLYAEN bit enables the Channel A delay counter, supporting phase-offset control. Do not change the value of the PWM_CTL.DLYAEN bit while the PWM is enabled (PWM_CTL.GLOBEN =1).	
		0	Disable
		1	Enable

Table 18-6: PWM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R0/W1A)	SWTRIP	Software Trip. The PWM_CTL.SWTRIP bit permits tripping a fault condition through software, shutting down PWM output. This bit always read as 0. If the PWM_CTL.SWTRIP bit and PWM_CTL.GLOBEN bit are set in the same write, the write does not trip the fault condition.	
		1	Force a Fault Trip Condition
1 (R/W)	EMURUN	Output Behavior During Emulation Mode. The PWM_CTL.EMURUN bit selects PWM output behavior during emulation mode.	
		0	Disable Outputs
		1	Enable Outputs
0 (R/W)	GLOBEN	Module Enable. The PWM_CTL.GLOBEN bit enables the PWM, enabling all timers and outputs. While this bit is enabled, processor code should not change the value of the PWM_CTL.DLYAEN bit, PWM_CTL.DLYBEN bit, PWM_CTL.DLYCEN bit, PWM_CTL.DLYDEN bit, PWM_CTL.EXTSYNCSEL bit, or any bits in the PWM_CHANCFG register. Note that there is a latency between PWM disable and the cessation of output waveforms. There is also a latency between PWM enable and start of output waveforms. For the latency description, see the PWM functional description.	
		0	Disable
		1	Enable

Channel Config Register

The PWM_CHANCFG register configures Channel A, B, C, and D reference timer selection, high and low side output features, and enables high frequency chopping operation. Do not change the value of any bits in the PWM register while the PWM is enabled (PWM_CTL.GLOBEN =1).

PWM_CHANCFG: Channel Config Register - R/W

Reset = 0x0000 0000

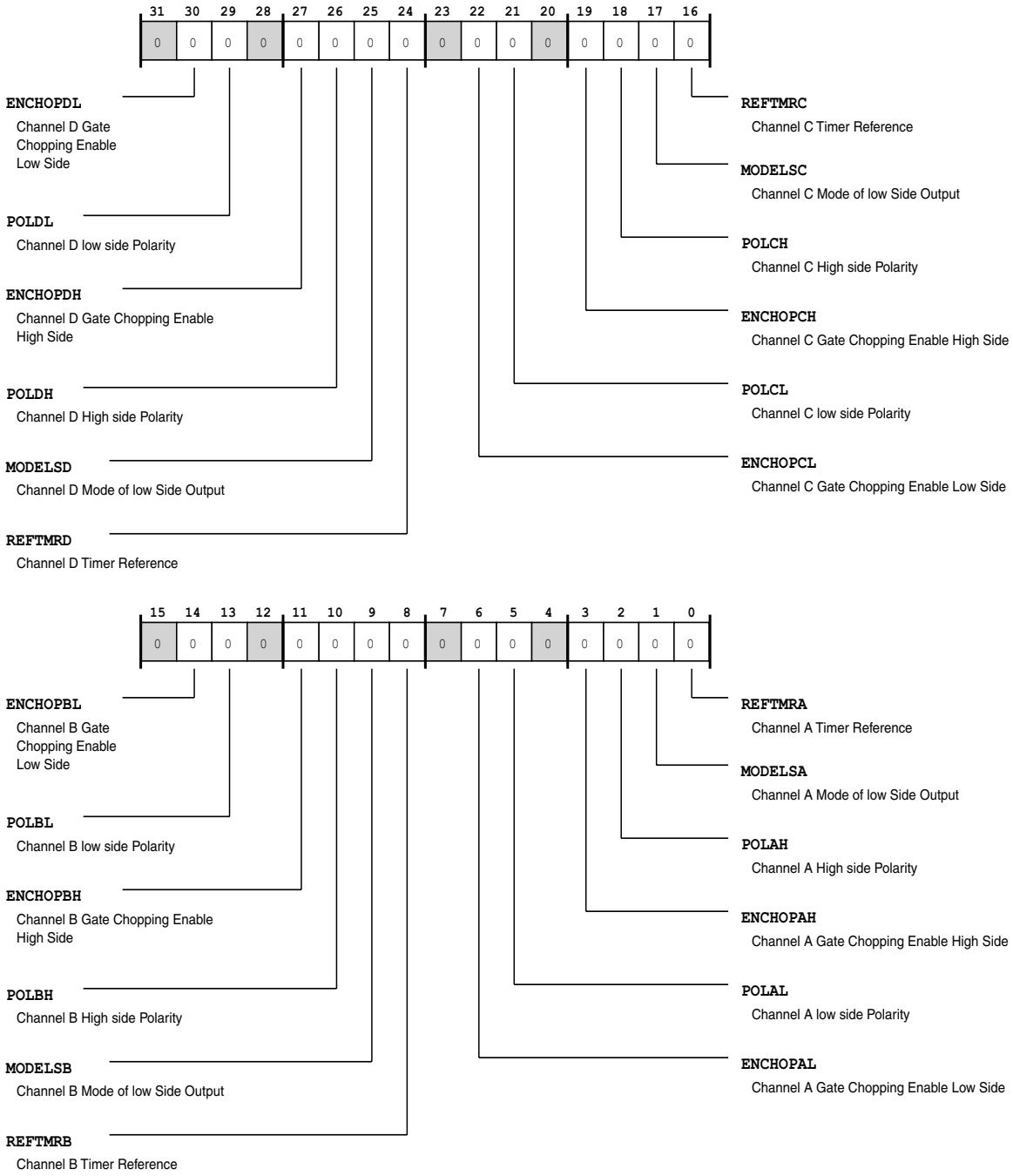


Figure 18-22: PWM_CHANCFG Register Diagram

Table 18-7: PWM_CHANCFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	ENCHOPDL	Channel D Gate Chopping Enable Low Side. The PWM_CHANCFG.ENCHOPDL bit enables mixing of the Channel D low side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel D Low Side
		1	Enable Chopping Channel D Low Side
29 (R/W)	POLDL	Channel D low side Polarity. The PWM_CHANCFG.POLDL bit selects the Channel D low side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
27 (R/W)	ENCHOPDH	Channel D Gate Chopping Enable High Side. The PWM_CHANCFG.ENCHOPDH bit enables mixing of the Channel D high side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel D High Side
		1	Enable Chopping Channel D High Side
26 (R/W)	POLDH	Channel D High side Polarity. The PWM_CHANCFG.POLDH bit selects the Channel D high side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
25 (R/W)	MODELSD	Channel D Mode of low Side Output. The PWM_CHANCFG.MODELSD bit selects whether the low side output waveform is based on independent controls or whether the low side output depends on the high side output controls. When PWM_CHANCFG.MODELSD =0, the low side output is an inverted form of the high side output, which is generated using the PWM_BH0 and PWM_BH1 registers for pulse width, using the PWM_BCTL.PULSEMODEHI bits for pulse positioning, and PWM_CHANCFG.POLBH bits for polarity.	
		0	Invert of high output
		1	Independent control

Table 18-7: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
24 (R/W)	REFTMRD	Channel D Timer Reference. The PWM_CHANCFG.REFTMRD bit selects whether the PWM uses PWMTMR1 or PWMTMR0 as the reference timer for Channel D operation.	
		0	PWMTMR0 is Channel D reference
		1	PWMTMR1 is Channel D reference
22 (R/W)	ENCHOPCL	Channel C Gate Chopping Enable Low Side. The PWM_CHANCFG.ENCHOPCL bit enables mixing of the Channel C low side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel C Low Side
		1	Enable Chopping Channel C Low Side
21 (R/W)	POLCL	Channel C low side Polarity. The PWM_CHANCFG.POLCL bit selects the Channel C low side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
19 (R/W)	ENCHOPCH	Channel C Gate Chopping Enable High Side. The PWM_CHANCFG.ENCHOPCH bit enables mixing of the Channel C high side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel C High Side
		1	Enable Chopping Channel C High Side
18 (R/W)	POLCH	Channel C High side Polarity. The PWM_CHANCFG.POLCH bit selects the Channel C high side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High

Table 18-7: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	MODELSC	Channel C Mode of low Side Output. The PWM_CHANCFG.MODELSC bit selects whether the low side output waveform is based on independent controls or whether the low side output depends on the high side output controls. When PWM_CHANCFG.MODELSC =0, the low side output is an inverted form of the high side output, which is generated using the PWM_BH0 and PWM_BH1 registers for pulse width, using the PWM_BCTL.PULSEMODEHI bits for pulse positioning, and PWM_CHANCFG.POLBH bits for polarity.	
		0	Invert of high output
		1	Independent control
16 (R/W)	REFTMRC	Channel C Timer Reference. The PWM_CHANCFG.REFTMRC bit selects whether the PWM uses PWMTMR1 or PWMTMR0 as the reference timer for Channel C operation.	
		0	PWMTMR0 is Channel C reference
		1	PWMTMR1 is Channel C reference
14 (R/W)	ENCHOPBL	Channel B Gate Chopping Enable Low Side. The PWM_CHANCFG.ENCHOPBL bit enables mixing of the Channel B low side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel B Low Side
		1	Enable Chopping Channel B Low Side
13 (R/W)	POLBL	Channel B low side Polarity. The PWM_CHANCFG.POLBL bit selects the Channel B low side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
11 (R/W)	ENCHOPBH	Channel B Gate Chopping Enable High Side. The PWM_CHANCFG.ENCHOPBH bit enables mixing of the Channel B high side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel B High Side
		1	Enable Chopping Channel B High Side

Table 18-7: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	POLBH	Channel B High side Polarity. The PWM_CHANCFG.POLBH bit selects the Channel B high side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
9 (R/W)	MODELSB	Channel B Mode of low Side Output. The PWM_CHANCFG.MODELSB bit selects whether the low side output waveform is based on independent controls or whether the low side output depends on the high side output controls. When PWM_CHANCFG.MODELSB =0, the low side output is an inverted form of the high side output, which is generated using the PWM_BH0 and PWM_BH1 registers for pulse width, using the PWM_BCTL.PULSEMODEHI bits for pulse positioning, and PWM_CHANCFG.POLBH bits for polarity.	
		0	Invert of high output
		1	Independent control
8 (R/W)	REFTMRB	Channel B Timer Reference. The PWM_CHANCFG.REFTMRB bit selects whether the PWM uses PWMTMR1 or PWMTMR0 as the reference timer for Channel B operation.	
		0	PWMTMR0 is Channel B reference
		1	PWMTMR1 is Channel B reference
6 (R/W)	ENCHOPAL	Channel A Gate Chopping Enable Low Side. The PWM_CHANCFG.ENCHOPAL bit enables mixing of the Channel A low side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel A Low Side
		1	Enable Chopping Channel A Low Side
5 (R/W)	POLAL	Channel A low side Polarity. The PWM_CHANCFG.POLAL bit selects the Channel A low side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High

Table 18-7: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	ENCHOPAH	Channel A Gate Chopping Enable High Side. The PWM_CHANCFG.ENCHOPAH bit enables mixing of the Channel A high side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel A High Side
		1	Enable Chopping Channel A High Side
2 (R/W)	POLAH	Channel A High side Polarity. The PWM_CHANCFG.POLAH bit selects the Channel A high side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
1 (R/W)	MODELSA	Channel A Mode of low Side Output. The PWM_CHANCFG.MODELSA bit selects whether the low side output waveform is based on independent controls or whether the low side output depends on the high side output controls. When PWM_CHANCFG.MODELSA =0, the low side output is an inverted form of the high side output, which is generated using the PWM_AH0 and PWM_AH1 registers for pulse width, using the PWM_ACTL.PULSEMODEHI bits for pulse positioning, and PWM_CHANCFG.POLAH bits for polarity.	
		0	Invert of high output
		1	Independent control
0 (R/W)	REFTMRA	Channel A Timer Reference. The PWM_CHANCFG.REFTMRA bit selects whether the PWM uses PWMTMR1 or PWMTMR0 as the reference timer for Channel A operation.	
		0	PWMTMR0 is Channel A reference
		1	PWMTMR1 is Channel A reference

Trip Config Register

The PWM_TRIPCFG register configures Channel A, B, C, and D trip operation for trip inputs TRIP0 and TRIP1.

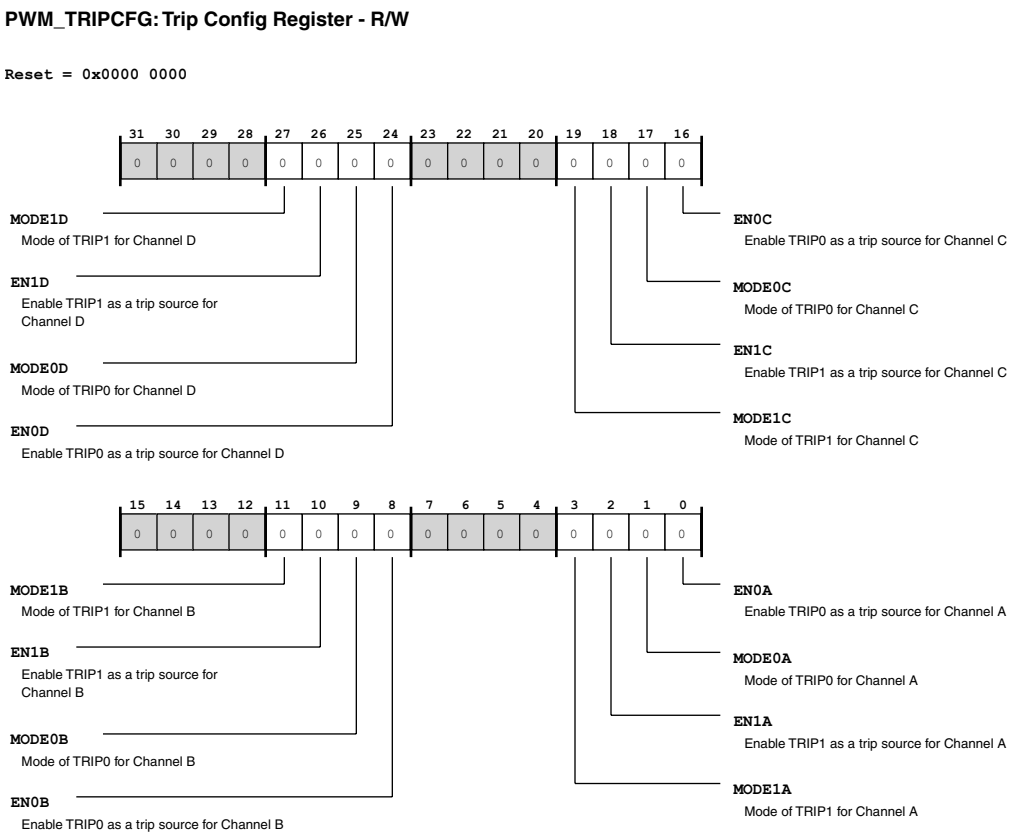


Figure 18-23: PWM_TRIPCFG Register Diagram

Table 18-8: PWM_TRIPCFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
27 (R/W)	MODE1D	Mode of TRIP1 for Channel D. The PWM_TRIPCFG.MODE1D bit selects the trip mode of TRIP1 for Channel D. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP1 Input
		1	Self Restart on TRIP1 Input
26 (R/W)	EN1D	Enable TRIP1 as a trip source for Channel D. The PWM_TRIPCFG.EN1D bit enables TRIP1 as a trip source for Channel D.	
		0	Disable TRIP1 for Channel D
		1	Enable TRIP1 for Channel D

Table 18-8: PWM_TRIPCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
25 (R/W)	MODE0D	Mode of TRIP0 for Channel D. The PWM_TRIPCFG.MODE0D bit selects the trip mode of TRIP0 for Channel D. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP0 Input
		1	Self Restart on TRIP0 Input
24 (R/W)	EN0D	Enable TRIP0 as a trip source for Channel D. The PWM_TRIPCFG.EN0D bit enables TRIP0 as a trip source for Channel D.	
		0	Disable TRIP0 for Channel D
		1	Enable TRIP0 for Channel D
19 (R/W)	MODE1C	Mode of TRIP1 for Channel C. The PWM_TRIPCFG.MODE1C bit selects the trip mode of TRIP1 for Channel C. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP1 Input
		1	Self Restart on TRIP1 Input
18 (R/W)	EN1C	Enable TRIP1 as a trip source for Channel C. The PWM_TRIPCFG.EN1C bit enables TRIP1 as a trip source for Channel C.	
		0	Disable TRIP1 for Channel C
		1	Enable TRIP1 for Channel C
17 (R/W)	MODE0C	Mode of TRIP0 for Channel C. The PWM_TRIPCFG.MODE0C bit selects the trip mode of TRIP0 for Channel C. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP0 Input
		1	Self Restart on TRIP0 Input
16 (R/W)	EN0C	Enable TRIP0 as a trip source for Channel C. The PWM_TRIPCFG.EN0C bit enables TRIP0 as a trip source for Channel C.	
		0	Disable TRIP0 for Channel C
		1	Enable TRIP0 for Channel C

Table 18-8: PWM_TRIPCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	MODE1B	Mode of TRIP1 for Channel B. The PWM_TRIPCFG.MODE1B bit selects the trip mode of TRIP1 for Channel B. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP1 Input
		1	Self Restart on TRIP1 Input
10 (R/W)	EN1B	Enable TRIP1 as a trip source for Channel B. The PWM_TRIPCFG.EN1B bit enables TRIP1 as a trip source for Channel B.	
		0	Disable TRIP1 for Channel B
		1	Enable TRIP1 for Channel B
9 (R/W)	MODE0B	Mode of TRIP0 for Channel B. The PWM_TRIPCFG.MODE0B bit selects the trip mode of TRIP0 for Channel B. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP0 Input
		1	Self Restart on TRIP0 Input
8 (R/W)	EN0B	Enable TRIP0 as a trip source for Channel B. The PWM_TRIPCFG.EN0B bit enables TRIP0 as a trip source for Channel B.	
		0	Disable TRIP0 for Channel B
		1	Enable TRIP0 for Channel B
3 (R/W)	MODE1A	Mode of TRIP1 for Channel A. The PWM_TRIPCFG.MODE1A bit selects the trip mode of TRIP1 for Channel A. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP1 Input
		1	Self Restart on TRIP1 Input
2 (R/W)	EN1A	Enable TRIP1 as a trip source for Channel A. The PWM_TRIPCFG.EN1A bit enables TRIP1 as a trip source for Channel A.	
		0	Disable TRIP1 for Channel A
		1	Enable TRIP1 for Channel A

Table 18-8: PWM_TRIPCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	MODE0A	Mode of TRIP0 for Channel A. The PWM_TRIPCFG.MODE0A bit selects the trip mode of TRIP0 for Channel A. In fault-trip mode (PWM_TRIPCFG.MODE0A =0), after the input is tripped, the trip status appears in the corresponding channels fault-trip status bit (for example, PWM_STAT.FLTTRIPA), and the PWM immediately shuts down outputs of that channel. After a fault trip occurs, when the trip condition is no longer active, the processor may cause channel outputs to resume by completing a write-1-to-clear the corresponding fault-trip status bit. The raw (input level) trip input state is available from the PWM_STAT.RAWTRIP0 and PWM_STAT.RAWTRIP0 bits. In self-restart mode (PWM_TRIPCFG.MODE0A =1), after the input is tripped, the trip status appears in the corresponding channels self-restart status bit (for example, PWM_STAT.SRTRIPA), and the PWM immediately shuts down outputs of that channel. On the next timer period boundary (of the PWMTMRx used by that channel), if the trip condition is not active, the PWM clears the status and restarts the channels output.	
		0	Fault Trip on TRIP0 Input
		1	Self Restart on TRIP0 Input
0 (R/W)	EN0A	Enable TRIP0 as a trip source for Channel A. The PWM_TRIPCFG.EN0A bit enables TRIP0 as a trip source for Channel A.	
		0	Disable TRIP0 for Channel A
		1	Enable TRIP0 for Channel A

Status Register

The PWM_STAT register indicates the PWM PWMTRIP1-0 fault and input level status, indicates the Channel A-D fault and self-restart status, and indicates the PWMTMR4-0 phase.

PWM_STAT: Status Register - R/W

Reset = 0x0000 0000

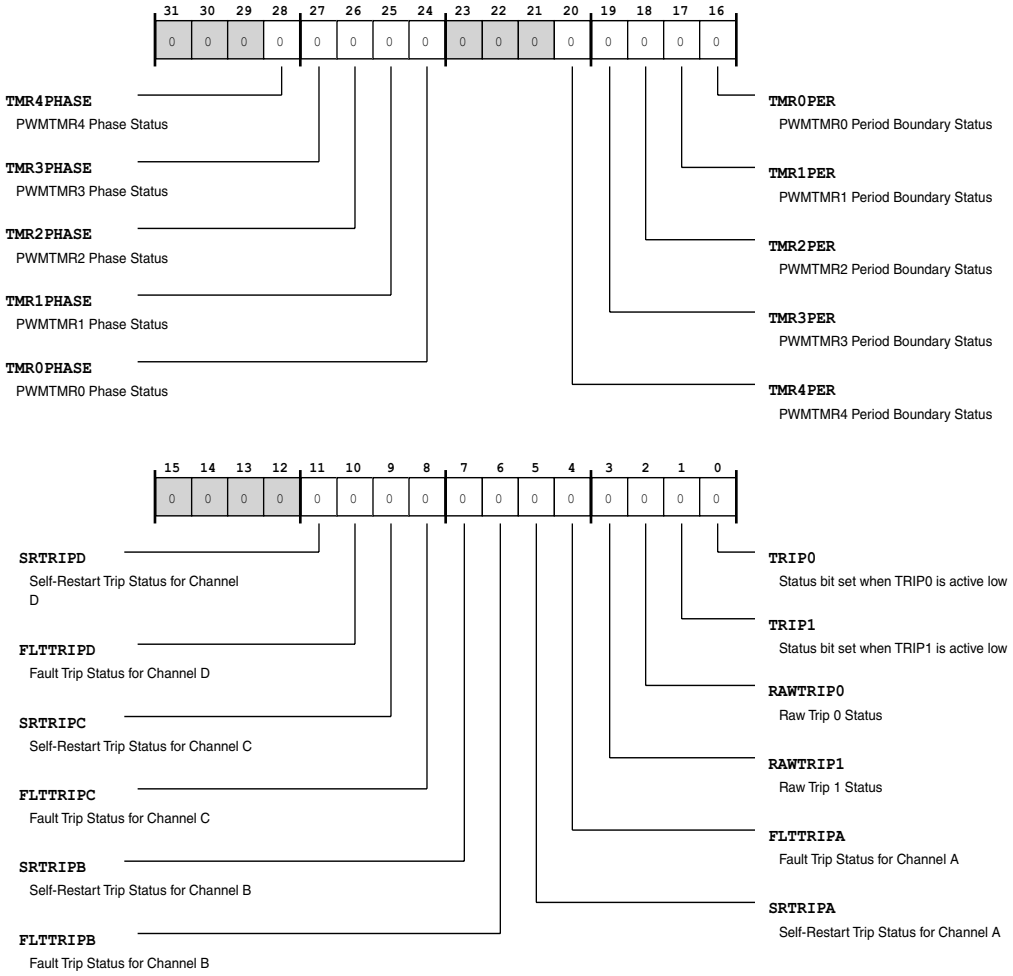


Figure 18-24: PWM_STAT Register Diagram

Table 18-9: PWM_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
28 (R/W1C)	TMR4PHASE	PWMTMR4 Phase Status. The PWM_STAT.TMR4PHASE bit indicates the current phase for the PWMTMR4 waveform.	
		0	1st Half Phase
		1	2nd Half Phase

Table 18-9: PWM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
27 (R/W1C)	TMR3PHASE	PWMTMR3 Phase Status. The PWM_STAT.TMR3PHASE bit indicates the current phase for the PWMTMR3 waveform.	
		0	1st Half Phase
		1	2nd Half Phase
26 (R/W1C)	TMR2PHASE	PWMTMR2 Phase Status. The PWM_STAT.TMR2PHASE bit indicates the current phase for the PWMTMR2 waveform.	
		0	1st Half Phase
		1	2nd Half Phase
25 (R/W1C)	TMR1PHASE	PWMTMR1 Phase Status. The PWM_STAT.TMR1PHASE bit indicates the current phase for the PWMTMR1 waveform.	
		0	1st Half Phase
		1	2nd Half Phase
24 (R/W1C)	TMR0PHASE	PWMTMR0 Phase Status. The PWM_STAT.TMR0PHASE bit indicates the current phase for the PWMTMR0 waveform.	
		0	1st Half Phase
		1	2nd Half Phase
20 (R/W1C)	TMR4PER	PWMTMR4 Period Boundary Status. The PWM_STAT.TMR4PER bit indicates whether or not the PWMTMR4 period boundary has been reached.	
		0	PWMTMR4 period boundary not reached
		1	PWMTMR4 period boundary reached
19 (R/W1C)	TMR3PER	PWMTMR3 Period Boundary Status. The PWM_STAT.TMR3PER bit indicates whether or not the PWMTMR3 period boundary has been reached.	
		0	PWMTMR3 period boundary not reached
		1	PWMTMR3 period boundary reached

Table 18-9: PWM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W1C)	TMR2PER	PWMTMR2 Period Boundary Status. The PWM_STAT.TMR2PER bit indicates whether or not the PWMTMR2 period boundary has been reached.	
		0	PWMTMR2 period boundary not reached
		1	PWMTMR2 period boundary reached
17 (R/W1C)	TMR1PER	PWMTMR1 Period Boundary Status. The PWM_STAT.TMR1PER bit indicates whether or not the PWMTMR1 period boundary has been reached.	
		0	PWMTMR1 period boundary not reached
		1	PWMTMR1 period boundary reached
16 (R/W1C)	TMR0PER	PWMTMR0 Period Boundary Status. The PWM_STAT.TMR0PER bit indicates whether or not the PWMTMR0 period boundary has been reached.	
		0	PWMTMR0 period boundary not reached
		1	PWMTMR0 period boundary reached
11 (R/NW)	SRTRIPD	Self-Restart Trip Status for Channel D. The PWM_STAT.SRTRIPD bit indicates whether the PWM Channel D self-restart has been tripped. For more information, see the PWM_TRIPCFG.MODEOA bit description.	
		0	Channel D Self-Restart Trip Status is "not tripped"
		1	Channel D Self-Restart Trip Status is "tripped"
10 (R/W1C)	FLTTRIPD	Fault Trip Status for Channel D. The PWM_STAT.FLTTRIPD bit indicates whether the PWM Channel D fault has been tripped. For more information, see the PWM_TRIPCFG.MODEOA bit description.	
		0	Channel D Fault Trip Status is "not tripped"
		1	Channel D Fault Trip Status is "tripped"

Table 18-9: PWM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/NW)	SRTRIPC	Self-Restart Trip Status for Channel C. The PWM_STAT.SRTRIPC bit indicates whether the PWM Channel C self-restart has been tripped. For more information, see the PWM_TRIPCFG.MODEOA bit description.	
		0	Channel C Self-Restart Trip Status is "not tripped"
		1	Channel C Self-Restart Trip Status is "tripped"
8 (R/W1C)	FLTTRIPC	Fault Trip Status for Channel C. The PWM_STAT.FLTTRIPC bit indicates whether the PWM Channel C fault has been tripped. For more information, see the PWM_TRIPCFG.MODEOA bit description.	
		0	Channel C Fault Trip Status is "not tripped"
		1	Channel C Fault Trip Status is "tripped"
7 (R/NW)	SRTRIPB	Self-Restart Trip Status for Channel B. The PWM_STAT.SRTRIPB bit indicates whether the PWM Channel B self-restart has been tripped. For more information, see the PWM_TRIPCFG.MODEOA bit description.	
		0	Channel B Self-Restart Trip Status is "not tripped"
		1	Channel B Self-Restart Trip Status is "tripped"
6 (R/W1C)	FLTTRIPB	Fault Trip Status for Channel B. The PWM_STAT.FLTTRIPB bit indicates whether the PWM Channel B fault has been tripped. For more information, see the PWM_TRIPCFG.MODEOA bit description.	
		0	Channel B Fault Trip Status is "not tripped"
		1	Channel A Fault Trip Status is "tripped"
5 (R/NW)	SRTRIPA	Self-Restart Trip Status for Channel A. The PWM_STAT.SRTRIPA bit indicates whether the PWM Channel A self-restart has been tripped. For more information, see the PWM_TRIPCFG.MODEOA bit description.	
		0	Channel A Self-Restart Trip Status is "not tripped"
		1	Channel A Self-Restart Trip Status is "tripped"

Table 18-9: PWM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W1C)	FLTTRIPA	Fault Trip Status for Channel A. The PWM_STAT.FLTTRIPA bit indicates whether the PWM Channel A fault has been tripped. For more information, see the PWM_TRIPCFG.MODEOA bit description.	
		0	Channel A Fault Trip Status is "not tripped"
		1	Channel A Fault Trip Status is "tripped"
3 (R/NW)	RAWTRIP1	Raw Trip 1 Status. The PWM_STAT.RAWTRIP1 bit indicates the raw input level for the PWM TRIP1 input.	
		0	TRIP1 Level is Low
		1	TRIP1 Level is High
2 (R/NW)	RAWTRIP0	Raw Trip 0 Status. The PWM_STAT.RAWTRIP0 bit indicates the raw input level for the PWM TRIP0 input.	
		0	TRIP0 Level is Low
		1	TRIP0 Level is High
1 (R/W1C)	TRIP1	Status bit set when TRIP1 is active low. The PWM_STAT.TRIP1 bit indicates whether the PWM TRIP1 fault has been tripped with an active-low input.	
		0	TRIP1 status is "not tripped"
		1	TRIP1 status is "tripped" (active low)
0 (R/W1C)	TRIP0	Status bit set when TRIP0 is active low. The PWM_STAT.TRIP0 bit indicates whether the PWM TRIP0 fault has been tripped with an active-low input.	
		0	TRIP0 status is "not tripped"
		1	TRIP0 status is "tripped" (active low)

Interrupt Mask Register

The PWM_IMSK register masks (disables) or unmasks (enables) PWM interrupts. When an unmasked interrupt occurs, the PWM latches the interrupt status in the PWM_ILAT register.

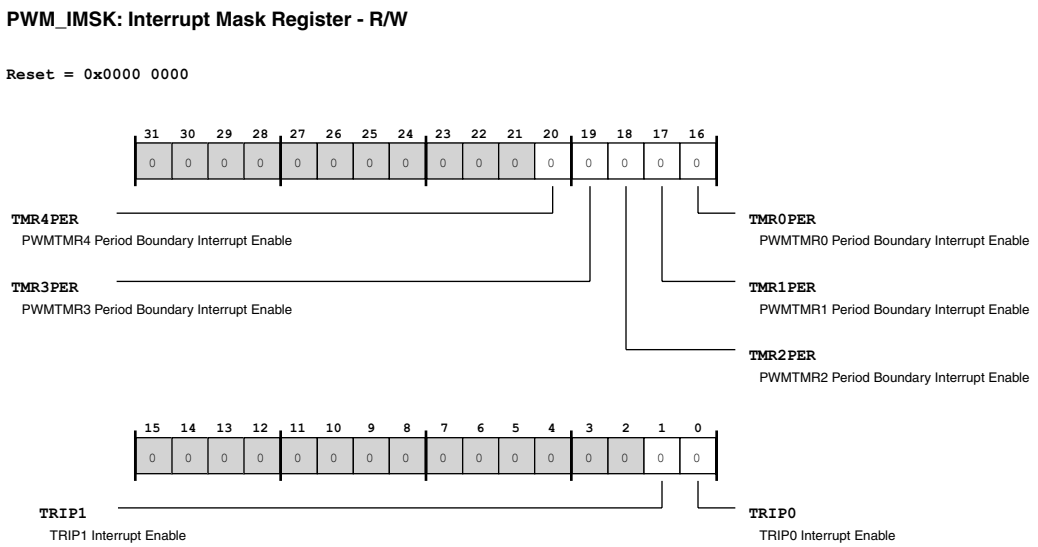


Figure 18-25: PWM_IMSK Register Diagram

Table 18-10: PWM_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W)	TMR4PER	PWMTMR4 Period Boundary Interrupt Enable. The PWM_IMSK.TMR4PER bit enables (unmasks) the PWMTMR4 period boundary interrupt. This condition occurs when the timers period boundary is reached (PWM_STAT.TMR4PER =1).	
		0	Mask PWMTMR4 Period Interrupt
		1	Unmask PWMTMR4 Period Interrupt
19 (R/W)	TMR3PER	PWMTMR3 Period Boundary Interrupt Enable. The PWM_IMSK.TMR3PER bit enables (unmasks) the PWMTMR3 period boundary interrupt. This condition occurs when the timers period boundary is reached (PWM_STAT.TMR3PER =1).	
		0	Mask PWMTMR3 Period Interrupt
		1	Unmask PWMTMR3 Period Interrupt
18 (R/W)	TMR2PER	PWMTMR2 Period Boundary Interrupt Enable. The PWM_IMSK.TMR2PER bit enables (unmasks) the PWMTMR2 period boundary interrupt. This condition occurs when the timers period boundary is reached (PWM_STAT.TMR2PER =1).	
		0	Mask PWMTMR2 Period Interrupt
		1	Unmask PWMTMR2 Period Interrupt

Table 18-10: PWM_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	TMR1PER	PWMTMR1 Period Boundary Interrupt Enable. The PWM_IMSK.TMR1PER bit enables (unmasks) the PWMTMR1 period boundary interrupt. This condition occurs when the timers period boundary is reached (PWM_STAT.TMR1PER =1).	
		0	Mask PWMTMR1 Period Interrupt
		1	Unmask PWMTMR1 Period Interrupt
16 (R/W)	TMR0PER	PWMTMR0 Period Boundary Interrupt Enable. The PWM_IMSK.TMR0PER bit enables (unmasks) the PWMTMR0 period boundary interrupt. This condition occurs when the timers period boundary is reached (PWM_STAT.TMR0PER =1).	
		0	Mask PWMTMR0 Period Interrupt
		1	Unmask PWMTMR0 Period Interrupt
1 (R/W)	TRIP1	TRIP1 Interrupt Enable. The PWM_IMSK.TRIP1 bit enables (unmasks) the TRIP1 interrupt. This condition occurs when fault input is tripped (PWM_STAT.TRIP1 =1).	
		0	Mask TRIP1 Interrupt
		1	Unmask TRIP1 Interrupt
0 (R/W)	TRIP0	TRIP0 Interrupt Enable. The PWM_IMSK.TRIP0 bit enables (unmasks) the TRIP0 interrupt. This condition occurs when fault input is tripped (PWM_STAT.TRIP0 =1).	
		0	Mask TRIP0 Interrupt
		1	Unmask TRIP0 Interrupt

Interrupt Latch Register

The PWM_ILAT register latches the occurrence of unmasked (enabled) PWM interrupts. These interrupts are unmasked or masked with the PWM_IMSK register.

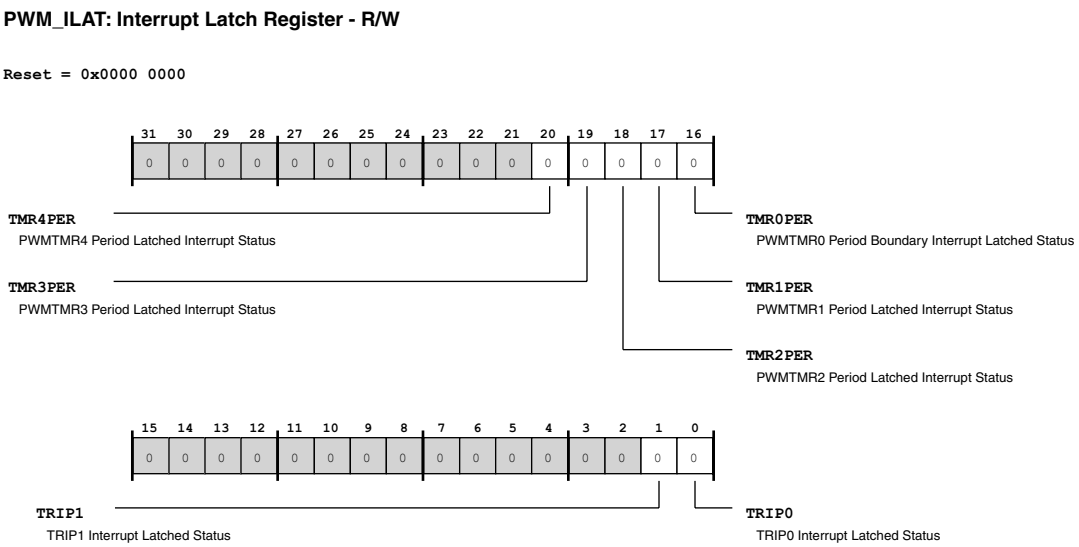


Figure 18-26: PWM_ILAT Register Diagram

Table 18-11: PWM_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W1C)	TMR4PER	PWMTMR4 Period Latched Interrupt Status. The PWM_ILAT.TMR4PER bit indicates the latched status of the PWMTMR4 period boundary interrupt.	
		0	No Interrupt Latched
		1	Interrupt Latched
19 (R/W1C)	TMR3PER	PWMTMR3 Period Latched Interrupt Status. The PWM_ILAT.TMR3PER bit indicates the latched status of the PWMTMR3 period boundary interrupt.	
		0	No Interrupt Latched
		1	Interrupt Latched
18 (R/W1C)	TMR2PER	PWMTMR2 Period Latched Interrupt Status. The PWM_ILAT.TMR2PER bit indicates the latched status of the PWMTMR2 period boundary interrupt.	
		0	No Interrupt Latched
		1	Interrupt Latched

Table 18-11: PWM_ILAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W1C)	TMR1PER	PWMTMR1 Period Latched Interrupt Status. The PWM_ILAT.TMR1PER bit indicates the latched status of the PWMTMR1 period boundary interrupt.	
		0	No Interrupt Latched
		1	Interrupt Latched
16 (R/W1C)	TMR0PER	PWMTMR0 Period Boundary Interrupt Latched Status. The PWM_ILAT.TMR0PER bit indicates the latched status of the PWMTMR0 period boundary interrupt.	
		0	No Interrupt Latched
		1	Interrupt Latched
1 (R/W1C)	TRIP1	TRIP1 Interrupt Latched Status. The PWM_ILAT.TRIP1 bit indicates the latched status of the TRIP1 interrupt.	
		0	No Interrupt Latched
		1	Interrupt Latched
0 (R/W1C)	TRIP0	TRIP0 Interrupt Latched Status. The PWM_ILAT.TRIP0 bit indicates the latched status of the TRIP0 interrupt.	
		0	No Interrupt Latched
		1	Interrupt Latched

Chop Configuration Register

The PWM_CHOPCFG register holds a divisor value that controls the chopping frequency. The PWM permits a mixing of the output signals with a high-frequency chopping signal to aid with interfacing to pulse transformers. Also note that high-frequency chopping may be independently enabled for each channel's high-side and the low-side outputs using channel control bits. (For example, control chopping for Channel A with the PWM_CHANCFG.ENCHOPAH and PWM_CHANCFG.ENCHOPAH bits.)

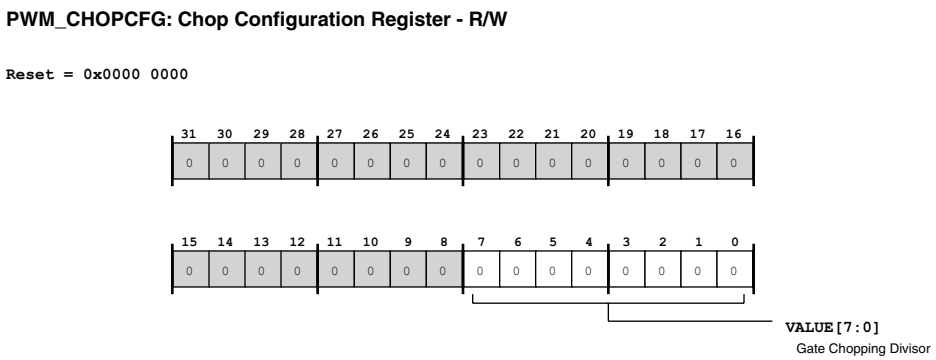


Figure 18-27: PWM_CHOPCFG Register Diagram

Table 18-12: PWM_CHOPCFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Gate Chopping Divisor. The PWM_CHOPCFG.VALUE bits provide the high frequency chopping divisor. When the divisor value is changed, the new period takes effect from the next edge of the chopping signal. The PWM_CHOPCFG.VALUE value may be calculated using either of the following formulas: CHOPDIV = $[(T_{CHOP}/T_{CK}) / 4] - 1$ CHOPDIV = $[(f_{CK} / f_{CHOP}) / 4] - 1$

Dead Time Register

The PWM_DT register controls the dead time, which the PWM inserts into the pairs of output signals. Note that each channel has its own version of a double buffered dead time register, the double buffering of which depends on the period boundary of the PWMTMRx that the channel could be currently using. The dead time, T_d, is related to the value in the PWM_DT register by:

T_d = PWM_DT x 2 x t_{CK}

Note that the PWM holds the buffered PWM_DT value for a channel at 0 if the channel's low side mode is independent (for example, PWM_CHANCFG.MODELSA =1). Also, note that the PWM_DT value must be less than half the respective timer period (for example, PWM_TM0/2). For more information about applying dead time to PWM output pairs, see the PWM Functional Description section.

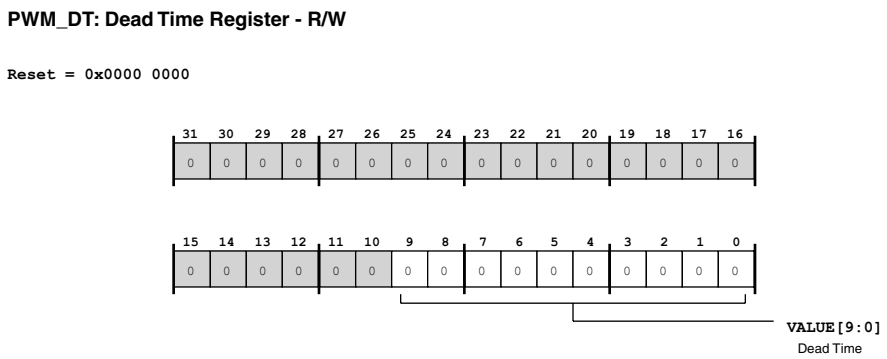


Figure 18-28: PWM_DT Register Diagram

Table 18-13: PWM_DT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	VALUE	Dead Time. The PWM_DT.VALUE bits select the dead time that the PWM adds to the timing of the output pairs.

Sync Pulse Width Register

The PWM_SYNC_WID register selects the pulse width for the external sync pulse available on the PWM_SYNC pin. The relation between the PWM_SYNC_WID register value and the pulse width (T_{PWM_SYNC}) is give by the formula:

PWM_SYNC_WID = (T_{PWM_SYNC} / t_{CK}) -1

For more information about applying the sync pulse width, see the PWM Functional Description section. Note that if the pulse width is changed in between sync pulses, the PWM applies the changed width on the next internal sync pulse. If, while the sync pulse is active, the chosen timer reaches its period boundary, the changed pulse width takes effect on that period boundary.

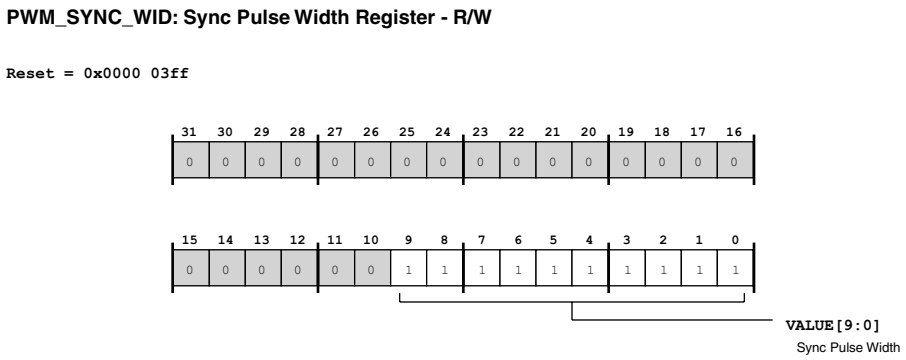


Figure 18-29: PWM_SYNC_WID Register Diagram

Table 18-14: PWM_SYNC_WID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	VALUE	Sync Pulse Width. The PWM_SYNC_WID.VALUE bits select the pulse width for the external sync pulse available on the PWM_SYNC pin.

Timer 0 Period Register

The PWM_TM0 register controls the switch period (T_{SP} of the PWMTMR0 timer. The PWM_TM0 value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM0= (T_{SP}) / 2 x t_{CK}

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that PWM_TM0 values of 0 and 1 are not defined and must not be used when the PWM is enabled.

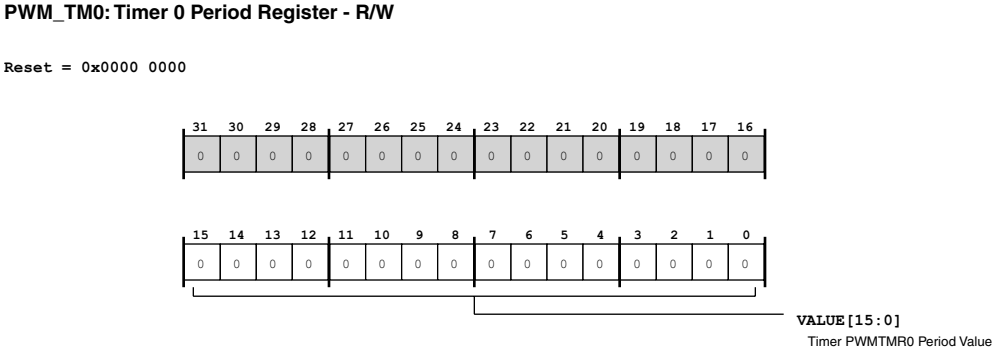


Figure 18-30: PWM_TM0 Register Diagram

Table 18-15: PWM_TM0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR0 Period Value. The PWM_TM0.VALUE bits select the period for the PWMTMR0 timer.

Timer 1 Period Register

The PWM_TM1 register controls the switch period (T_{SP} of the PWMTMR1 timer. The PWM_TM1 value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

$$PWM\_TM1 = (T_{SP}) / 2 \times t_{CK}$$

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that PWM_TM1 values of 0 and 1 are not defined and must not be used when the PWM is enabled.

PWM_TM1: Timer 1 Period Register - R/W

Reset = 0x0000 0000

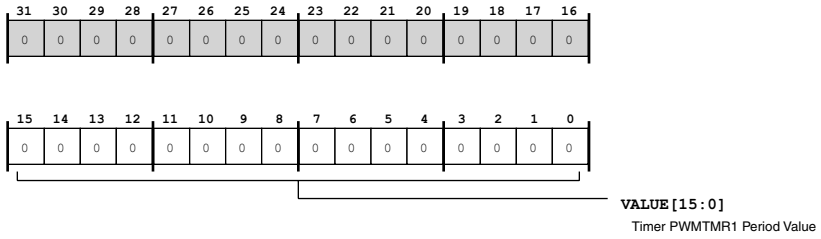


Figure 18-31: PWM_TM1 Register Diagram

Table 18-16: PWM_TM1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR1 Period Value. The PWM_TM1.VALUE bits select the period for the PWMTMR1 timer.

Timer 2 Period Register

The PWM_TM2 register controls the switch period (T_{SP} of the PWMTMR2 timer. The PWM_TM2 value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

$$PWM\_TM1 = (T_{SP}) / 2 \times t_{CK}$$

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that PWM_TM2 values of 0 and 1 are not defined and must not be used when the PWM is enabled.

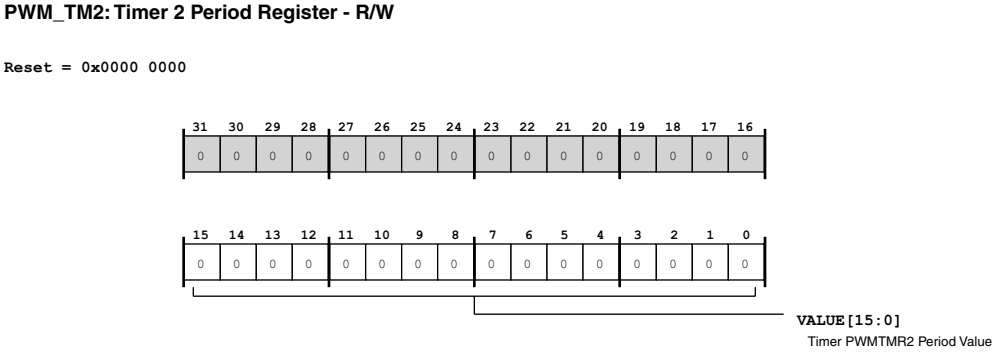


Figure 18-32: PWM_TM2 Register Diagram

Table 18-17: PWM_TM2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR2 Period Value. The PWM_TM2.VALUE bits select the period for the PWMTMR2 timer.

Timer 3 Period Register

The PWM_TM3 register controls the switch period (T_{SP} of the PWMTMR3 timer. The PWM_TM3 value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM3= (T_{SP}) / 2 x t_{CK}

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that PWM_TM3 values of 0 and 1 are not defined and must not be used when the PWM is enabled.

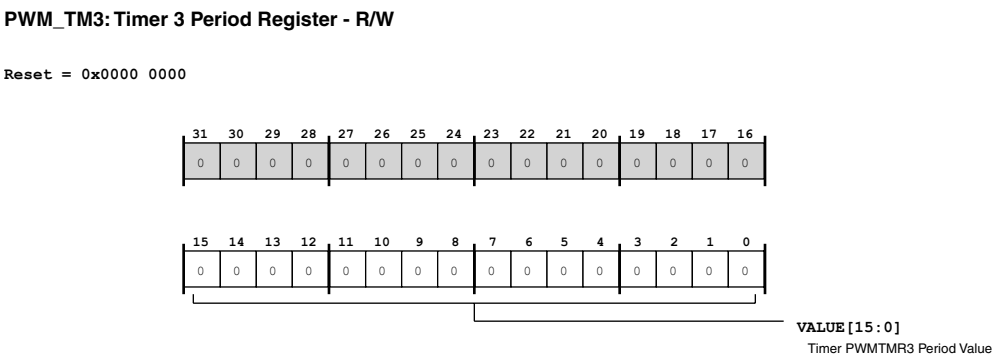


Figure 18-33: PWM_TM3 Register Diagram

Table 18-18: PWM_TM3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR3 Period Value. The PWM_TM3.VALUE bits select the period for the PWMTMR3 timer.

Timer 4 Period Register

The PWM_TM4 register controls the switch period (T_{SP} of the PWMTMR4 timer. The PWM_TM4 value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM4= (T_{SP}) / 2 x t_{CK}

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that PWM_TM4 values of 0 and 1 are not defined and must not be used when the PWM is enabled.

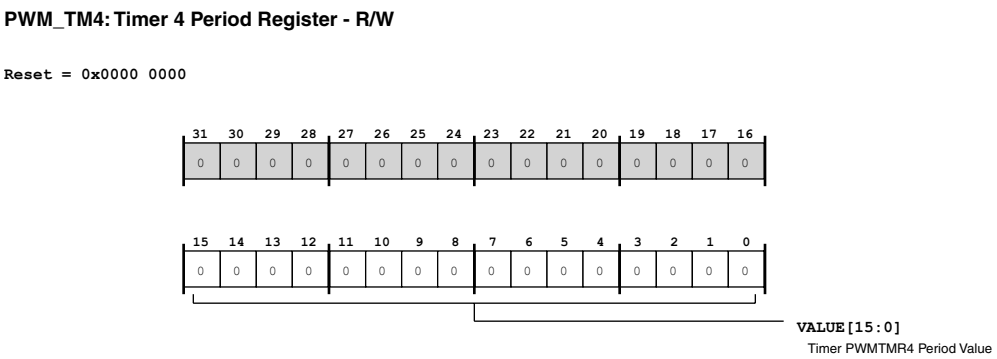


Figure 18-34: PWM_TM4 Register Diagram

Table 18-19: PWM_TM4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR4 Period Value. The PWM_TM4.VALUE bits select the period for the PWMTMR4 timer.

Channel A Delay Register

The PWM_DLYA register controls a delay for the Channel A timer (only PWMTMR1, PWMTMR2, PWMTMR3 or PWMTMR4) with reference to the main timer (PWMTMR0). To use apply this delay, the delay must be enabled (PWM_CTL.DLYAEN =1}. For more information about applying the delay, see the PWM Functional Description section. Note that the PWM_DLYA delay value must be less that less that twice the period value of the timer being used for the channel (for example, if PWMTMR1 is used, PWM_DLYA must be less than 2xPWMTMR1). Also, note that the period of the main timer must be an integer multiple of the timer being used for the channel (for example, if PWMTMR1 is used, PWMTMR0 = NxPW-MTMR1, where N is an integer).

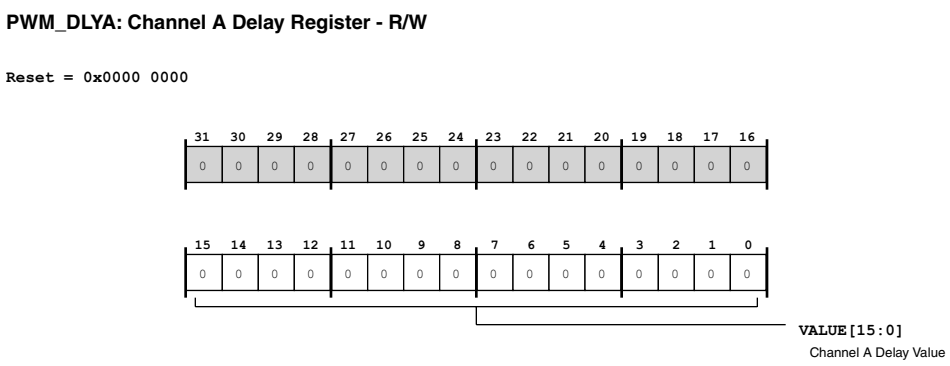


Figure 18-35: PWM_DLYA Register Diagram

Table 18-20: PWM_DLYA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Channel A Delay Value. The PWM_DLYA.VALUE bits select the phase delay between the main timer (PWMTMR0) and the timer used for Channel A.

Channel B Delay Register

The PWM_DLYB register controls a delay for the Channel B timer (only PWMTMR1, PWMTMR2, PWMTMR3 or PWMTMR4) with reference to the main timer (PWMTMR0). To use apply this delay, the delay must be enabled (PWM_CTL.DLYBEN =1}. For more information about applying the delay, see the PWM Functional Description section. Note that the PWM_DLYB delay value must be less that less that twice the period value of the timer being used for the channel (for example, if PWMTMR1 is used, PWM_DLYB must be less than 2xPWMTMR1). Also, note that the period of the main timer must be an integer multiple of the timer being used for the channel (for example, if PWMTMR1 is used, PWMTMR0 = NxPW-MTMR1, where N is an integer).

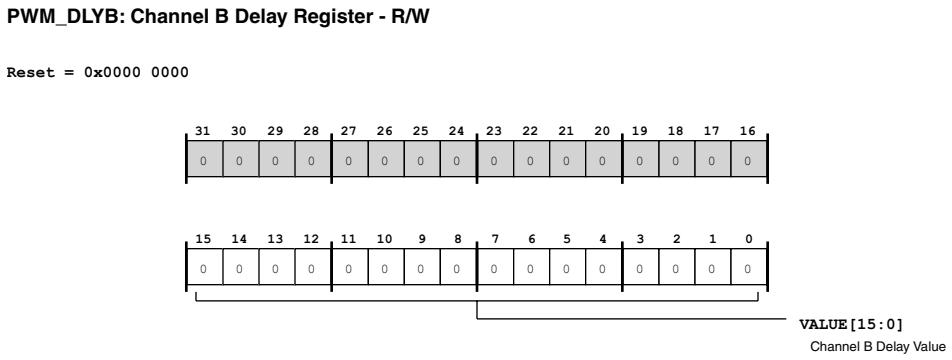


Figure 18-36: PWM_DLYB Register Diagram

Table 18-21: PWM_DLYB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Channel B Delay Value. The PWM_DLYB.VALUE bits select the phase delay between the main timer (PWMTMR0) and the timer used for Channel B.

Channel C Delay Register

The PWM_DLYC register controls a delay for the Channel C timer (only PWMTMR1, PWMTMR2, PWMTMR3 or PWMTMR4) with reference to the main timer (PWMTMR0). To use apply this delay, the

delay must be enabled (PWM_CTL.DLYCEN =1}. For more information about applying the delay, see the PWM Functional Description section. Note that the PWM_DLYC delay value must be less that less that twice the period value of the timer being used for the channel (for example, if PWMTMR1 is used, PWM_DLYC must be less than 2xPWMTMR1). Also, note that the period of the main timer must be an integer multiple of the timer being used for the channel (for example, if PWMTMR1 is used, PWMTMR0 = NxPW-MTMR1, where N is an integer).

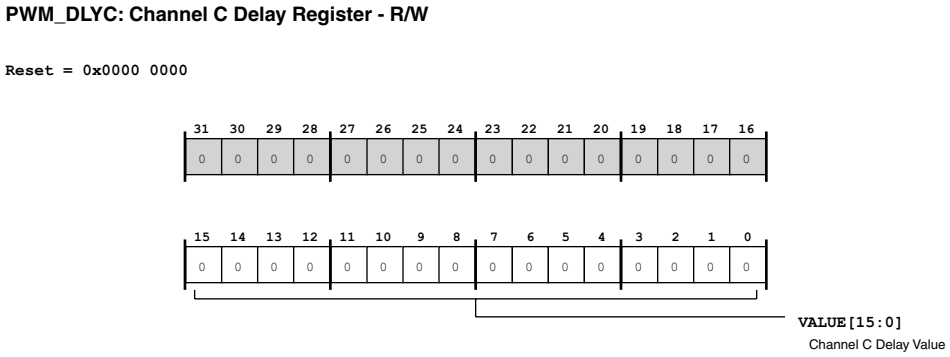


Figure 18-37: PWM_DLYC Register Diagram

Table 18-22: PWM_DLYC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Channel C Delay Value. The PWM_DLYC.VALUE bits select the phase delay between the main timer (PWMTMR0) and the timer used for Channel C.

Channel D Delay Register

The PWM_DLYD register controls a delay for the Channel D timer (only PWMTMR1, PWMTMR2, PWMTMR3 or PWMTMR4) with reference to the main timer (PWMTMR0). To use apply this delay, the delay must be enabled (PWM_CTL.DLYDEN =1}. For more information about applying the delay, see the PWM Functional Description section. Note that the PWM_DLYD delay value must be less that less that twice the period value of the timer being used for the channel (for example, if PWMTMR1 is used, PWM_DLYD must be less than 2xPWMTMR1). Also, note that the period of the main timer must be an integer multiple of the timer being used for the channel (for example, if PWMTMR1 is used, PWMTMR0 = NxPW-MTMR1, where N is an integer).

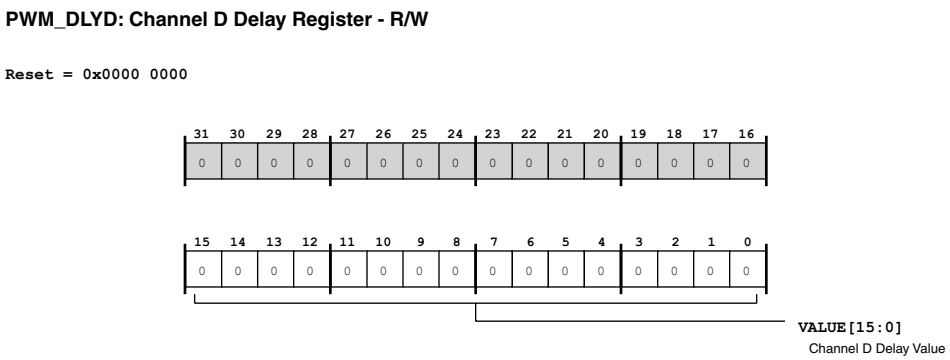


Figure 18-38: PWM_DLYD Register Diagram

Table 18-23: PWM_DLYD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Channel D Delay Value. The PWM_DLYD.VALUE bits select the phase delay between the main timer (PWMTMR0) and the timer used for Channel D.

Channel A Control Register

The PWM_ACTL register selects the low and high side output pulse mode, enables low and high side output, and enables low/high side output crossover.

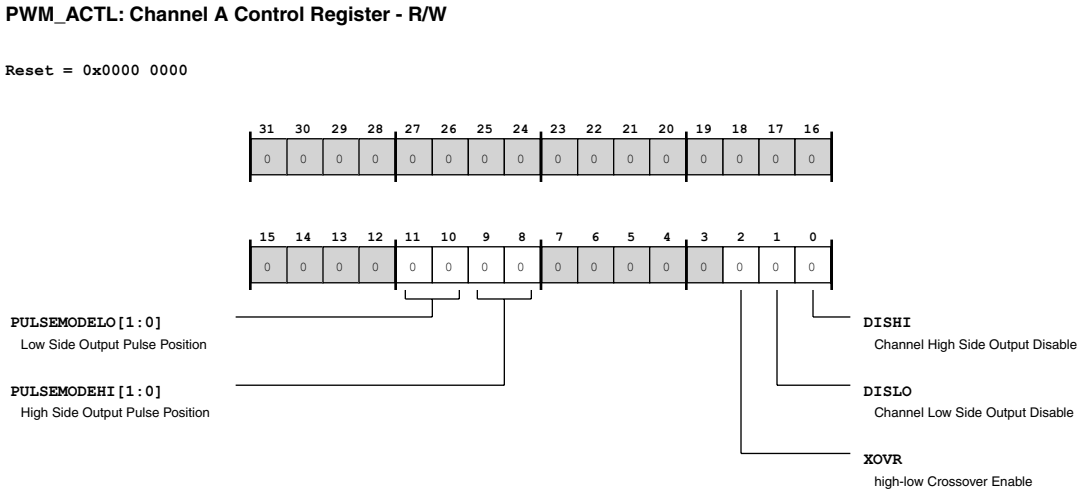


Figure 18-39: PWM_ACTL Register Diagram

Table 18-24: PWM_ACTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	PULSEMODELO	Low Side Output Pulse Position. The PWM_ACTL.PULSEMODELO bits select the pulse position for Channel A low side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the centre of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_ALO register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the centre of the PWM period. This mode uses both the duty-cycle registers (PWM_ALO and PWM_AL1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty-cycle registers (PWM_ALO and PWM_AL1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half
9:8 (R/W)	PULSEMODEHI	High Side Output Pulse Position. The PWM_ACTL.PULSEMODEHI bits select the pulse position for Channel A high side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the centre of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_AHO register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the centre of the PWM period. This mode uses both the duty-cycle registers (PWM_AHO and PWM_AH1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty-cycle registers (PWM_AHO and PWM_AH1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half

Table 18-24: PWM_ACTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	XOVR	high-low Crossover Enable. The PWM_ACTL.XOVR bit enables crossover between the channels high and low side outputs. When enabled, this bit directs the PWM to send the low-side output through the high-side output pin and the high-side output through the low side output pin.	
		0	Disable Crossover
		1	Enable Crossover
1 (R/W)	DISLO	Channel Low Side Output Disable. The PWM_ACTL.DISLO bit enables the channels low side output.	
		0	Disable Low Side Output
		1	Enable Low Side Output
0 (R/W)	DISHI	Channel High Side Output Disable. The PWM_ACTL.DISHI bit enables the channels high side output.	
		0	Disable High Side Output
		1	Enable High Side Output

Channel A-High Duty-0 Register

The PWM_AH0 and PWM_AH1 registers determine the width for the high side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the high side output pulses for the Channel A duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_ACTL.PULSEMODEHI bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_AH0 register to determine the assertion and de-assertion count for the high side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts Channel A high pulse output for count less than PWM_AH0 and de-asserts this output for count greater than PWM_AH1.

The value range for the PWM_AH0 and PWM_AH1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_AH0 and PWM_AH1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_DT (two's complement) to +PWM_TM0/2 + PWM_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_DT (two's complement) to -PWM_TM0/2 - PWM_DT.

Note that using values in the PWM_AH0 or PWM_AH1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

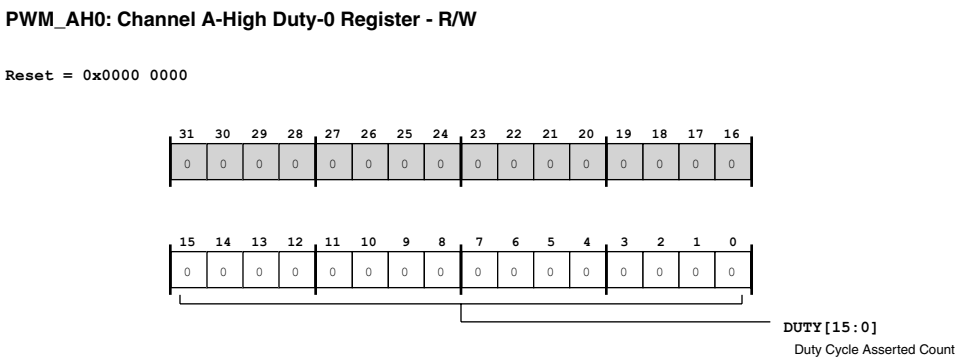


Figure 18-40: PWM_AH0 Register Diagram

Table 18-25: PWM_AH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_AH0.DUTY bits select the duty cycle asserted count for Channel A high side output.

Channel A-High Duty-1 Register

The PWM_AH0 and PWM_AH1 registers determine the width for the high side output pulses. For more information, see the PWM_AH0 register description.

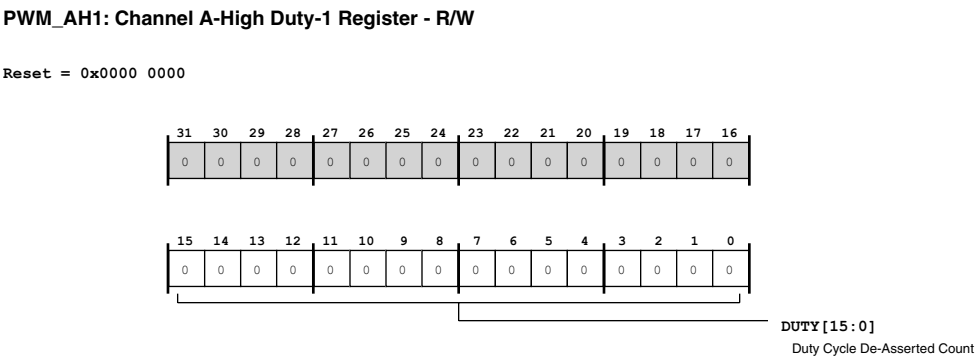


Figure 18-41: PWM_AH1 Register Diagram

Table 18-26: PWM_AH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_AH1.DUTY bits select the duty cycle de-asserted count for Channel A high side output.

Channel A-Low Duty-0 Register

The PWM_AL0 and PWM_AL1 registers determine the width for the low side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the low side output pulses for the Channel A duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_ACTL.PULSEMODEL0 bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_AL0 register to determine the assertion and de-assertion count for the low side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts Channel A low pulse output for count less than PWM_AL0 and de-asserts this output for count greater than PWM_AL1.

The value range for the PWM_AL0 and PWM_AL1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_AL0 and PWM_AL1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_DT (two's complement) to +PWM_TM0/2 + PWM_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_DT (two's complement) to -PWM_TM0/2 - PWM_DT.

Note that using values in the PWM_AL0 or PWM_AL1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

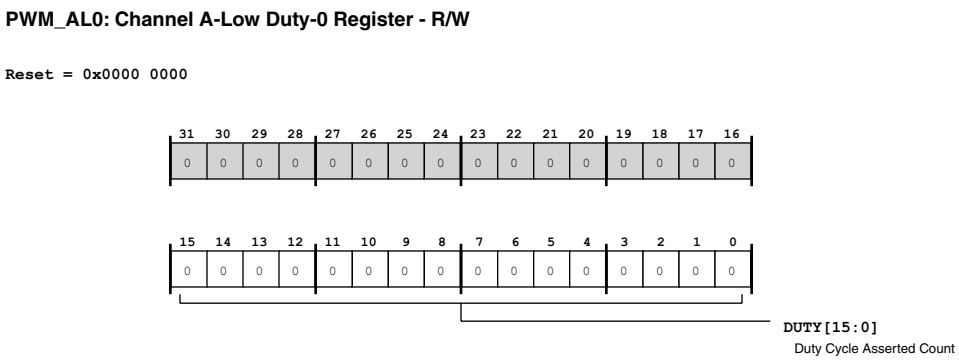


Figure 18-42: PWM_AL0 Register Diagram

Table 18-27: PWM_AL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_AL0.DUTY bits select the duty cycle asserted count for Channel A low side output.

Channel A-Low Duty-1 Register

The PWM_AL0 and PWM_AL1 registers determine the width for the low side output pulses. For more information, see the PWM_AL0 register description.

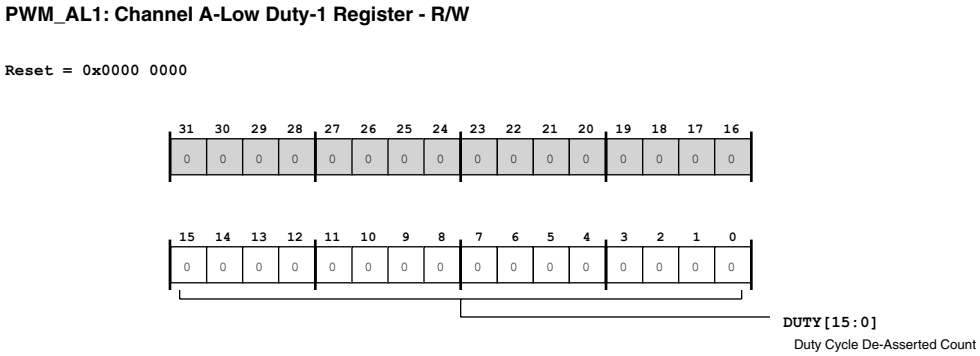


Figure 18-43: PWM_AL1 Register Diagram

Table 18-28: PWM_AL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_AL1.DUTY bits select the duty cycle de-asserted count for Channel A low side output.

Channel B Control Register

The PWM_BCTL register selects the low and high side output pulse mode, enables low and high side output, and enables low/high side output crossover.

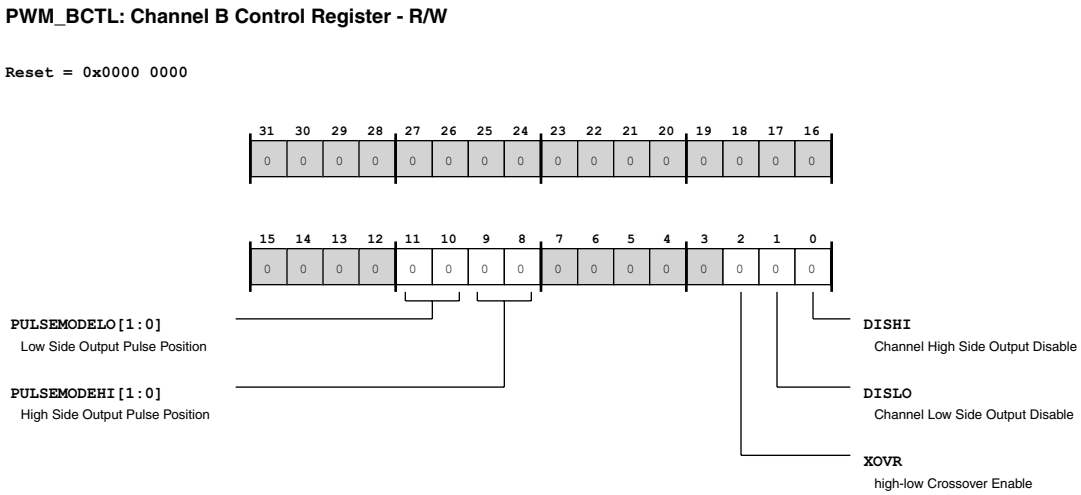


Figure 18-44: PWM_BCTL Register Diagram

Table 18-29: PWM_BCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	PULSEMODELO	Low Side Output Pulse Position. The PWM_BCTL.PULSEMODELO bits select the pulse position for Channel B low side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the centre of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_BLO register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the centre of the PWM period. This mode uses both the duty-cycle registers (PWM_BLO and PWM_BL1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty-cycle registers (PWM_BLO and PWM_BL1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half
9:8 (R/W)	PULSEMODEHI	High Side Output Pulse Position. The PWM_BCTL.PULSEMODEHI bits select the pulse position for Channel B high side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the centre of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_BHO register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the centre of the PWM period. This mode uses both the duty-cycle registers (PWM_BHO and PWM_BH1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty-cycle registers (PWM_BHO and PWM_BH1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half

Table 18-29: PWM_BCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	XOVR	high-low Crossover Enable. The PWM_BCTL.XOVR bit enables crossover between the channels high and low side outputs. When enabled, this bit directs the PWM to send the low-side output through the high-side output pin and the high-side output through the low side output pin.	
		0	Disable Crossover
		1	Enable Crossover
1 (R/W)	DISLO	Channel Low Side Output Disable. The PWM_BCTL.DISLO bit enables the channels low side output.	
		0	Disable Low Side Output
		1	Enable Low Side Output
0 (R/W)	DISHI	Channel High Side Output Disable. The PWM_BCTL.DISHI bit enables the channels high side output.	
		0	Disable High Side Output
		1	Enable High Side Output

Channel B-High Duty-0 Register

The PWM_BH0 and PWM_BH1 registers determine the width for the high side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the high side output pulses for the Channel B duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_BCTL.PULSEMODEHI bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_BH0 register to determine the assertion and de-assertion count for the high side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts Channel B high pulse output for count less than PWM_BH0 and de-asserts this output for count greater than PWM_BH1.

The value range for the PWM_BH0 and PWM_BH1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_BH0 and PWM_BH1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_DT (two's complement) to +PWM_TM0/2 + PWM_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_DT (two's complement) to -PWM_TM0/2 - PWM_DT.

Note that using values in the PWM_BH0 or PWM_BH1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

PWM_BH0: Channel B-High Duty-0 Register - R/W

Reset = 0x0000 0000

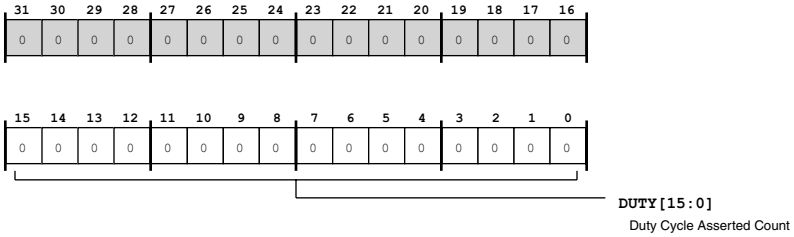


Figure 18-45: PWM_BH0 Register Diagram

Table 18-30: PWM_BH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count.

Channel B-High Duty-1 Register

The PWM_BH0 and PWM_BH1 registers determine the width for the high side output pulses. For more information, see the PWM_BH0 register description.

PWM_BH1: Channel B-High Duty-1 Register - R/W

Reset = 0x0000 0000

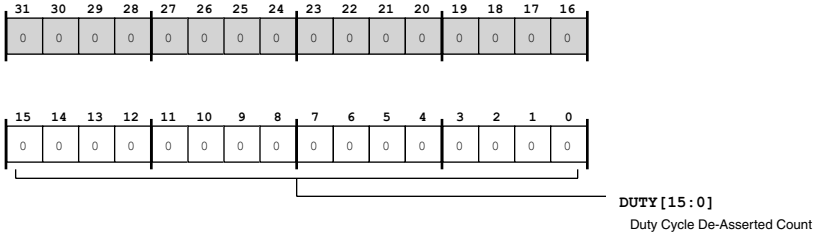


Figure 18-46: PWM_BH1 Register Diagram

Table 18-31: PWM_BH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count.

Channel B-Low Duty-0 Register

The PWM_BL0 and PWM_BL1 registers determine the width for the low side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the low side output pulses for the Channel B duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_BCTL.PULSEMODEL0 bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_BL0 register to determine the assertion and de-assertion count for the low side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts Channel B low pulse output for count less than PWM_BL0 and de-asserts this output for count greater than PWM_BL1.

The value range for the PWM_BL0 and PWM_BL1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_BL0 and PWM_BL1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_DT (two's complement) to +PWM_TM0/2 + PWM_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_DT (two's complement) to -PWM_TM0/2 - PWM_DT.

Note that using values in the PWM_BL0 or PWM_BL1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

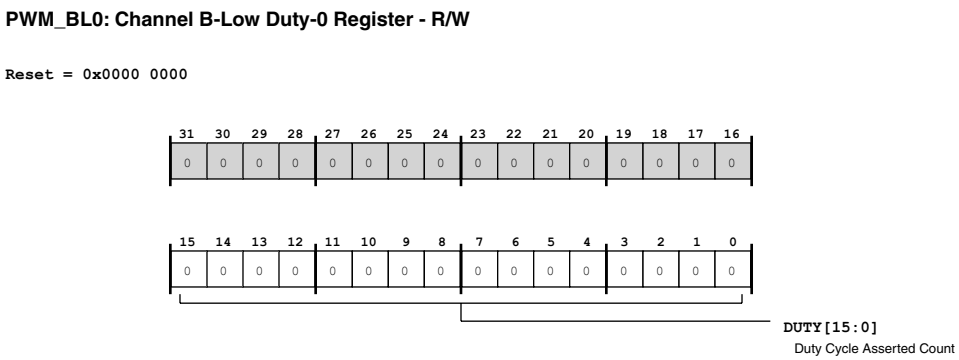


Figure 18-47: PWM_BL0 Register Diagram

Table 18-32: PWM_BL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_BL0.DUTY bits select the duty cycle asserted count for Channel B low side output.

Channel B-Low Duty-1 Register

The PWM_BL0 and PWM_BL1 registers determine the width for the low side output pulses. For more information, see the PWM_BL0 register description.

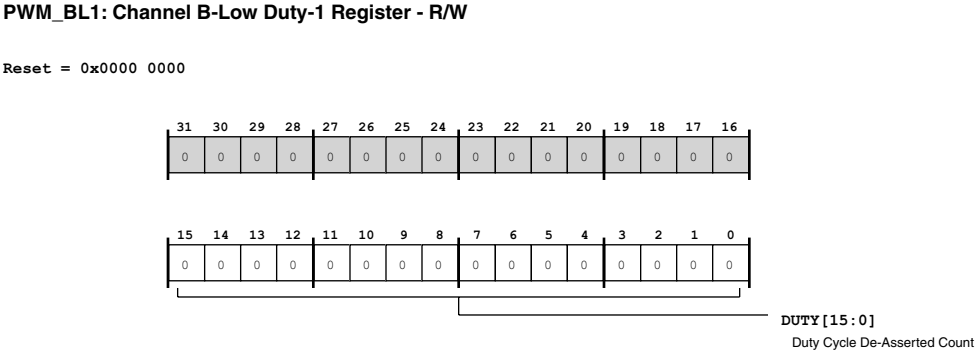


Figure 18-48: PWM_BL1 Register Diagram

Table 18-33: PWM_BL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_BL1.DUTY bits select the duty cycle de-asserted count for Channel B low side output.

Channel C Control Register

The PWM_CCTL register selects the low and high side output pulse mode, enables low and high side output, and enables low/high side output crossover.

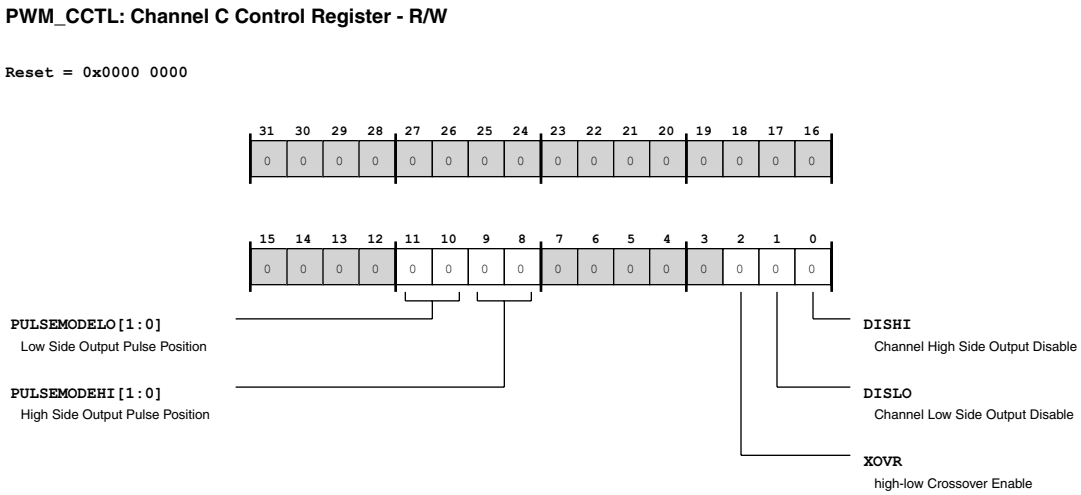


Figure 18-49: PWM_CCTL Register Diagram

Table 18-34: PWM_CCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	PULSEMODELO	Low Side Output Pulse Position. The PWM_CCTL.PULSEMODELO bits select the pulse position for Channel C low side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the centre of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_CLO register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the centre of the PWM period. This mode uses both the duty-cycle registers (PWM_CLO and PWM_CL1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty-cycle registers (PWM_CLO and PWM_CL1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half
9:8 (R/W)	PULSEMODEHI	High Side Output Pulse Position. The PWM_CCTL.PULSEMODEHI bits select the pulse position for Channel C high side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the centre of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_CHO register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the centre of the PWM period. This mode uses both the duty-cycle registers (PWM_CHO and PWM_CH1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty-cycle registers (PWM_CHO and PWM_CH1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half

Table 18-34: PWM_CCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	XOVR	high-low Crossover Enable. The PWM_CCTL.XOVR bit enables crossover between the channels high and low side outputs. When enabled, this bit directs the PWM to send the low-side output through the high-side output pin and the high-side output through the low side output pin.	
		0	Disable Crossover
		1	Enable Crossover
1 (R/W)	DISLO	Channel Low Side Output Disable. The PWM_CCTL.DISLO bit enables the channels low side output.	
		0	Disable Low Side Output
		1	Enable Low Side Output
0 (R/W)	DISHI	Channel High Side Output Disable. The PWM_CCTL.DISHI bit enables the channels high side output.	
		0	Disable High Side Output
		1	Enable High Side Output

Channel C-High Pulse Duty Register 0

The PWM_CH0 and PWM_CH1 registers determine the width for the high side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the high side output pulses for the Channel C duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_CCTL.PULSEMODEHI bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_CH0 register to determine the assertion and de-assertion count for the high side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts Channel C high pulse output for count less than PWM_CH0 and de-asserts this output for count greater than PWM_CH1.

The value range for the PWM_CH0 and PWM_CH1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_CH0 and PWM_CH1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_DT (two's complement) to +PWM_TM0/2 + PWM_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_DT (two's complement) to -PWM_TM0/2 - PWM_DT.

Note that using values in the PWM_CH0 or PWM_CH1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

PWM_CH0: Channel C-High Pulse Duty Register 0 - R/W

Reset = 0x0000 0000

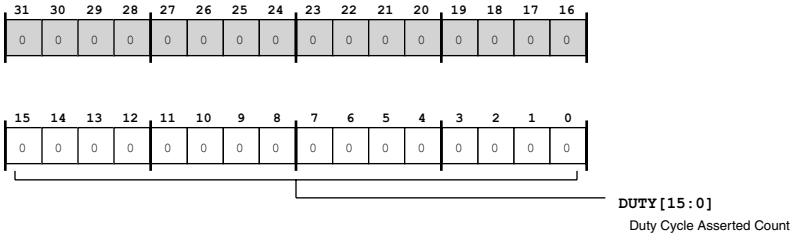


Figure 18-50: PWM_CH0 Register Diagram

Table 18-35: PWM_CH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_CH0.DUTY bits select the duty cycle asserted count for Channel C high side output.

Channel C-High Pulse Duty Register 1

The PWM_CH0 and PWM_CH1 registers determine the width for the high side output pulses. For more information, see the PWM_CH0 register description.

PWM_CH1: Channel C-High Pulse Duty Register 1 - R/W

Reset = 0x0000 0000

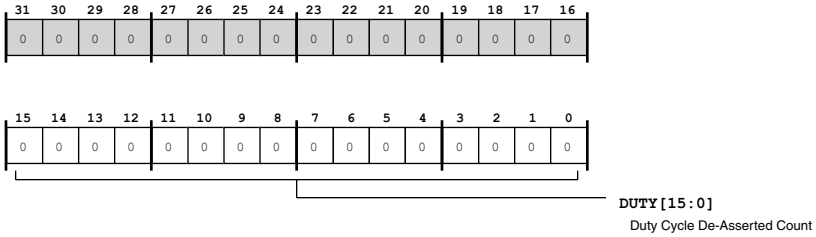


Figure 18-51: PWM_CH1 Register Diagram

Table 18-36: PWM_CH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_CH1.DUTY bits select the duty cycle de-asserted count for Channel C high side output.

Channel C-Low Pulse Duty Register 0

The PWM_CL0 and PWM_CL1 registers determine the width for the low side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the low side output pulses for the Channel C duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_CCTL.PULSEMODEL0 bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_CL0 register to determine the assertion and de-assertion count for the low side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts Channel C low pulse output for count less than PWM_CL0 and de-asserts this output for count greater than PWM_CL1.

The value range for the PWM_CL0 and PWM_CL1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_CL0 and PWM_CL1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_DT (two's complement) to +PWM_TM0/2 + PWM_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_DT (two's complement) to -PWM_TM0/2 - PWM_DT.

Note that using values in the PWM_CL0 or PWM_CL1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

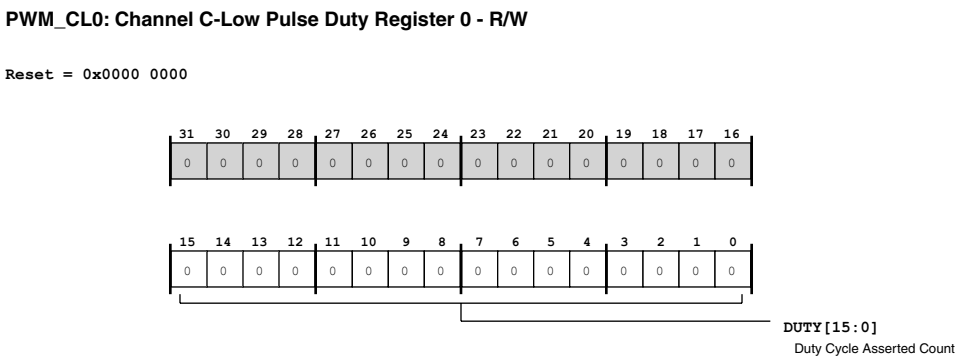


Figure 18-52: PWM_CL0 Register Diagram

Table 18-37: PWM_CL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_CLO.DUTY bits select the duty cycle asserted count for Channel C low side output.

Channel C-Low Duty-1 Register

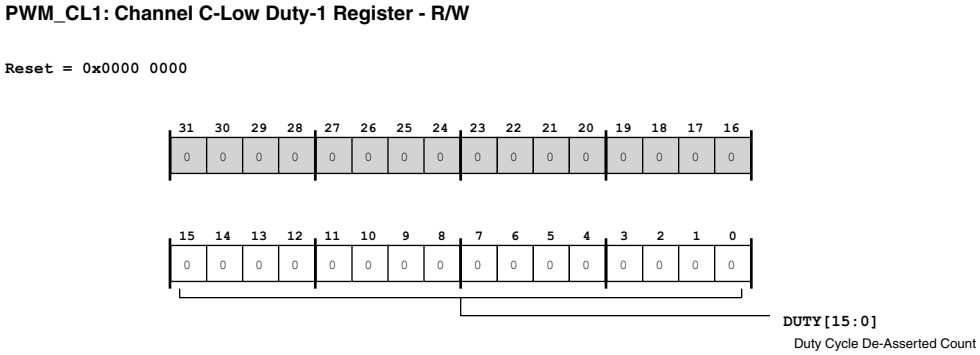


Figure 18-53: PWM_CL1 Register Diagram

Table 18-38: PWM_CL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count.

Channel D Control Register

The PWM_DCTL register selects the low and high side output pulse mode, enables low and high side output, and enables low/high side output crossover.

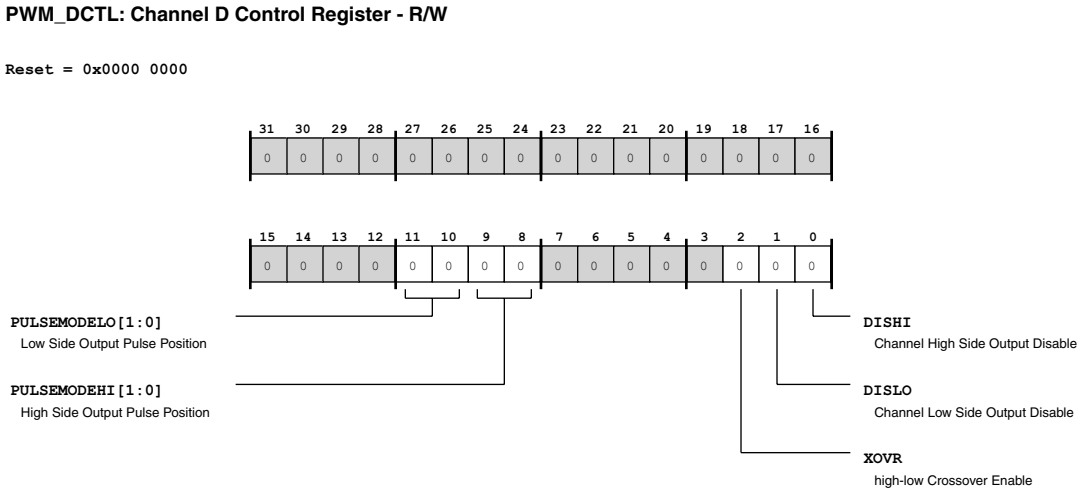


Figure 18-54: PWM_DCTL Register Diagram

Table 18-39: PWM_DCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	PULSEMODELO	Low Side Output Pulse Position. The PWM_DCTL.PULSEMODELO bits select the pulse position for Channel D low side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the centre of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_DLO register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the centre of the PWM period. This mode uses both the duty-cycle registers (PWM_DLO and PWM_DL1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty-cycle registers (PWM_DLO and PWM_DL1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half

Table 18-39: PWM_DCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	PULSEMODEHI	High Side Output Pulse Position. The PWM_DCTL.PULSEMODEHI bits select the pulse position for Channel D high side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the centre of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_DH0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the centre of the PWM period. This mode uses both the duty-cycle registers (PWM_DH0 and PWM_DH1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty-cycle registers (PWM_DH0 and PWM_DH1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half
2 (R/W)	XOVR	high-low Crossover Enable. The PWM_DCTL.XOVR bit enables crossover between the channels high and low side outputs. When enabled, this bit directs the PWM to send the low-side output through the high-side output pin and the high-side output through the low side output pin.	
		0	Disable Crossover
		1	Enable Crossover
1 (R/W)	DISLO	Channel Low Side Output Disable. The PWM_DCTL.DISLO bit enables the channels low side output.	
		0	Disable Low Side Output
		1	Enable Low Side Output
0 (R/W)	DISHI	Channel High Side Output Disable. The PWM_DCTL.DISHI bit enables the channels high side output.	
		0	Disable High Side Output
		1	Enable High Side Output

Channel D-High Duty-0 Register

The PWM_DH0 and PWM_DH1 registers determine the width for the high side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the high side output pulses for the Channel D duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_DCTL.PULSEMODEHI bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_DH0 register to determine the assertion and de-assertion count for the high side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts Channel D high pulse output for count less than PWM_DH0 and de-asserts this output for count greater than PWM_DH1.

The value range for the PWM_DH0 and PWM_DH1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_DH0 and PWM_DH1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_DT (two's complement) to +PWM_TM0/2 + PWM_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_DT (two's complement) to -PWM_TM0/2 - PWM_DT.

Note that using values in the PWM_DH0 or PWM_DH1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

PWM_DH0: Channel D-High Duty-0 Register - R/W

Reset = 0x0000 0000

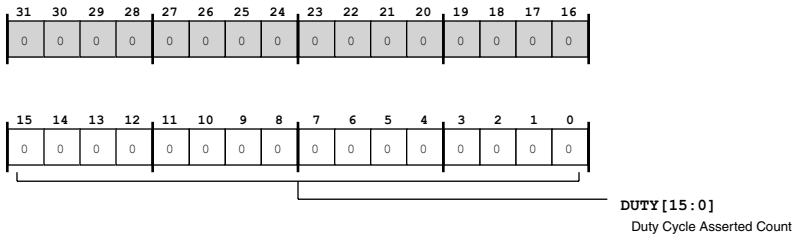


Figure 18-55: PWM_DH0 Register Diagram

Table 18-40: PWM_DH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_DH0.DUTY bits select the duty cycle asserted count for Channel D high side output.

Channel D-High Pulse Duty Register 1

The PWM_DH0 and PWM_DH1 registers determine the width for the high side output pulses. For more information, see the PWM_DH0 register description.

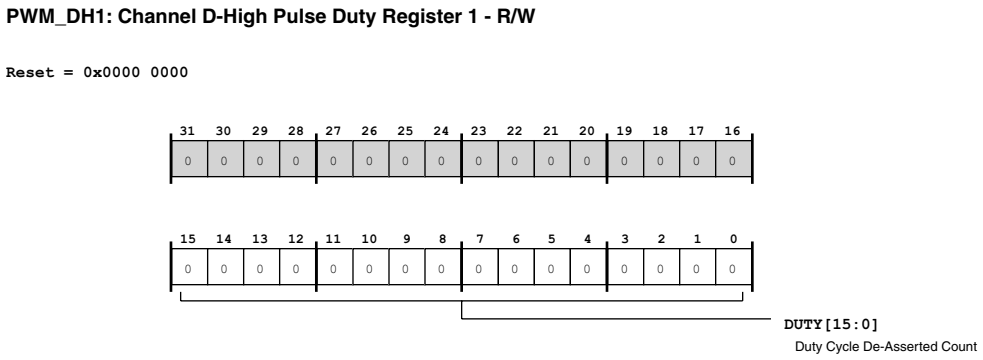


Figure 18-56: PWM_DH1 Register Diagram

Table 18-41: PWM_DH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_DH1.DUTY bits select the duty cycle de-asserted count for Channel D high side output.

Channel D-Low Pulse Duty Register 0

The PWM_DL0 and PWM_DL1 registers determine the width for the low side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the low side output pulses for the Channel D duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_DCTL.PULSEMODEL0 bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_DL0 register to determine the assertion and de-assertion count for the low side output pulses. When the pulse mode is

asymmetrical, left half, or right half, the PWM asserts Channel D low pulse output for count less than PWM_DL0 and de-asserts this output for count greater than PWM_DL1.

The value range for the PWM_DL0 and PWM_DL1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_DL0 and PWM_DL1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_DT (two's complement) to +PWM_TM0/2 + PWM_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_DT (two's complement) to -PWM_TM0/2 - PWM_DT.

Note that using values in the PWM_DL0 or PWM_DL1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

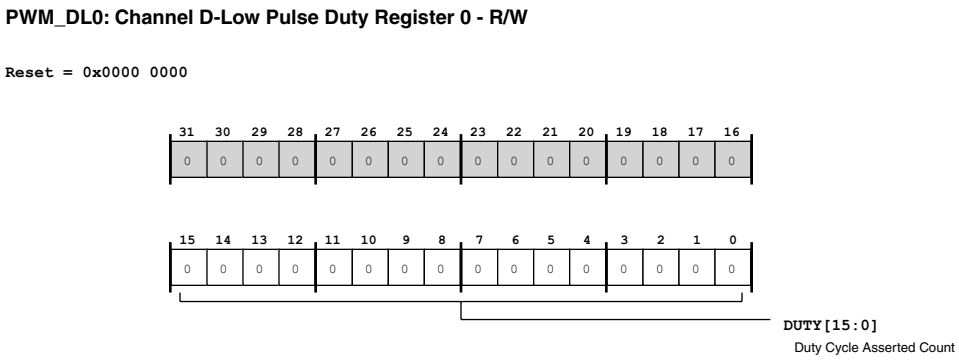


Figure 18-57: PWM_DL0 Register Diagram

Table 18-42: PWM_DL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_DL0.DUTY bits select the duty cycle asserted count for Channel D low side output.

Channel D-Low Pulse Duty Register 1

The PWM_DL0 and PWM_DL1 registers determine the width for the low side output pulses. For more information, see the PWM_DL0 register description.

PWM_DL1: Channel D-Low Pulse Duty Register 1 - R/W

Reset = 0x0000 0000

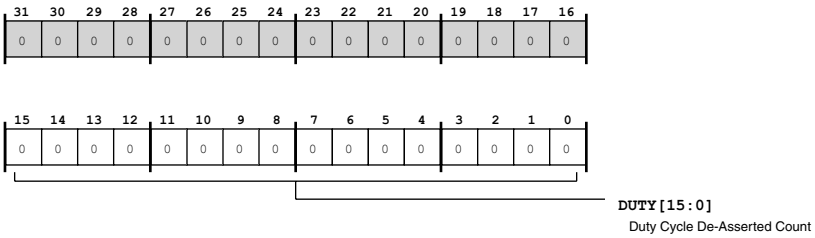


Figure 18-58: PWM_DL1 Register Diagram

Table 18-43: PWM_DL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_DL1.DUTY bits select the duty cycle de-asserted count for Channel D low side output.

19 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. The UART includes interrupt-handling hardware. Interrupts can be generated from multiple events.

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.

Partial modem status and control functionality is supported by the UART module 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.

One of the peripheral timers can be used to provide a hardware-assisted auto-baud detection mechanism for use with the UART. The timers are external to the UART.

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
- Additional 8-stage receive FIFO with programmable threshold interrupt
- Flexible transmit and receive interrupt timing
- 3 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 loop back
- Improved bit rate granularity
- LIN break command/Inter-frame gap transmission support

Table 19-1: UART Specifications

Feature	Availability
Protocol	
Master-Capable	Yes
Slave-Capable	Yes
Transmission Simplex	Yes
Transmission Half-Duplex	Yes
Transmission Full-Duplex	Yes
Access Type	
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-BF60x UART Register List

The universal asynchronous receiver/transmitter (UART) module is a full-duplex peripheral compatible with PC-style industry-standard UARTs. The UARTs convert data between serial and parallel formats. The serial communication follows an asynchronous protocol that supports various word length, stop bits, and parity generation options. The UARTs include interrupt-handling hardware. Interrupts can be gener-

ated from multiple events. A set of registers govern UART operations. For more information on UART functionality, see the UART register descriptions.

Table 19-2: ADSP-BF60x UART Register List

Name	Description
UART_CTL	Control Register
UART_STAT	Status Register
UART_SCR	Scratch Register
UART_CLK	Clock Rate Register
UART_IMSK	Interrupt Mask Register
UART_IMSK_SET	Interrupt Mask Set Register
UART_IMSK_CLR	Interrupt Mask Clear Register
UART_RBR	Receive Buffer Register
UART_THR	Transmit Hold Register
UART_TAIP	Transmit Address/Insert Pulse Register
UART_TSR	Transmit Shift Register
UART_RSR	Receive Shift Register
UART_TXCNT	Transmit Counter Register
UART_RXCNT	Receive Counter Register

ADSP-BF60x UART Interrupt List

Table 19-3: ADSP-BF60x UART Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
UART0 Transmit DMA	80	17	LEVEL
UART0 Receive DMA	81	18	LEVEL
UART0 Status	82		LEVEL
UART1 Transmit DMA	83	19	LEVEL

Table 19-3: ADSP-BF60x UART Interrupt List Interrupt List (Continued)

Description	Interrupt ID	DMA Channel	Sensitivity
UART1 Receive DMA	84	20	LEVEL
UART1 Status	85		LEVEL

ADSP-BF60x UART Trigger List

Table 19-4: ADSP-BF60x UART Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
UART0 Transmit DMA	39	PULSE/EDGE
UART0 Receive DMA	40	PULSE/EDGE
UART1 Transmit DMA	41	PULSE/EDGE
UART1 Receive DMA	42	PULSE/EDGE

Table 19-5: ADSP-BF60x UART Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
UART0 Transmit DMA	39	
UART0 Receive DMA	40	
UART1 Transmit DMA	41	
UART1 Receive DMA	42	

ADSP-BF60x UART DMA List

Table 19-6: ADSP-BF60x UART DMA List DMA Channel List

Description	DMA Channel
UART0 Transmit DMA	DMA17
UART0 Receive DMA	DMA18
UART1 Transmit DMA	DMA19
UART1 Receive DMA	DMA20

UART Block Diagram

The following figure shows a simplified block diagram of one UART module and how it interconnects to the processor system.

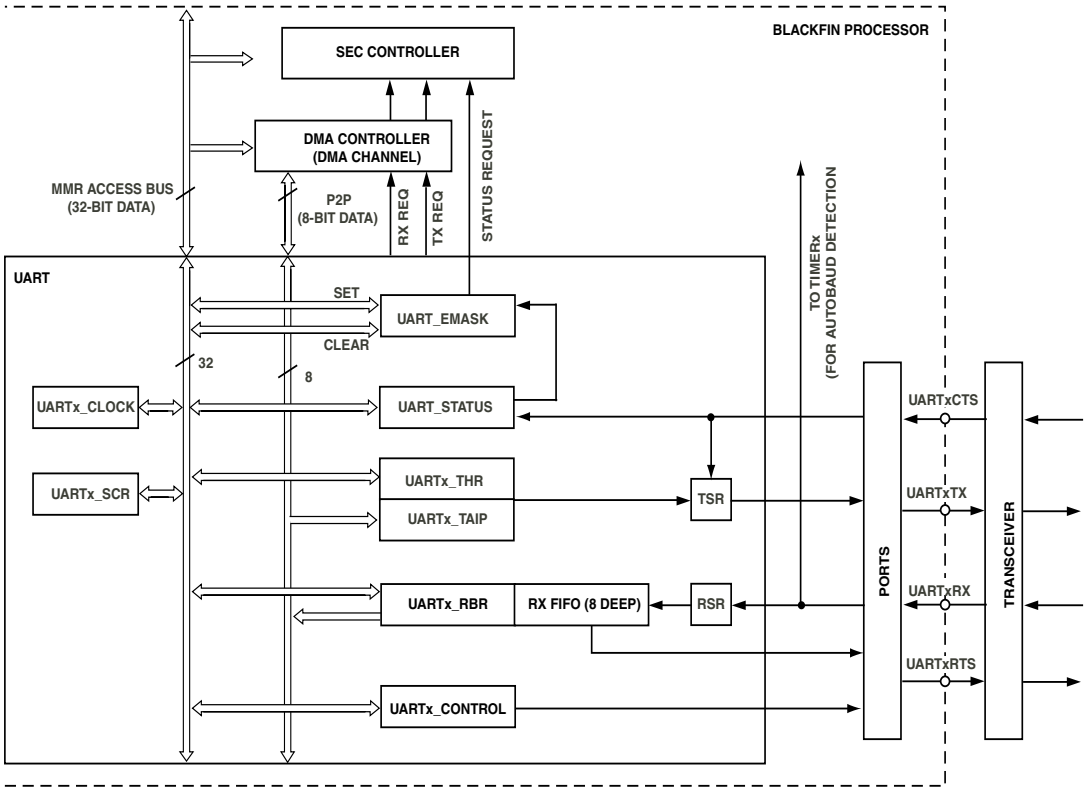


Figure 19-1: UART Block Diagram

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 can be used in either DMA or programmed core modes of operation. 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 (8-bit data width).

All UART registers are 32 bits wide and the registers connect to the peripheral MMR bus. Not all MMRs may be used and unused bits are zero-filled. The UART has three interrupt outputs described below.

- The transmit request 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 $\overline{\texttt{UART\_RX}}$ pin is also sensed by the alternative capture input (TIMER_ACIn) of one of the GP timers. When configured in capture mode, the GP timer can then be used to detect the bit rate of the received signal.

External Interface

Each UART features an $\overline{\texttt{UART\_RX}}$ (receive) and an $\overline{\texttt{UART\_TX}}$ (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 (for example, EIA-232, EIA-422, 4-wire EIA-485) or half duplex (for example, 2-wire EIA-485, LIN) standards. Additionally, the UART features a pair of clear to send, input pins ($\overline{\texttt{UART\_CTS}}$) and request to send, output pins ($\overline{\texttt{UART\_RTS}}$) for hardware flow control. UART signals are usually 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, a $\overline{\texttt{UART\_RTS}}$/$\overline{\texttt{UART\_CTS}}$ hardware flow control mechanism is supported. The $\overline{\texttt{UART\_RTS}}$ signal is an output that connects to the communication partner's $\overline{\texttt{UART\_CTS}}$ input. If data transfer is bidirectional, the handshake is as shown in the figure below.

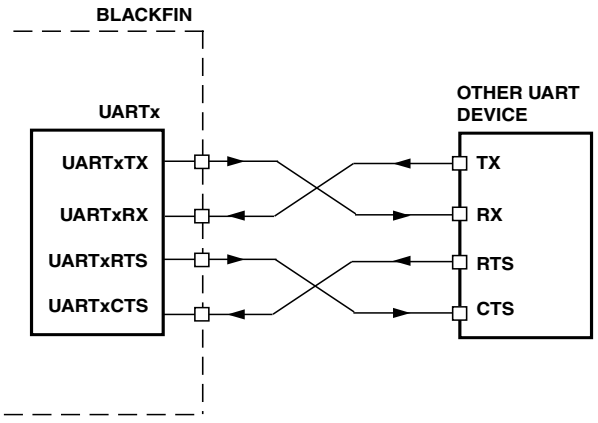


Figure 19-2: UART Hardware Flow

The receiver can de-assert the $\overline{\texttt{UART\_RTS}}$ signal to indicate that its receive buffer is getting full in both DMA and core mode because continued data transfers may cause an overrun error. Consequently, the transmitter pauses when the $\overline{\texttt{UART\_CTS}}$ input is in a de-asserted 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 $\overline{\text{UART\_CTS}}$ pin is asserted again, the transmitter resumes and loads the content of UART_THR register into the UART_TSR register.

UART Bit Rate Generation

The sample clock is characterized by the peripheral clock (*SCLK*) and the 16-bit divisor in the UART_CLKregister. The UART clock is enabled by the UART_CTL.ENbit. 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.EDBObit is set, so that the following equation applies:

Bit Rate = $SCLK/16^{(1\text{-EDBO})} \times$ Divisor

ADSP-BF60x Processor Example

The following table provides example divide factors required to support standard baud rates at a *SCLK* of 125 MHz.

Table 19-7: UART Bit Rate Examples With 125 MHz SCLK0

Bit Rate	D factor = 16			D factor = 1		
	DL	Actual	% Error	DL	Actual	% Error
2,400	3,255	2,400.15	0.006	52,083	2,400.02	0.001
4,800	1,628	4,798.83	0.024	26,042	4,799.94	0.001
9,600	814	9,597.67	0.024	13,021	9,599.88	0.001
19,200	407	19,195.33	0.024	6,510	19,201.23	0.006
38,400	203	38,485.22	0.022	3,255	38,402.46	0.006
57,600	136	57,444.85	0.269	2,170	57,603.69	0.006
115,200	68	114,889.71	0.269	1,085	115,207.37	0.006
921,600	8	976,562.50	5.964	136	919,117.65	0.269
1,500,000	5	1,562,500.00	4.167	83	1,516,021.10	0.402
3,000,000	3	2,604,166.67	13.194	42	2,976,190.48	0.794
6,250,000	1	7,812,500.00	25.000	20	6,250,000.00	0.000

NOTE: Careful selection of *SCLK*frequencies—that is, even multiples of desired bit rates— can result in lower error percentages.

Setting the bit clock equal to the sample clock (UART_CLK.EDBO=1) improves bit rate granularity and enables the bit clock to more closely match the bit rate of the communication partner. The disadvantage to this configuration is that the power dissipation is higher and the sample points may not be as accurate. Therefore, it is recommended to use UART_CLK.EDBO=1 mode only when bit rate accuracy is not acceptable in UART_CLK.EDBO=0 mode.

The UART_CLK.EDBO=1 mode is not intended to increase operation speed beyond the electrical limitations of the UART transfer protocol.

Autobaud Detection

At the chip level, the $\overline{\text{UART\_RX}}$ pin is usually routed to an alternate capture input (TIMER_ACIn) of a general-purpose timer. When working in width capture mode, this general-purpose timer can be used to automatically detect the bit rate applied to the $\overline{\text{UART\_RX}}$ pin by an external device. The capture capabilities of the timer are often used to supervise the bit rate at runtime. If the UART was communicating 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, autobaud detection is used for initial bit rate negotiations where the processor is most likely a slave device waiting for the host to send a predefined autobaud character. This is a common situation for UART booting. The UART_CTL.EN bit should not be enabled while autobaud detection is performed, to prevent the UART from starting a receive operation with incorrect bit rate matching. Alternatively, the UART can be disconnected from its $\overline{\text{UART\_RX}}$ pin by setting the UART_CTL.LOOP_EN bit.

A software routine can detect the pulse widths of serial stream bit cells. Because the sample base of the timer is synchronous with the UART operation (all derived from the same *SCLK*) the pulse widths can be used to calculate the bit rate divider for the UART by using the following formula.

A software routine can detect the pulse widths of serial stream bit cells. Because the sample base of the timer is synchronous with the UART operation—all derived from the same *SCLK*—the pulse widths can be used to calculate the bit rate divider for the UART by using the following formula.

$$\text{Divisor} = \text{TIMER\_TMRn\_WID}/16^{(1-\text{EDBO})} \times \text{Number of captured UART bits}$$

In order to increase the number of timer counts and therefore the resolution of the captured signal, it is recommended not to measure just the pulse width of a single bit, but to enlarge the pulse of interest over more bits. Traditionally, a NULL character (ASCII 0x00) is used in autobaud detection, as shown below.

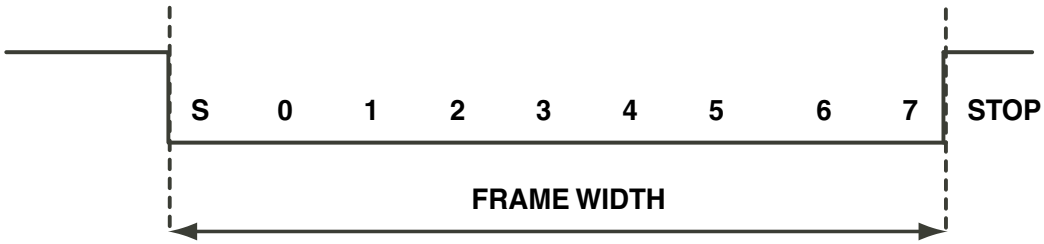


Figure 19-3: Autobaud Detection

Because the example frame encloses 8 data bits and 1 start bit, apply the following formula.

$$\text{Divisor} = \text{TIMER\_TMRn\_WID}/16^{(1-\text{EDBO})} \times 9$$

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 might not return adequate results without additional analog signal conditioning. Measuring signal periods works around this issue and is strongly recommended.

For example, predefine ASCII character "@" (0x40) as the autobaud detection character and measure the period between two subsequent falling edges. As shown in the figure below, 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.

Divisor = TIMER_TMRn_PER/$16^{(1-EDBO)} \times 8$

Or:

- Divisor = TIMER_TMRn_PER>> 7 if UART_CLK.EDBO=0
- Divisor = TIMER_TMRn_PER>> 3 if UART_CLK.EDBO=1

The following figure shows the ASCII "@" (0x40) detection character.

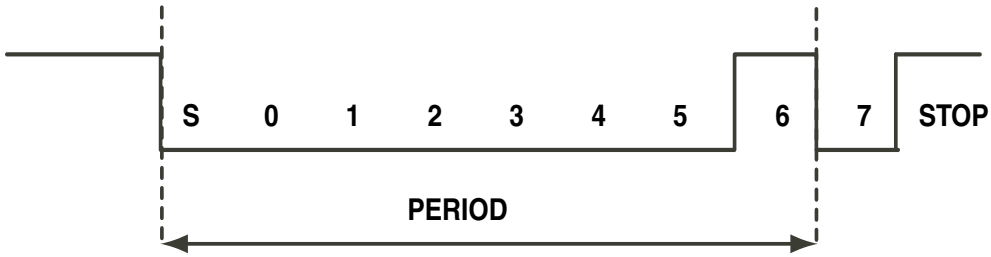


Figure 19-4: Autobaud Detection Character 0x40

UART Debug Features

The UART has the option to automatically calculate and transmit a parity bit. The following table summarizes parity behavior assuming 8-bit data words (UART_CTL.WLS=b#11).

Table 19-8: 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 loop back mode.

The UART can be set to internal loop back mode (UART_CTL.LOOP_EN=1). Loop back mode disconnects the receiver's input 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. Loop back mode also forces the $\overline{\text{UART\_RTS}}$ pin to de-assert, disconnects the UART_STAT.CTS bit from the $\overline{\text{UART\_CTS}}$ input pin, and connects the internal version of $\overline{\text{UART\_RTS}}$ to the UART_STAT.CTS bit.

Additionally, the $\overline{\text{UART\_TX}}$ pin can be forced to zero asynchronously using the UART_CTL.SB bit.

UART Operating Modes

The UART's main operating modes are described in the following sections.

- *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 format of received and transmitted character frames is controlled by the UART_CTL register. Data is always transmitted and received with the least significant bit (LSB) first.

The following figure shows a typical physical bit stream measured on a $\overline{\text{UART\_TX}}$ pin.

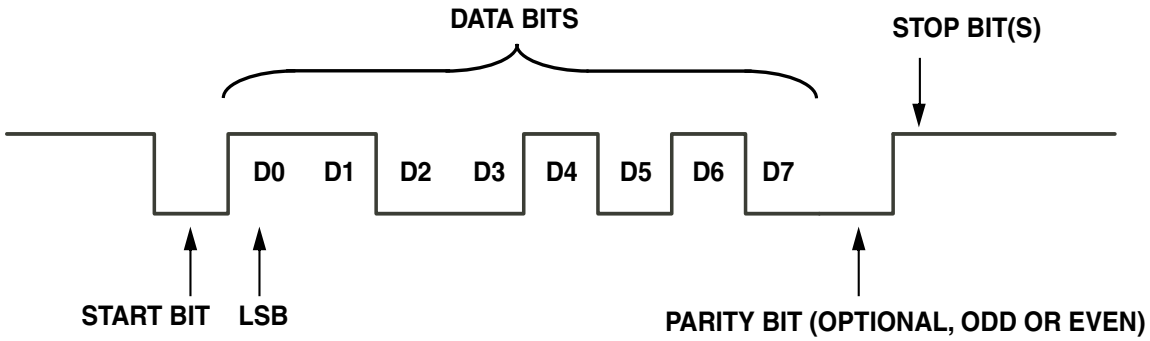


Figure 19-5: Bit Stream on a UART TX Pin Transmitting an "S" Character (0x53)

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. Pulse position modulation is not supported.

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 16x clock is used to determine an IrDA pulse sample window, from which the RZI modulated NRZ code is recovered.

NOTE: The `UART_CLK.EDBO` bit is not valid in IrDA mode—this bit should be cleared (=0) in this mode.

Multi-Drop Bus Mode

The UART protocol is not only used for point-to-point connections (defined in the EIA-232 standard), but also in networks where the EIA-485 standard is a popular representative of UART-based bus systems. In such networks node addressing is important.

In a multidrop bus (MDB) network for example, the UART frame is enhanced by an address bit. The address bit is inserted between the data bits and the optional parity bit. To configure the UART for MDB mode, the mode of operation bits (`UART_CTL.MOD` [5:4]) should be set to 01.

By convention the address bit is transmitted low for regular data bytes. If set it marks special address bytes that require the attention of all nodes on the network.

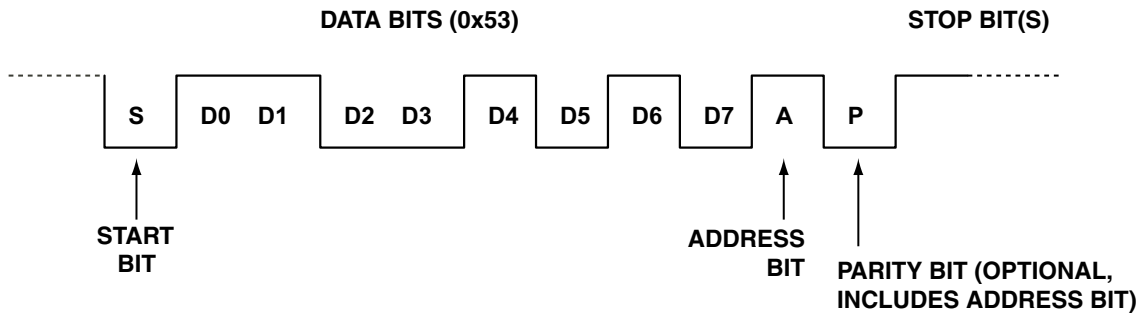


Figure 19-6: UART Frame with Address Bit

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/insert pulse register (UART_TAIP) instead, the same transmit operation is initiated with the only exception that the address bit is sent high.

The receiver's UART_STAT.ADDR bit signals whether the frame that was just received had the address bit set or not. It is updated by hardware every time a new frame has been received. When the enable address word interrupt bit (UART_IMSK.EAWI) is set, the reception of an address byte triggers a special status interrupt.

The address sticky bit (UART_STAT.ASTKY) is the sticky version of the UART_STAT.ADDR bit. It is set by hardware whenever the UART_STAT.ADDR bit is set. The UART_STAT.ASTKY bit can only be cleared by software 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 is not loaded by interrupt activity 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 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 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 is capable of transferring data using both the core and DMA. Receive and transmit paths operate completely 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, data is moved to and from the UART by the processor core. A write to the UART_THR register initiates the transmit operation. If no former operation is pending, the data is immediately passed from the UART_THR register to the UART_TSR register. There, it is shifted out at the bit rate characterized by the UART_CTL 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) is always transmitted first. This 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 length to be transmitted as a prologue to every multi-byte message. It length needs to be at least 13 bit times.

With previous UARTs there were two options to implement this protocol: either a null byte is transmitted with a temporarily lowered bit rate, or the period is generated by a software counter and the transmit pin is pulled low through the asynchronous set break (SB) mechanisms. 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, but when 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 and 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. Note that polarity can be inverted 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 may 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 is followed by 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. If however the pulse width needs to be determined more precisely, the GP timers can be used.

On ADSP-BF60x processors each $\overline{\text{UART\_RX}}$ pin is also routed to any of the GP timers through their alternate capture input (TACI). This is not only useful for bit rate detection (*autobaud*) but also helps to precisely measure the pulse widths on the $\overline{\text{UART\_RX}}$ input. Additionally, the new windowed watchdog width mode of the GP timers can issue an interrupt or a fault condition if the received pulse width is shorter than a bit time or longer than the worst case break condition.

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; that is, the UART_CTL.STB and UART_CTL.STBH bits have no impact to the receiver.

The UART receiver senses the falling edges of the receive input. When an edge is detected, the receiver starts sampling the input according to settings in the UART_CLK register. The start bit is sampled (majority sampling) close to its midpoint. If sampled low, a valid start condition is assumed. Otherwise, the detected falling edge is discarded.

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 can be seen as a 9-stage receive buffer. If the stop bit of the 9th 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 UART_STAT.OE bit is cleared by software. 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 over samples every bit 16 times and makes a majority decision based on the middle three samples. This improves immunity against noise and hazards on the line. Spurious pulses of less than two times the sampling clock period are disregarded.

Normally, every incoming bit is sampled at exactly the 7th, 8th and 9th sample clock. If, however, the UART_CLK.EDBO bit is set to 1 to achieve better bit rate granularity and accuracy as required at high operation speeds, the bits are one roughly sampled at 7/16th, 8/16th and 9/16th of their period. Hardware design should ensure that the incoming signal is stable between 6/16th and 10/16th of the nominal bit period.

Reception starts when a falling edge is detected on the $\overline{\text{UART\_RX}}$ input pin. The receiver attempts to see a start bit. The data is shifted into the UART_RSR register. After the 9th sample of the first stop bit is

processed, the received data is copied to the 8-stage receive FIFO and the UART_RSR recovers for further data reception.

The receiver samples data bits close to their midpoint. Because the receiver clock is usually asynchronous to the transmitter's data rate, the sampling point may 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 takes action (status update) and prepares itself 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, such that a 0 is transmitted as a high pulse of 16 UART clock periods and a 1 is transmitted as a low pulse for 16 UART clock periods. The leading edge of the pulse is then delayed by six UART clock periods. Similarly, the trailing edge of the pulse is truncated by eight UART clock periods. For a 16-cycle UART clock period, this results in the final representation of the original 0 as a high pulse of only 3/16 clock periods. The pulse is centered around the middle of the bit time, as shown in the figure below. 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 figure below, the error terms associated with the bit rate generator are very 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.

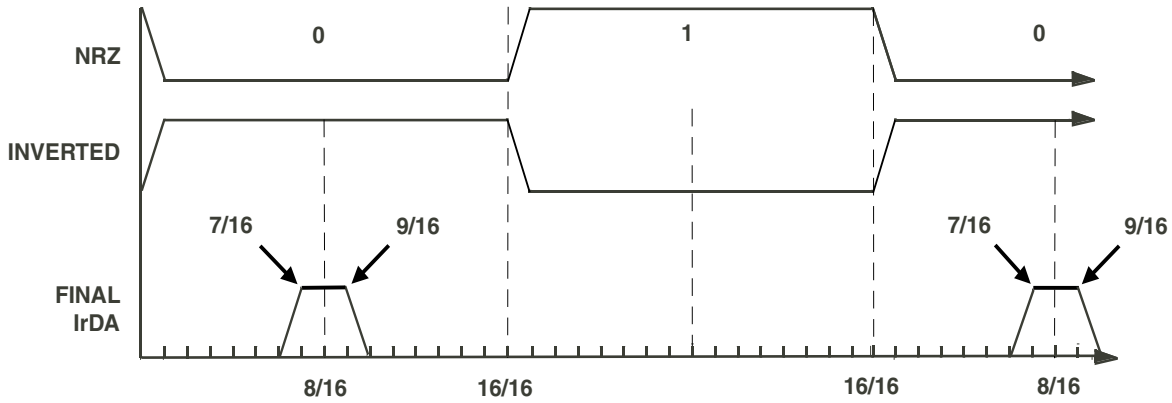


Figure 19-7: IrDA Transmit Pulse

IrDA Receive Operation

The IrDA receiver function is more complex than the transmit function. The receiver must discriminate the IrDA pulse and reject noise. To do this, 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 an initial pulse is seen. If the pulse is absent when the counter expires, it is considered a glitch. Otherwise, it is interpreted as a 0. This 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. Sources outside of the chip and not part of the transmitter can be avoided by appropriate shielding. 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 may 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.

The receive sampling window is determined by a counter that is clocked at the 16x bit-time sample clock. The sampling window is re-synchronized 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 following figure provides examples of each polarity type.

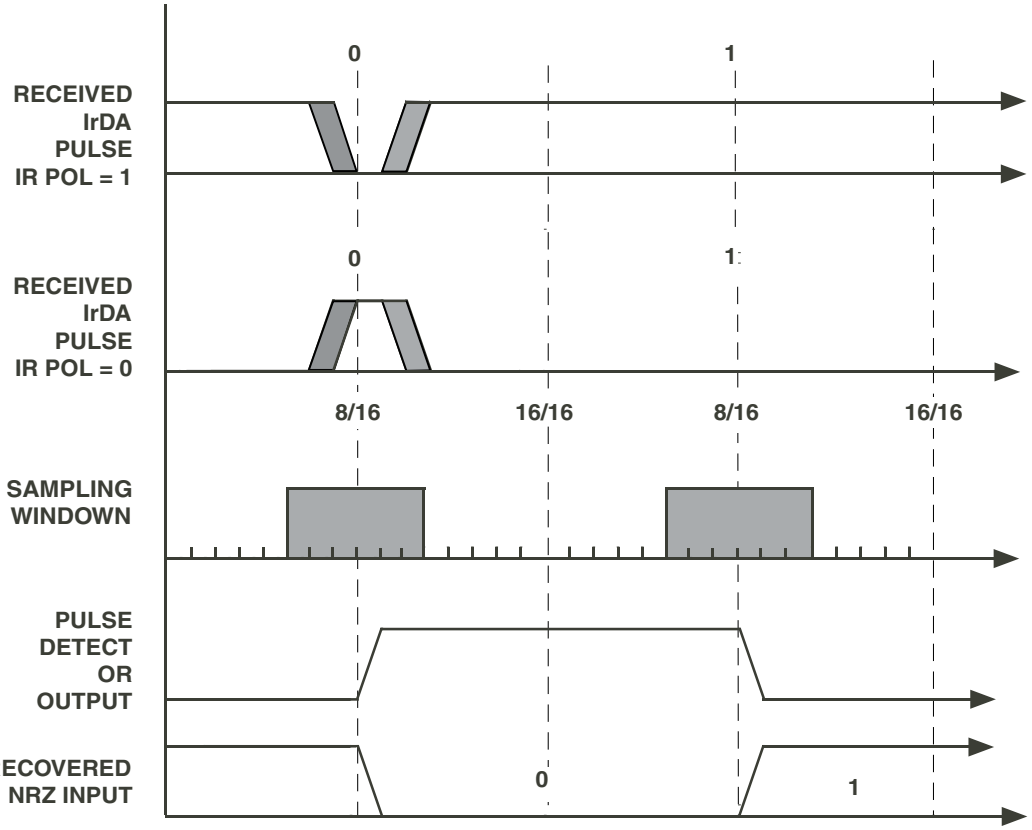


Figure 19-8: IrDA Receiver Pulse Detection

MDB Transmit Operation

In MDB mode, receive and transmit paths operate completely 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, every incoming bit is sampled at exactly the 7th, 8th and 9th sample clock. If, however, the UART_CLK.EDBO bit is set to achieve better bit rate granularity and accuracy as required at high operation speeds, the bits are roughly sampled at 7/16th, 8/16th and 9/16th of their period. Hardware design should 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 latency is determined by the bus activity and arbitration mechanism and not by the processor loading and interrupt priorities.

To enable UART DMA, first set up the system DMA control registers and then enable the UART_IMSK.ERBFI and/or UART_IMSK.ETBEI interrupts. This 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.

NOTE: The UART's status interrupt goes directly to the system event controller (SEC), bypassing the DMA unit completely.

For transmit DMA, programs should 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 is allowed to 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 might be lost.

When the UART_IMSK.ETBEI bit is set, an initial transmit DMA request is issued immediately. The program should then clear 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 strobe on the memory side is still enabled by the DMA_CFG.EN bit. If the DMA count is less than the DMA FIFO depth, which is 4, then the DMA interrupt might be requested before the UART_IMSK.ETBEI bit is set. If this is behavior not wanted, 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 is generated. Transmission may abort in the middle of the stream, causing data loss, if the UART clock was disabled without additional 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.DI_EN 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 DMA's last word has left the UART transmit buffers. If DI_EN and UART_IMSK.EDTPTI are set, when finishing STOP mode, the DMA requests two interrupts.

The UART's DMA supports 8-bit and 16-bit operation, but not 32-bit operation. Sign extension is also not supported.

Mixing DMA and Core Modes

Switching from DMA mode to core operation on the fly 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 by the time 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.

Therefore, when planning to switch from a DMA to the core mode, always set the DMA_CFG.SYNC bit in the word of the last descriptor or work unit before handing over control to core mode. Then, after the interrupt 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 very 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, the UART_IMSK.ETBEI bit should be pulsed to initiate DMA transmission.

Setting Up Hardware Flow Control

Use the following steps to setup UART hardware flow control.

1. Configure automatic or manual hardware flow control for the receiver through the UART_CTL.ARTS bit, and/or the transmitter through the UART_CTL.ACTS bit.
2. Configure $\overline{\text{UART\_CTS}}$ and $\overline{\text{UART\_RTS}}$ polarity through the UART_CTL.FCPOL bit.

AFTER COMPLETING THIS TASK:

On reset, when the UART is not yet enabled and the port multiplexing has not been programmed, the $\overline{\text{UART\_RTS}}$ pin is not driven. Some applications may require the $\overline{\text{UART\_RTS}}$ signal to be pulled to either state by a resistor during reset.

UART Event Control

Status flags in the UART_STAT register are available to signal data reception, parity, and error conditions, if required.

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 interrupts independently and easily.

The UART_IMSK registers are 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/or UART_IMSK.ETBEI bits in this register are normally set.

Each UART module has three interrupt outputs. One is dedicated for transmission, one for reception, and the third is used to report status events. Transmit and receive requests are routed 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. Note that 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 redirects receive and transmit requests to the status interrupt output. The status interrupt goes directly to the SEC without being routed through the DMA controller

Interrupt Servicing

UART writes and reads can be accomplished through interrupt service routines (ISRs). Separate interrupt lines are provided for transmit, receive, and status. The independent interrupts can be enabled individually by the UART_IMSK register group. The UART_CTL.EN bit must be set to enable UART transmit interrupts.

The ISRs can evaluate the status bits in the UART_STAT register to determine the signaling interrupt source. Interrupts must also be assigned and unmasked by the processor's system event controller. The ISRs must clear the interrupt latches explicitly. To reduce interrupt frequency on the receive side in core mode, the UART_IMSK.ERFCI status interrupt may be used 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

Transmit interrupts are enabled by the UART_IMSK_SET.ETBEI bit.

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.

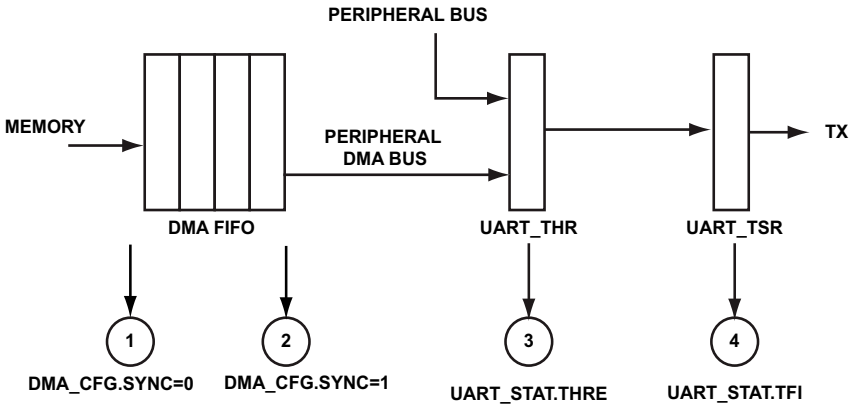


Figure 19-9: Transmit Interrupts

The transmit request is asserted along with the UART_STAT.THRE bit, indicating that the transmit buffer is ready for new data. Note that 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. Accordingly, the UART_IMSK.ETBEI bit can be cleared through the UART_IMSK_CLR register if the string transmission has completed.

The UART_STAT.THRE bit is cleared by hardware when new data is written to the UART_THR register. These writes also clear the transmit interrupt request. However, they also initiate further transmission. If continued transmission is not desired, the transmit request can alternatively be cleared through the UART_IMSK_CLR.ETBEI bit register. Transfers of data from the UART_THR register to the UART_TSR register re-set this status flag in the UART_STAT register.

The UART_STAT.TEMT bit can be interrogated to discover any ongoing transmission. The UART_STAT.TEMT bit's sticky counterpart, UART_STAT.TFI, indicates if the transmit buffer has drained and can trigger a status interrupt, if required. 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. At that time it is safe to disable the UART_CTL.EN bit or to three-state off-chip line drivers. By this time an interrupt can be generated 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 (hereafter 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, these status flags can be polled to determine when data is ready to move. Note that 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_SSTATn register without enabling the interrupts by the SEC_CCTLn 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, those two words are sent.

Receive Interrupts

Receive interrupts are enabled by the UART_IMSK_SET.ERBFI bit. 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.

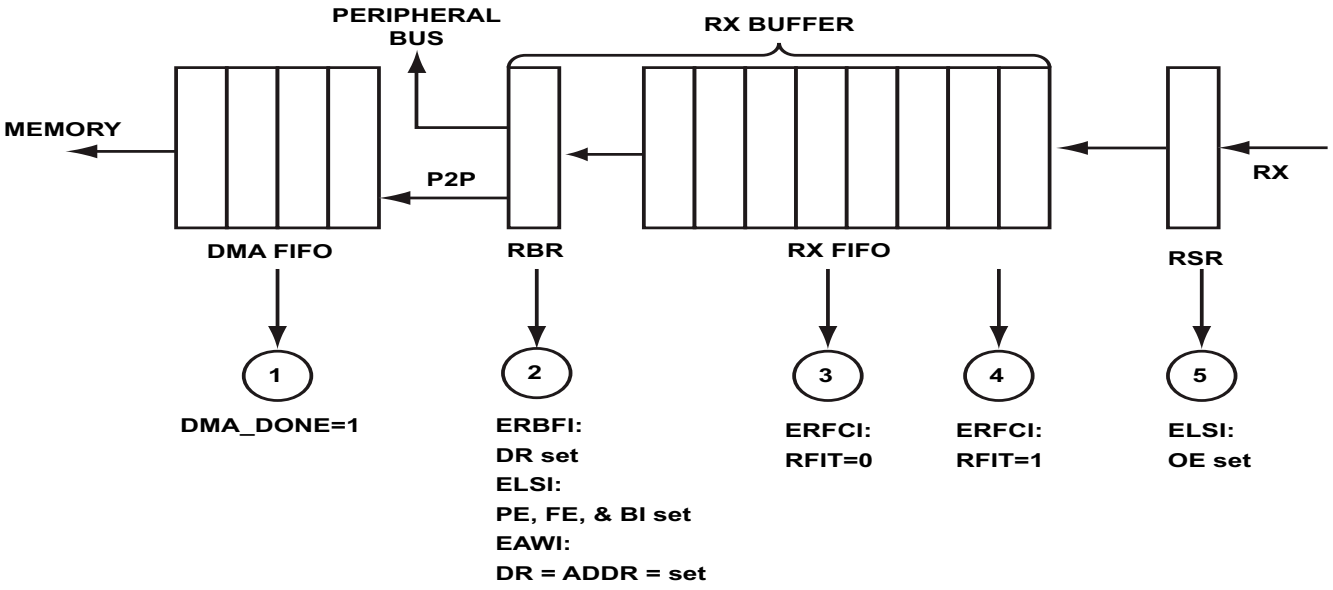


Figure 19-10: Receive Interrupts

The UART_STAT.DR, UART_STAT.ADDR, UART_STAT.ASTKY, UART_STAT.PE, UART_STAT.FE, and UART_STAT.BI bits are updated along with UART_RBR register. The UART_STAT.OE bit updated as soon as an overflow condition occurs (for example when a frame's stop bit is received and the receive FIFO is full). When the UART_RBR register is not read in time, the received data is protected from being overwritten by new data until the UART_STAT.OE bit is cleared by software. Only the content of the UART_RSR register can be overwritten in the overrun case.

The state of the 8-deep receive FIFO can be monitored by the UART_STAT.RFCS bit. The buffer's behavior is controlled by the UART_CTL.RFIT bit. 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. The UART_STAT.RFCS bit is cleared by hardware 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 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. Status interrupt requests are enabled by the bit.

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/invalid stop bit (UART_STAT.FE bit)
- Break indicator (UART_STAT.BI bit)

Status Interrupts

The UART status interrupt channels are used for the following purposes.

- Line status interrupts
- Flow control interrupts
- Receive FIFO threshold interrupts
- Transmission finished interrupt

Line status interrupts are enabled by the UART_IMSK.ELSI bit. If set, the status interrupt request is asserted with any of the UART_STAT.BI, UART_STAT.FE, UART_STAT.PE, or UART_STAT.OE receive errors bits. The error bits in the UART_STAT register are cleared by W1C operation. Once all error conditions are cleared, the interrupt request de-asserts.

The receive FIFO count interrupt is enabled by the UART_IMSK_SET.ERFCI bit. If set, a status interrupt is generated when the UART_STAT.RFCS is active. The UART_STAT.RFCS bit indicates a receive buffer threshold level. If the UART_CTL.RFITbit is cleared, software can safely read two words out of the UART_RBRregister 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 and the UART_STAT.RFCS bit clear when the UART_RBR is read a sufficient number 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 flow control interrupts are enabled by the UART_IMSK_SET.EDSSI bit. If active, a status interrupt is generated when the sticky UART_STAT.SCTS bit register is set, indicating that the transmitter's $\overline{\text{UART\_CTS}}$ input been re-asserted. A W1C operation to the SCTS bit clears the interrupt request.

A transmission finished interrupt is enabled by the UART_IMSK_SET.ETFI bit. 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 completely finished. The interrupt request is cleared by a W1C operation to the UART_STAT.TFI bit.

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 include the address (UART_STAT.ADDR) bit, address sticky (UART_STAT.ASTKY) bit and enable address word interrupt (UART_IMSK.EAWI). One of the key features of the multi-drop bus protocol is its address bit, which signifies to the slaves that the master is transmitting an address word (to be read by all) or a data word (to be read by the addressed slave only). The UART hardware provides for an efficient method of handling the situation described above with the use of UART_STAT.ASTKY bit.

NOTE: The UART_STAT.ASTKY bit is used in multi-drop bus mode to indicate if a peripheral is currently being addressed. The UART_STAT.ASTKY bit is a sticky version of the UART_STAT.ADDR bit and is set by hardware whenever the UART_STAT.ADDR bit is set. It can only be cleared by software with a W1C operation. With the ASTKY bit set, words are received irrespective of the mode bit/address bit setting. With the UART_STAT.ASTKY bit cleared, only address words (mode bit=1) are received and words with mode bit=0 is ignored (not moved from the UART_RSR to the Receive FIFO) in MDB mode. This bit does not affect reception in non-MDB modes.

UART Programming Model

The following sections provide basic procedures for configuring various UART operations.

Detecting Autobaud

Please refer to *Autobaud Detection* for more information. The required steps are:

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_TMRn_CFG.TMODE=1101, TIMER_TMRn_CFG.OUTDIS=1, TIMER_TMRn_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.

   STEP RESULT: The bit rate can be derived from the timer period register value according to the formula provided in the *Autobaud Detection* section.

Using Common Initialization Steps

Certain steps are common to all UART modes, regardless of using the core or the DMA running the transfers.

1. All UART signals are multiplexed and compete with other functions at pin level. First, the port registers need to be programmed according to the guidelines in the PORTs chapter.
2. Program the UART_CLK register. Refer to *UART Bit Rate Generation*.
3. Program the UART_CTL register and enable the UART clock.

Using Core Transfers

A core transmit operation is accomplished by writing data into the UART_THR register, when the UART_STAT.THRE bit is set. 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, and/or the transmitter through the UART_CTL.ACTS bit.
2. Configure $\overline{\text{UART\_CTS}}$ and $\overline{\text{UART\_RTS}}$ polarity through the UART_CTL.FCPOL bit.

ADSP-BF60x UART Register Descriptions

UART (UART) contains the following registers.

Table 19-9: ADSP-BF60x UART Register List

Name	Description
UART_CTL	Control Register
UART_STAT	Status Register
UART_SCR	Scratch Register

Table 19-9: ADSP-BF60x UART Register List (Continued)

Name	Description
UART_CLK	Clock Rate Register
UART_IMSK	Interrupt Mask Register
UART_IMSK_SET	Interrupt Mask Set Register
UART_IMSK_CLR	Interrupt Mask Clear Register
UART_RBR	Receive Buffer Register
UART_THR	Transmit Hold Register
UART_TAIP	Transmit Address/Insert Pulse Register
UART_TSR	Transmit Shift Register
UART_RSR	Receive Shift Register
UART_TXCNT	Transmit Counter Register
UART_RXCNT	Receive Counter Register

Control Register

The UART_CTL register provides enable/disable control for internal UART and for the IrDA mode of operation. This register 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.

UART_CTL: Control Register - R/W

Reset = 0x0000 0000

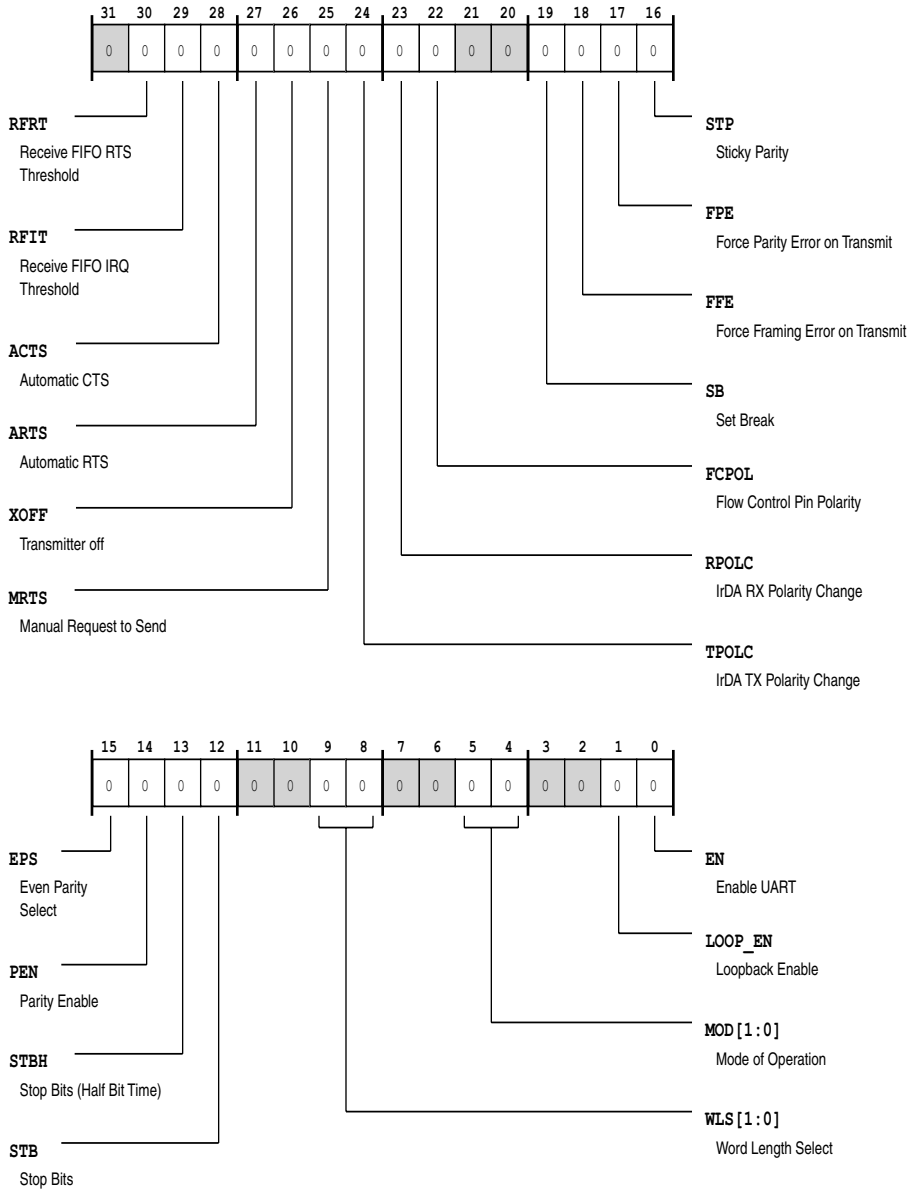


Figure 19-11: UART_CTL Register Diagram

Table 19-10: UART_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	RFRT	Receive FIFO RTS Threshold. The UART_CTL.RFRT bit controls $\overline{\text{UART\_RTS}}$ pin assertion and de-assertion timing. This bit is ignored if UART_CTL.ARTS is cleared. If set, the $\overline{\text{UART\_RTS}}$ pin is de-asserted when the receive buffer already holds seven words and an eighth start bit is detected. It is re-asserted when the FIFO contains seven words or less. If cleared, de-assert $\overline{\text{UART\_RTS}}$ pin when the RX buffer already holds four words and a fifth start bit is detected. The $\overline{\text{UART\_RTS}}$ pin is re-asserted when the RX buffer contains no more than 4 words.	
		0	De-assert RTS if RX FIFO word count > 4; assert if <= 4
		1	De-assert RTS if RX FIFO word count > 7; assert if <= 7
29 (R/W)	RFIT	Receive FIFO IRQ Threshold. The UART_CTL.RFIT bit controls the timing of the UART_STAT.RFCS bit. If UART_CTL.RFIT is cleared, the receive threshold is two. If UART_CTL.RFIT 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 UART_CTL.ACTS bit must be set to enable the $\overline{\text{UART\_CTS}}$ input pin for $\overline{\text{UART\_TX}}$ handshaking. If enabled, the UART_STAT.CTS bit holds the value (if UART_CTL.FCPOL is set) or complement value (if UART_CTL.FCPOL is cleared) of the $\overline{\text{UART\_CTS}}$ input pin. The UART_STAT.CTS bit can be used to determine whether the external device is ready to receive data (if UART_STAT.CTS set) or whether it is busy (if UART_STAT.CTS cleared). If UART_CTL.ACTS is cleared, the $\overline{\text{UART\_TX}}$ handshaking protocol is disabled, and the $\overline{\text{UART\_TX}}$ line transmits data whenever there is data to send, regardless of the value of $\overline{\text{UART\_CTS}}$. Software can pause ongoing transmission by setting the UART_CTL.XOFF bit.	
		0	Disable TX handshaking protocol
		1	Enable TX handshaking protocol

Table 19-10: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
27 (R/W)	ARTS	Automatic RTS. The UART_CTL.ARTS bit must be set to enable the $\overline{\text{UART\_RTS}}$ input pin for $\overline{\text{UART\_TX}}$ handshaking.If set, hardware guarantees minimal $\overline{\text{UART\_RTS}}$ pin de-assertion pulse width of at least the number of data bits defined by the UART_CTL.WLS bit field. If cleared, the $\overline{\text{UART\_RTS}}$ pin is not generated automatically by hardware. The $\overline{\text{UART\_RTS}}$ pin can still be manually controlled by the UART_CTL.MRTS bit, and software is responsible for $\overline{\text{UART\_RTS}}$ pulse width control (if needed).	
		0	Disable RX handshaking protocol.
		1	Enable RX handshaking protocol.
26 (R/W)	XOFF	Transmitter off. The UART_CTL.XOFF bit (if set) turns off transmission (XOFF) by preventing the content of THR from being continued to TSR. When set, this bit turns on transmission (XON). The state of the UART_CTL.XOFF bit is ignored if the UART_CTL.ACTS bit is set.	
		0	Transmission ON, if ACTS=0
		1	Transmission OFF, if ACTS=0
25 (R/W)	MRTS	Manual Request to Send. The UART_CTL.MRTS bit controls the state of the $\overline{\text{UART\_RTS}}$ output pin when the UART_CTL.ARTS bit is cleared. When UART_CTL.MRTS is cleared, the UART de-asserts the $\overline{\text{UART\_RTS}}$ pin, signaling to the external device that the UART is not ready to receive. When UART_CTL.MRTS is set, the UART asserts the $\overline{\text{UART\_RTS}}$ pin, signaling to the external device that the UART is ready to receive.	
		0	De-assert RTS pin when ARTS=0
		1	Assert RTS pin when ARTS=0
24 (R/W)	TPOLC	IrDA TX Polarity Change. The UART_CTL.TPOLC bit selects the active low/high polarity for IrDA communications. This bit only is effective in IrDA mode. If set, in IrDA mode, the $\overline{\text{UART\_TX}}$ pin idles high. In UART or MDB mode, it is inverted-NRZ. If cleared, in IrDA mode, the $\overline{\text{UART\_TX}}$ pin idles low. In UART or MDB mode, it is NRZ.	
		0	Active-low TX polarity setting
		1	Active-high TX polarity setting

Table 19-10: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
23 (R/W)	RPOLC	IrDA RX Polarity Change. The UART_CTL.RPOLC bit selects the active low/high polarity for IrDA communications. This bit only is effective in IrDA mode. If set, in IrDA mode, the $\overline{\text{UART\_RX}}$ pin idles high. In UART or MDB mode, it is inverted-NRZ. If cleared, in IrDA mode, the $\overline{\text{UART\_RX}}$ pin idles low. In UART or MDB mode, it is NRZ.	
		0	Active-low RX polarity setting
		1	Active-high RX polarity setting
22 (R/W)	FCPOL	Flow Control Pin Polarity. The UART_CTL.FCPOL select the polarities of the $\overline{\text{UART\_CTS}}$ and $\overline{\text{UART\_RTS}}$ pins. When UART_CTL.FCPOL is cleared, the $\overline{\text{UART\_RTS}}$ and $\overline{\text{UART\_CTS}}$ pins are active low, and UART is halted when the $\overline{\text{UART\_RTS}}$ and $\overline{\text{UART\_CTS}}$ pin state is high. When UART_CTL.FCPOL is set, the $\overline{\text{UART\_RTS}}$ and $\overline{\text{UART\_CTS}}$ pins are active high, and UART is halted when the $\overline{\text{UART\_RTS}}$ and $\overline{\text{UART\_CTS}}$ pin state is low.	
		0	Active low CTS/RTS
		1	Active high CTS/RTS
19 (R/W)	SB	Set Break. If set, the UART_CTL.SB bit forces the $\overline{\text{UART\_TX}}$ 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 $\overline{\text{UART\_TX}}$ pin normally drives high, it can be used as a flag output pin, if the UART is not used. (For example, if UART_CTL.TPOLC is cleared, drive $\overline{\text{UART\_TX}}$ pin low; or if UART_CTL.TPOLC is set, drive $\overline{\text{UART\_TX}}$ pin high.)	
		0	No force
		1	Force TX pin to 0
18 (R/W)	FFE	Force Framing Error on Transmit. The UART_CTL.FFE bit is intended for test purposes. This bit is useful for debugging software, especially in loopback mode.	
		0	Normal operation
		1	Force error

Table 19-10: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	FPE	Force Parity Error on Transmit. The UART_CTL.FPE bit is intended for test purposes. This bit is useful for debugging software, especially in loopback mode.	
		0	Normal operation
		1	Force parity error
16 (R/W)	STP	Sticky Parity. The UART_CTL.STP 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 EPS 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 UART_CTL.STP cleared and UART_CTL.EPS set. Then, the count of logical-1 bits must be a even value. If the UART_CTL.STP bit is set, hardware parity calculation is disabled. In this case, the sent and received parity equals the inverted UART_CTL.EPS bit.	
		0	No Forced Parity
		1	Force (Stick) Parity to Defined Value (if PEN=1)
15 (R/W)	EPS	Even Parity Select.	
		0	Odd parity
		1	Even parity
14 (R/W)	PEN	Parity Enable. The UART_CTL.PEN enables parity transmission and parity check. The UART_CTL.PEN 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 UART_CTL.STP and UART_CTL.EPS 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 dont match. If UART_CTL.PEN is cleared, the UART_CTL.STP and the UART_CTL.EPS bits are ignored.	
		0	Disable
		1	Enable parity transmit and check

Table 19-10: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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 UART_CTL.STB bit controls how many stop bits are appended to transmitted data.	
		0	1 stop bit
		1	2 stop bits
9:8 (R/W)	WLS	Word Length Select. The UART_CTL.WLS 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 UART_CTL.MOD selects the UART operation mode (UMOD).	
		0	UART Mode
		1	MDB Mode
		2	IrDA SIR Mode
1 (R/W)	LOOP_EN	Loopback Enable. The UART_CTL.LOOP_EN enables UART loopback mode. When set, this bit disconnects the receivers input from the $\overline{\text{UART\_RX}}$ pin, and internally redirects the transmit output to the receiver. The $\overline{\text{UART\_TX}}$ pin remains active and continues to transmit data externally as well. Loopback mode also forces the $\overline{\text{UART\_RTS}}$ pin to its de-assertive state, disconnects the $\overline{\text{UART\_CTS}}$ bit from the $\overline{\text{UART\_CTS}}$ input pin, and directly connects the UART_CTL.MRTS bit to the UART_STAT.CTS bit. In loopback mode, setting the UART_CTL.MRTS bit sets the UART_STAT.CTS bit and enables the UARTs transmitter. Clearing to the UART_CTL.MRTS bit clears the UART_STAT.CTS bit and disables the UARTs transmitter.	
		0	Disable
		1	Enable

Table 19-10: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable UART. The UART_CTL.EN enables 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

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 $\overline{\text{UART\_CTS}}$ 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.

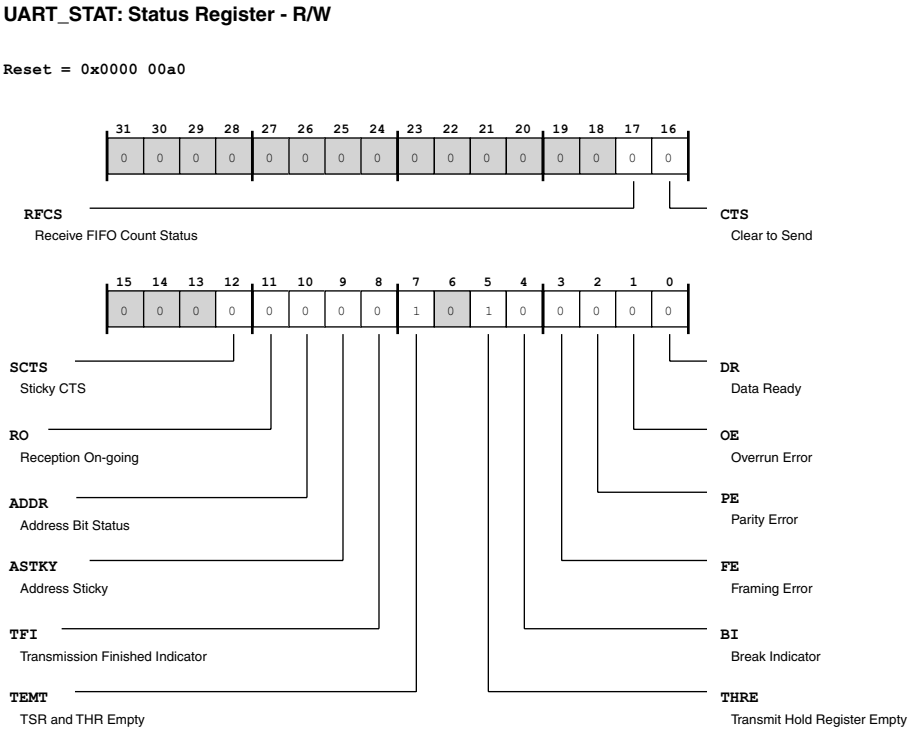


Figure 19-12: UART_STAT Register Diagram

Table 19-11: UART_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/NW)	RFCS	Receive FIFO Count Status. The UART_STAT.RFCS bit is set when the receive buffer holds more or equal entries than a certain threshold. The threshold is controlled by the UART_CTL.RFIT bit. If UART_CTL.RFIT is cleared, the threshold is four entries. If UART_CTL.RFIT is set, the threshold is seven entries. The UART_STAT.RFCS bit is cleared when the UART_RBR register is read sufficient times until the buffer is drained below the threshold. The UART_STAT.RFCS bit can trigger a status interrupt if enabled by the UART_IMSK_SET.ERFCI bit.	
		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)
16 (R/NW)	CTS	Clear to Send. The UART_STAT.CTS bit holds the value (if UART_CTL.FCPOL set) or the complement value (if UART_CTL.FCPOL cleared) of the $\overline{\text{UART\_CTS}}$ input pin. The UART_CTL.ACTS bit must be set to enable this feature. The core can read the value of the UART_STAT.CTS bit to determine whether the external device is ready to receive (UART_STAT.CTS set) or if it is busy (UART_STAT.CTS cleared). If UART_CTL.ACTS is cleared, the $\overline{\text{UART\_TX}}$ handshaking protocol is disabled, and the UART transmits data as long as there is data to transmit, regardless of the value of UART_STAT.CTS. When UART_CTL.ACTS is cleared, the software can pause transmission temporarily by setting the XOFF bit. Note that in loopback mode (UART_CTL.LOOP_EN set), the UART_STAT.CTS bit is disconnected from the $\overline{\text{UART\_CTS}}$ input pin. Instead, the bit is directly connected to the UART_CTL.MRTS 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 UART_STAT.SCTS bit is a sticky bit that is set when UART_STAT.CTS transitions from 0 to 1. The UART_STAT.SCTS bit is cleared by software with a W1C operation. This bit can trigger a line status interrupt if enabled by the UART_IMSK_SET.EDSSI bit.	
		0	CTS has not transitioned from low to high
		1	CTS has transitioned from low to high

Table 19-11: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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 UART_STAT.ADDR bit is used to mirror the address bit of the word in UART_RBR in multi-drop bus protocol, and is enabled only in MDB mode. The UART_STAT.ADDR bit is updated by hardware upon detecting a received word with the address bit in UART_RBR set or cleared. Additionally, software can set the ADDR bit with a write-1-to-set (W1S) operation.	
		0	Address bit is low
		1	Address bit is high
9 (R/W1C)	ASTKY	Address Sticky. The UART_STAT.ASTKY 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 UART_STAT.ADDR bit and is set by hardware when setting the UART_STAT.ADDR bit. The UART_STAT.ASTKY bit can only be cleared by software with a write-one-to-clear (W1C) operation. With the UART_STAT.ASTKY bit set, words will be received irrespective of the UART_CTL.MOD bit or UART_STAT.ADDR bit selection. With the UART_STAT.ASTKY bit cleared, only address words (UART_CTL.MOD bit set) will be received and words with UART_CTL.MOD bit cleared are ignored (not moved from the RSR to the RX FIFO) in MDB mode. The UART_STAT.ASTKY 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 UART_STAT.TFI bit is a sticky version of the UART_STAT.TEMT bit. While UART_STAT.TEMT is automatically cleared by hardware when new data is written to the UART_THR register, the sticky UART_STAT.TFI bit remains set, until it is cleared by software (W1C). The UART_STAT.TFI bit enables more flexible transmit interrupt timing.	
		0	TEMT did not transition from 0 to 1
		1	TEMT transition from 0 to 1

Table 19-11: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	TEMT	TSR and THR Empty. The UART_STAT.TEMT bit indicates that the UART_THR and UART_TAIP registers and the UART_TSR register are empty. In this case, the program is permitted to write to the UART_THR and UART_TAIP registers twice without losing data. The UART_STAT.TEMT bit can also be used as indicator that pending UART transmission is completed. At that time, it is safe to disable the UART_CTL.EN bit or to three-state the off-chip line driver.	
		0	Not empty TSR/THR
		1	TSR/THR Empty
5 (R/NW)	THRE	Transmit Hold Register Empty. The UART_STAT.THRE bit indicates that the UART transmit channel is ready for new data and software can write to the UART_THR and UART_TAIP registers. Writes to the UART_THR and UART_TAIP registers clear the UART_STAT.THRE. The bit is set again when the UART_THR and UART_TAIP registers are empty and ready to accept data.	
		0	Not empty THR/TAIP
		1	Empty THR/TAIP
4 (R/W1C)	BI	Break Indicator. The UART_STAT.BI bit indicates that the first stop bit is sampled low and the entire data word, including parity bit, consists of low bits only. (This condition indicates that $\overline{\text{UART\_RX}}$ was held low for more than the maximum word length.) The UART_STAT.BI bit is updated simultaneously with the UART_STAT.DR bit, that is, by the time the first stop bit is received or when data is loaded from the receive FIFO to the UART_RBR 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

Table 19-11: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W1C)	FE	Framing Error. The UART_STAT.FE bit indicates that the first stop bit is sampled. This bit is updated simultaneously with the UART_STAT.DR bit, that is, by the time the first stop bit is received or when data is loaded from the receive FIFO to the UART_RBR register. The UART_STAT.FE bit is sticky and can be cleared by W1C operations. Note that invalid stop bits can be simulated by setting the UART_CTL.FFE bit.	
		0	No error
		1	Invalid stop bit error
2 (R/W1C)	PE	Parity Error. The UART_STAT.PE bit indicates that the received parity bit does not match the expected value. This bit is updated simultaneously with the UART_STAT.DR bit, that is, by the time the first stop bit is received or when data is loaded from the receive FIFO to the UART_RBR register. The UART_STAT.PE bit is sticky and can be cleared by W1C operations. Note that invalid parity bits can be simulated by setting the UART_CTL.FPE bit.	
		0	No parity error
		1	Parity error
1 (R/W1C)	OE	Overrun Error. The UART_STAT.OE 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 UART_RBR register in time. In DMA receive mode, overruns are very unlikely to happen ever. After an overrun occurs, the UART_RBR and receive FIFO are protected from being overwritten by new data until the UART_STAT.OE bit is cleared by software. The content of the UART_RSR register is lost as soon as the overrun occurs. The UART_STAT.OE bit is sticky and can be cleared by W1C operations.	
		0	No overrun
		1	Overrun error

Table 19-11: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	DR	Data Ready. The UART_STAT.DR bit indicates that data is available in the receiver and can be read from the UART_RBR register. The bit is set by hardware when the receiver detects the first valid stop bit. The bit is cleared by hardware when the UART_RBR register is read.	
		0	No new data
		1	New data in RBR

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.

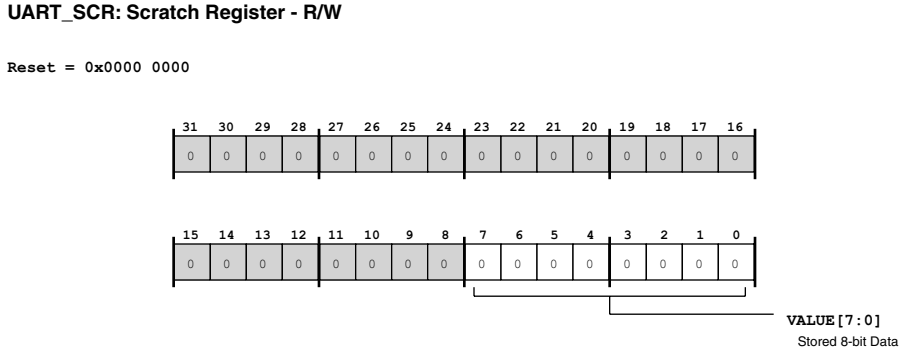


Figure 19-13: UART_SCR Register Diagram

Table 19-12: UART_SCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Stored 8-bit Data.

Clock Rate Register

The UART_CLK register divides the system clock (SCLK) down to the bit clock.

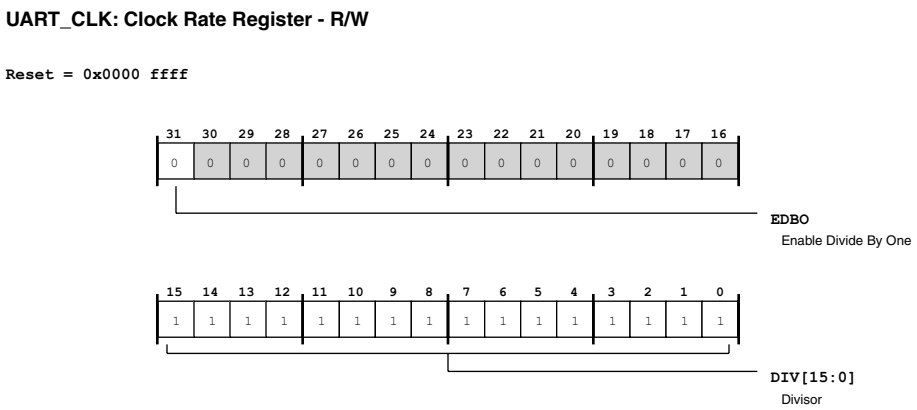


Figure 19-14: UART_CLK Register Diagram

Table 19-13: UART_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	EDBO	Enable Divide By One. The UART_CLK.EDBO bit enables bypassing of the divide-by-16 prescaler in bit clock generation. This improves bit rate granularity, especially at high bit rates. Do not set this bit in IrDA mode.	
		0	Bit clock prescaler = 16
		1	Bit clock prescaler = 1
15:0 (R/W)	DIV	Divisor. The UART_CLK.DIV provides the divisor for the UART's clock bit rate calculation. The bit rate is defined by: Bit Rate = SCLK / ($16^{(1\text{-EDBo})}$ x UART_CLK.DIV)	

Interrupt Mask 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) interrupts, and writing ones to UART_IMSK_CLR 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/or UART_IMSK.ETBEI bits are normally set. Setting this register without enabling system DMA causes the UART to notify the processor of data inventory state by means of 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 data transmit, data receive, and line status events independently, regardless whether DMA is enabled or not. If no DMA channels are assigned to the UART, set the UART_IMSK.ELSI bit to reroute transmit and receive interrupts to the status interrupt output.

With system DMA enabled, the UART uses DMA to transfer data to or from the processor. Dedicated DMA channels are available to receive and transmit operation. Line error handling can be configured completely independently from the receive/transmit setup.

The UART's DMA is enabled by first setting up the system DMA control registers and then enabling the UART_IMSK.ERBFI and/or UART_IMSK.ETBEI interrupts. This 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. However, UART's error interrupt goes directly to the system interrupt handling unit, bypassing the DMA unit completely.

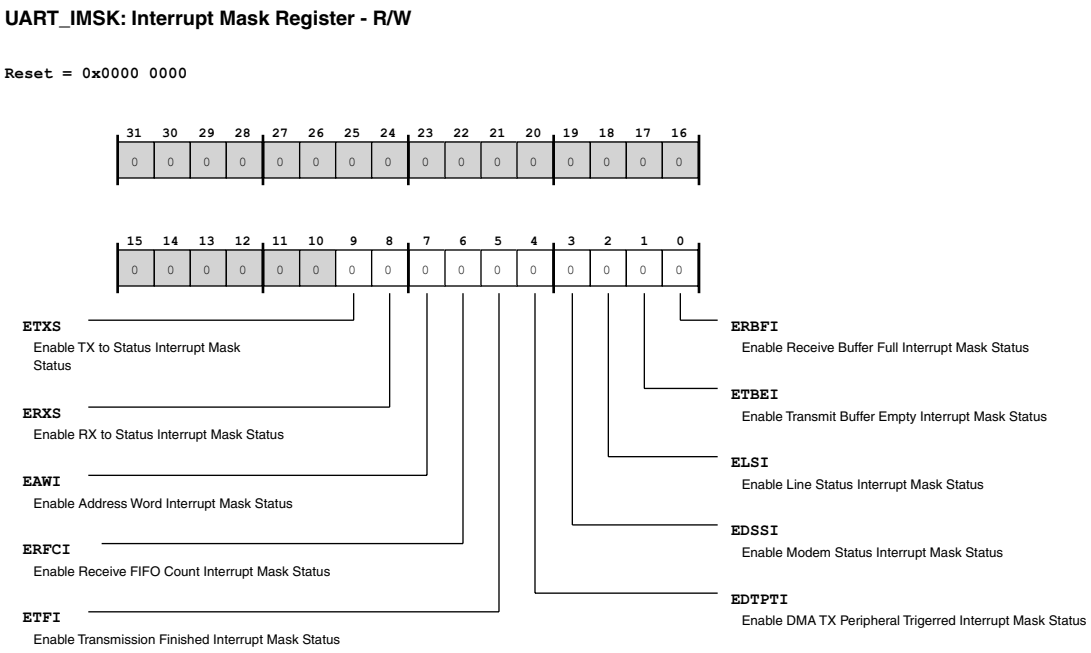


Figure 19-15: UART_IMSK Register Diagram

Table 19-14: 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 interrupts to status interrupt output. If cleared, TX interrupts 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 interrupts to status interrupt output. If cleared, RX interrupts are routed to normal interrupt outputs.	
		0	Interrupt is masked
		1	Interrupt is unmasked
7 (R/W)	EAWI	Enable Address Word Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.EAWI bit indicates generation of a status interrupt when an Address word in MDB-mode is present in the UART_RBR. A received word is an address word if the UART_STAT.ADDR 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 UART_IMSK.ERFCI bit indicates enabling of the receive buffer threshold interrupt if signaled by the UART_STAT.RFCS bit. Read the UART_RBR register sufficient times to clear the interrupt request.	
		0	Interrupt is masked
		1	Interrupt is unmasked

Table 19-14: UART_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	ETFI	Enable Transmission Finished Interrupt Mask Status. If set (interrupt unmasked) the UART_IMSK.ETFI 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 UART_STAT.TFI. The UART_IMSK.ETFI interrupt can be used to avoid expensive polling of the UART_STAT.TEMT bit, when the UART clock or line drivers should be disabled after transmission has completed. W1C the UART_STAT.TFI bit to clear the interrupt request. In DMA operation, the UART_IMSK.ETFI bits functionality might be preferred.	
		0	Interrupt is masked
		1	Interrupt is unmasked
4 (R/W)	EDTPTI	Enable DMA TX Peripheral Trigerred Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.EDTPTI bit indicates enabling of the DMA completion interrupt 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 by the time the UART_STAT.TEMT bit goes high after the last DMA data word is transmitted. When UART_IMSK.EDTPTI is set, usually the DDE_CFG_INT field is cleared to 00 in a STOP mode DMA. This set up suppresses the normal completion interrupt, and the UART_STAT.TEMT event is signaled through the DMA controller and triggers the DMA interrupt. If both (DDE_CFG_INT not 00 and UART_IMSK.EDTPTI set), two interrupts are requested at the end of a STOP mode DMA.	
		0	Interrupt is masked
		1	Interrupt is unmasked
3 (R/W)	EDSSI	Enable Modem Status Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.EDSSI bit indicates enabling of a modem status interrupt on the same status interrupt channel when the UART_STAT.SCTS bit is set. This indicates $\overline{\text{UART\_CTS}}$ pin re-assertion. Write-1-to-clear (W1C) the UART_STAT.SCTS bit to clear the interrupt request.	
		0	Interrupt is masked
		1	Interrupt is unmasked

Table 19-14: UART_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	ELSI	Enable Line Status Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.ELSI bit indicates that redirection of TX and RX interrupt requests to the status interrupt output of the UART by OR'ing them with the UART_STAT.OE, UART_STAT.PE, UART_STAT.FE, and UART_STAT.BI interrupt requests. Set this bit when no DMA channel is associated with the UART. Enabling UART_IMSK.ELSI disables the RX/TX interrupt channels and negates the UART_IMSK.EDTPTI 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 UART_IMSK.ETBEI bit indicates generation of a TX interrupt if the UART_STAT.THRE 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 UART_IMSK.ERBFI indicates generation of an RX interrupt if the UART_STAT.DR bit is set.	
		0	Interrupt is masked
		1	Interrupt is unmasked

Interrupt Mask Set 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) interrupts, 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.

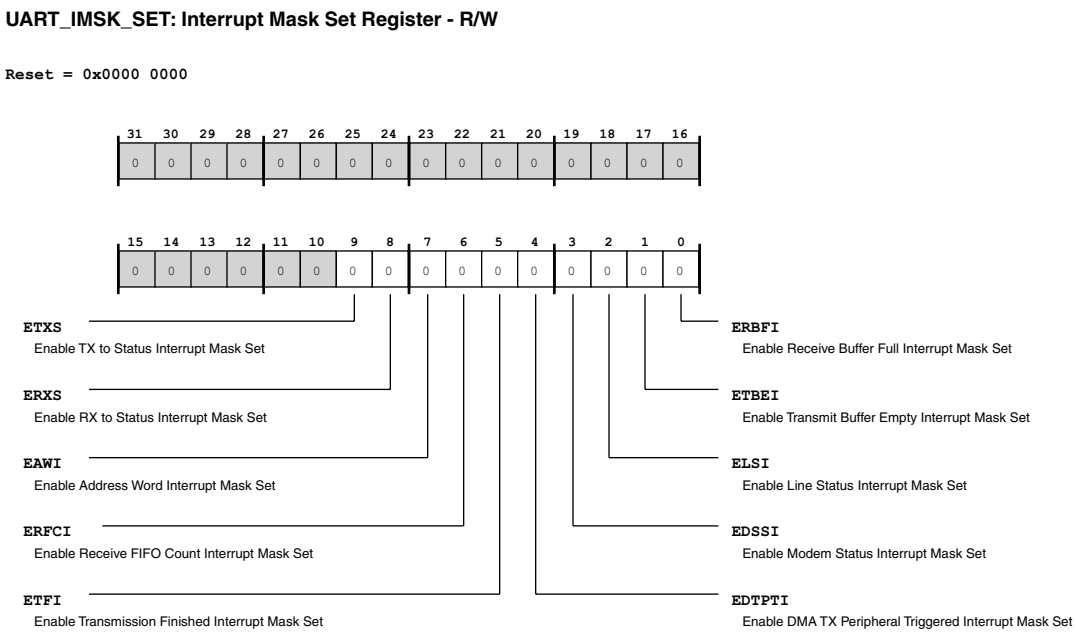


Figure 19-16: UART_IMSK_SET Register Diagram

Table 19-15: 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
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

Table 19-15: UART_IMSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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

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) interrupts, 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.

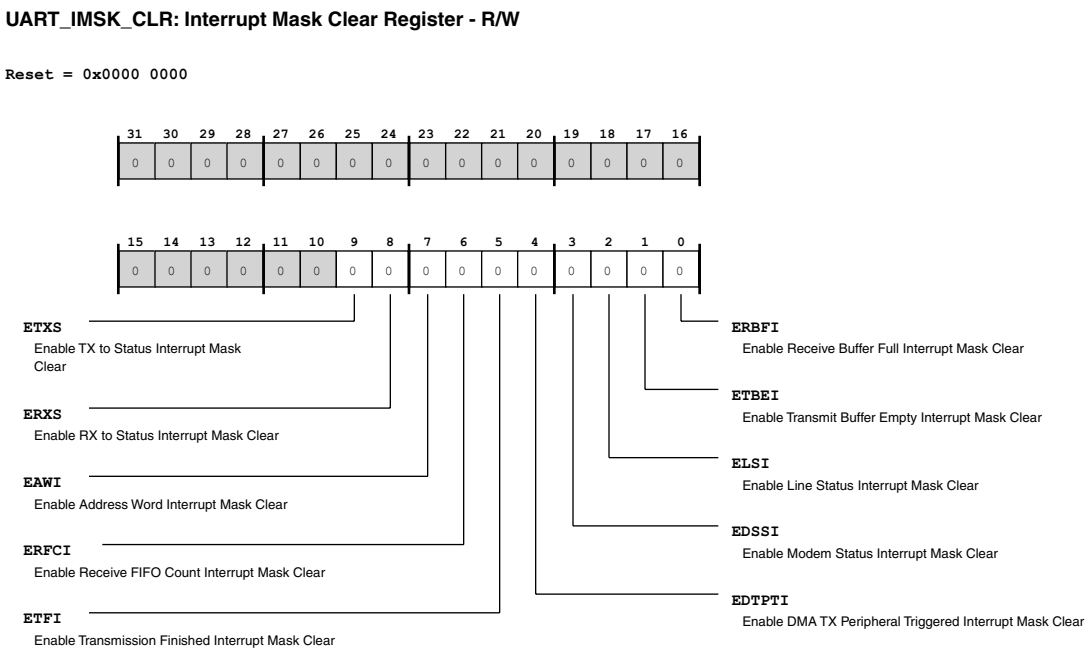


Figure 19-17: UART_IMSK_CLR Register Diagram

Table 19-16: 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
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

Table 19-16: UART_IMSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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

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.

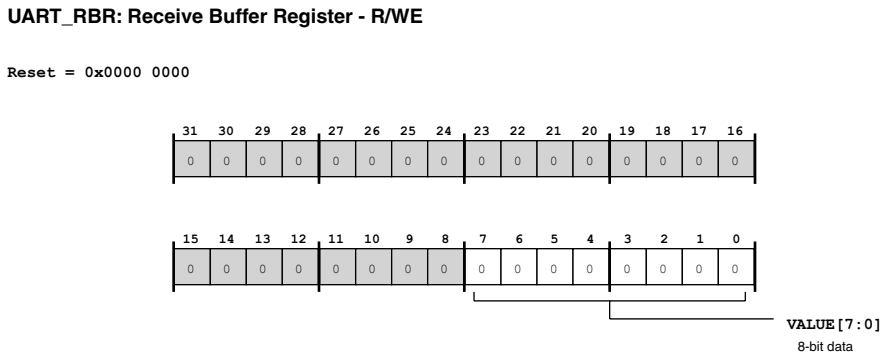


Figure 19-18: UART_RBR Register Diagram

Table 19-17: UART_RBR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	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.

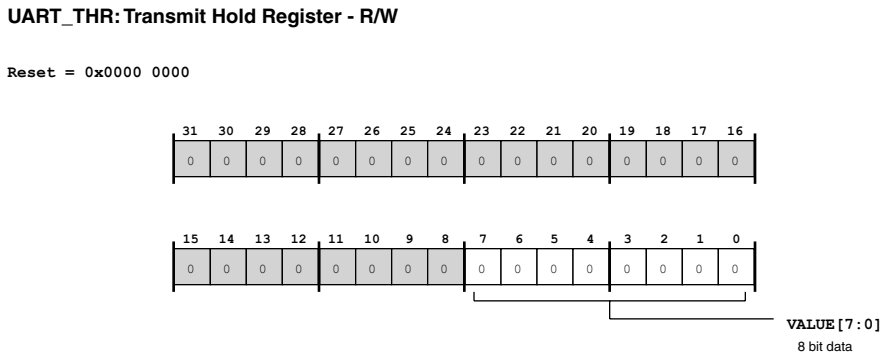


Figure 19-19: UART_THR Register Diagram

Table 19-18: UART_THR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	8 bit data.

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.

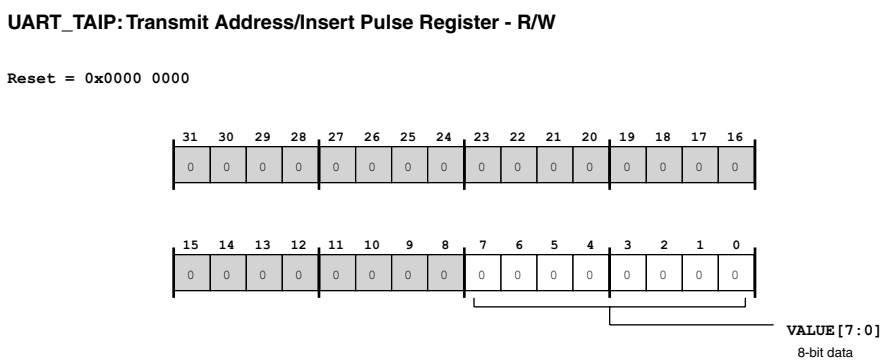


Figure 19-20: UART_TAIP Register Diagram

Table 19-19: UART_TAIP 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.

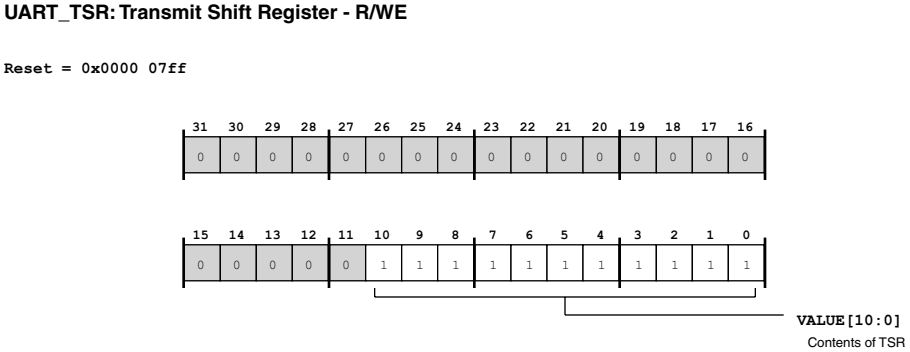


Figure 19-21: UART_TSR Register Diagram

Table 19-20: UART_TSR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/NW)	VALUE	Contents of TSR.

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).

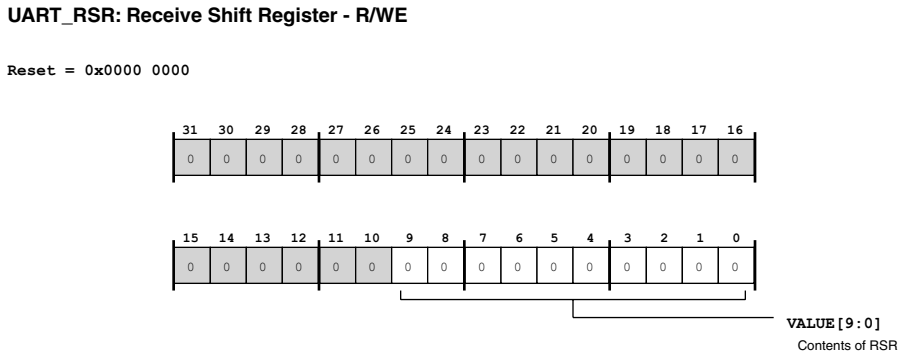


Figure 19-22: UART_RSR Register Diagram

Table 19-21: UART_RSR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/NW)	VALUE	Contents of RSR.

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).

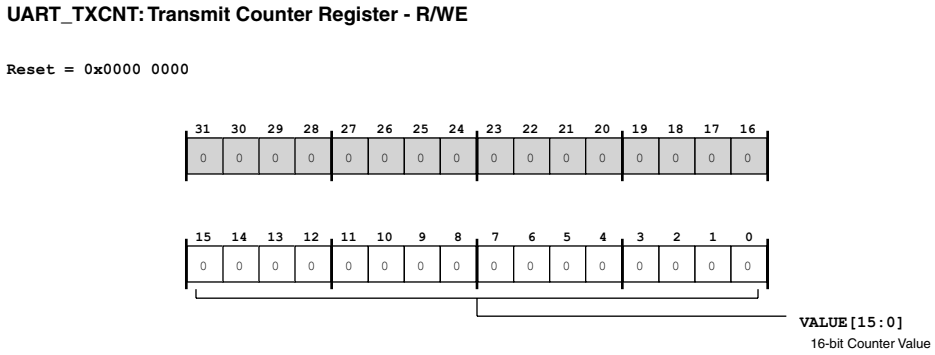


Figure 19-23: UART_TXCNT Register Diagram

Table 19-22: UART_TXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	16-bit Counter Value.

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).

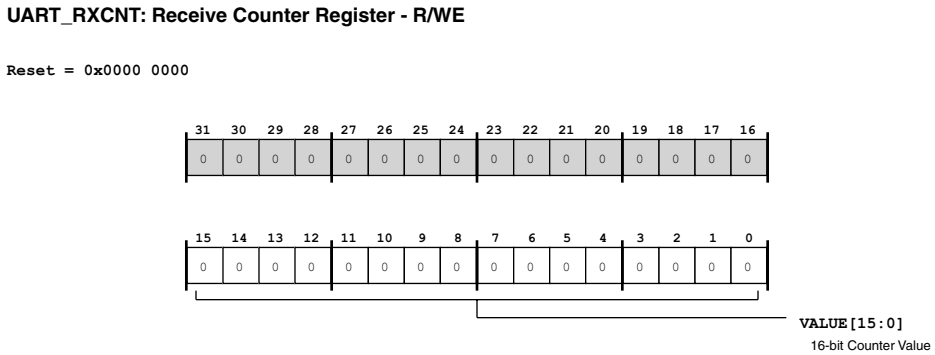


Figure 19-24: UART_RXCNT Register Diagram

Table 19-23: UART_RXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	16-bit Counter Value.

20 2-Wire Interface (TWI)

The processor has a 2-wire interface (TWI), that provides a simple exchange method of control data between multiple devices. The TWI module is compatible with the widely used I^2C 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 5 V 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 I^2C bus protocol.

TWI Features

The TWI is fully compatible with the widely used I^2C 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*

TWI Functional Description

The TWI interface is a shift register that serially transmits and receives data bits, one 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-BF60x TWI Register List

The 2-wire interface TWI controller allows a device to interface to an inter IC bus as specified by the Philips I^2C Bus Specification version 2.1 dated January 2000. A set of registers govern TWI operations. For more information on TWI functionality, see the TWI register descriptions.

Table 20-1: ADSP-BF60x TWI Register List

Name	Description
TWI_CLKDIV	SCL Clock Divider Register
TWI_CTL	Control Register
TWI_SLVCTL	Slave Mode Control Register
TWI_SLVSTAT	Slave Mode Status Register
TWI_SLVADDR	Slave Mode Address Register
TWI_MSTRCTL	Master Mode Control Registers
TWI_MSTRSTAT	Master Mode Status Register
TWI_MSTRADDR	Master Mode Address Register
TWI_ISTAT	Interrupt Status Register
TWI_IMSK	Interrupt Mask Register
TWI_FIFOCTL	FIFO Control Register
TWI_FIFOSTAT	FIFO Status Register
TWI_TXDATA8	Tx Data Single-Byte Register
TWI_TXDATA16	Tx Data Double-Byte Register
TWI_RXDATA8	Rx Data Single-Byte Register

Table 20-1: ADSP-BF60x TWI Register List (Continued)

Name	Description
TWI_RXDATA16	Rx Data Double-Byte Register

ADSP-BF60x TWI Interrupt List

Table 20-2: ADSP-BF60x TWI Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
TWI0 Data Interrupt	32		LEVEL
TWI1 Data Interrupt	33		LEVEL

TWI Block Diagram

The following figure shows the basic blocks of the TWI interface.

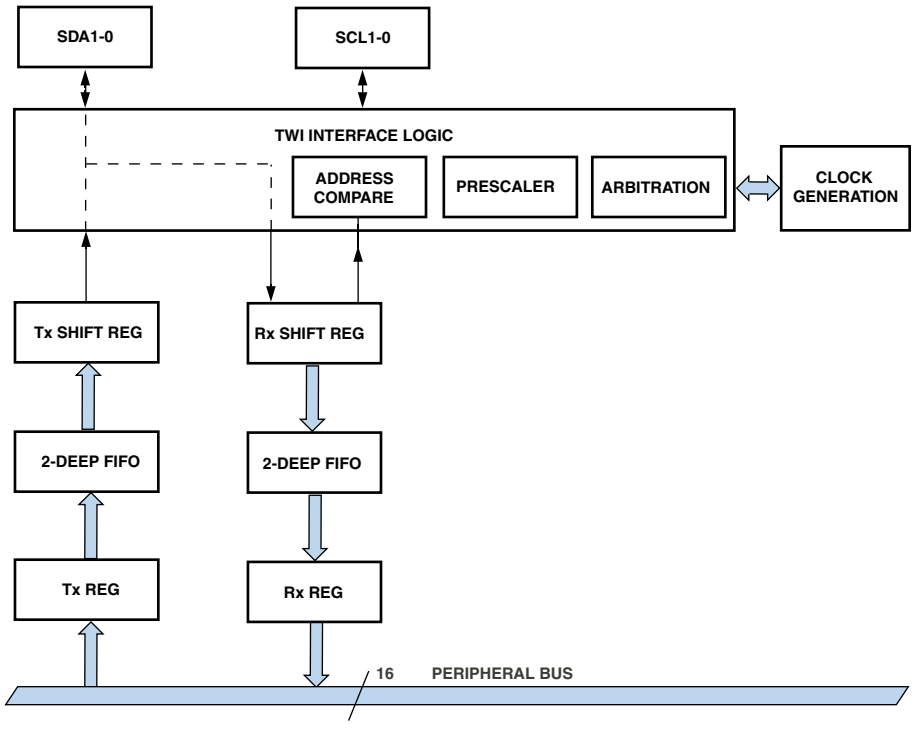


Figure 20-1: TWI Block Diagram

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 I^2C bus and no other external connections or logic are required.

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) is used to set the TWI_CTL.PRESCALE value which sets the relationship between the system clock (*SCLK*) and the TWI controller's internally timed events. 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. The value of prescale must be set to create an internal time reference with a period of 10 MHz. It is represented as a 7-bit binary value as shown below.

PRESCALE = f_{SCLK}/10MHz

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 MHz/10 MHz = 10. A prescale value of 14 in this case ensures that all timing requirements are met.

During master mode operation, the TWI_CLKDIV register values are used 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 (represented as an 8-bit binary value) the TWI_SCL waits before a new clock low period begins, assuming a single master. The TWI_CLKDIV.CLKLO field specifies 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 as shown below.

TWI_CLKDIV = TWI_SCL period/10 MHz time reference.

For example, for an TWI_SCL of 400 kHz (period = 1/400 kHz = 2500 ns) and an internal time reference of 10 MHz (period = 100 ns) the following equation is used:

TWI_CLKDIV = 2500 ns/100 ns = 25

Therefore, a TWI_SCL signal with a 30% duty cycle has TWI_CLKDIV.CLKLO=17 and TWI_CLKDIV.CLKHI=8. Note that TWI_CLKDIV.CLKLO and TWI_CLKDIV.CLKHI add up to TWI_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. Falling edges are controlled by slew rate, and rising edges are governed by the *RC* time

constant formed by the pull-up resistor and the TWI_SCL capacitance. See the "Register Descriptions" section for more details.

Serial Data Signal (SDA)

This is a bidirectional signal on which serial data is transmitted or received depending on the direction of the transfer.

Internal Interface

The peripheral bus interface supports the transfer of 16-bit wide data and is used by the processor in the support of register and FIFO buffer reads and writes. The TWI internal interface is comprised of the blocks described below.

Register block. Contains all control and status bits and reflects what can be written or read as outlined by the programming model. Status bits can be updated by their respective functional blocks.

FIFO buffer. Configured as a1-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. This time base is used for filtering of data and timing events specified by the electrical data sheet (See the Philips specification), as well as for TWI_SCL clock generation.

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.

TWI Architectural Concepts

The TWI controller follows the transfer protocol of the Philips I^2C Bus Specification version 2.1 dated January 2000.

TWI Protocol

The following figure shows a simple complete transfer.

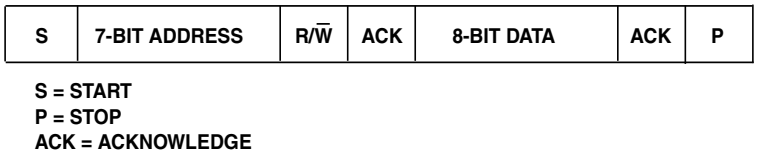


Figure 20-2: Data Transfer

To better understand the mapping of TWI controller register contents to a basic transfer, The following figure details the same transfer from the figure above noting the corresponding TWI controller bit names. In this illustration, the TWI controller successfully transmits one byte of data. The slave has acknowledged both address and data.

Figure 20-3: 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 has been initiated. 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. This is shown in the figure below.

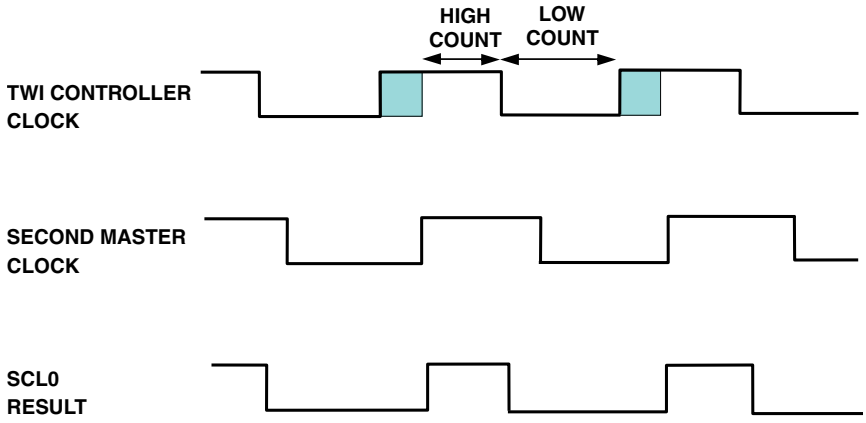


Figure 20-4: Clock Synchronization

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, allowing the external pull-up resistor to pull the `TWI_SCL` signal high, and the clock synchronization logic enters into a delay mode

(shaded area) until the TWI_SCL signal is detected at logic 1 level. At this time 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. This is shown in the figure below.

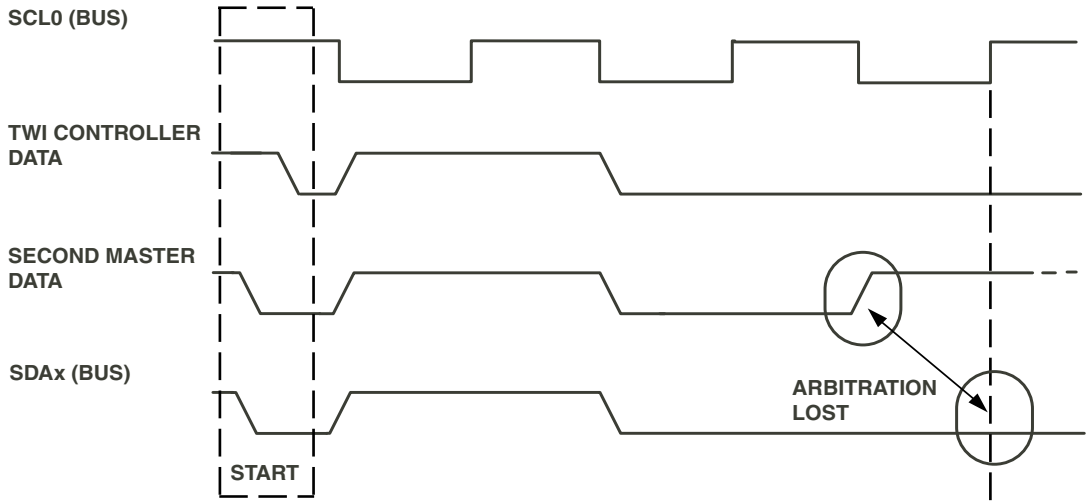


Figure 20-5: Bus Arbitration

The TWI controller monitors the serial data bus (SDA) while the TWI_SCL signal is high and if the TWI_SDA signal is determined to be an active logic 0 level while the TWI controller's data is a logic 1 level, the TWI controller has lost arbitration and stops generating the clock and data signals. Note that 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 with the exception of repeated start combined transfers, as shown in the figure below.

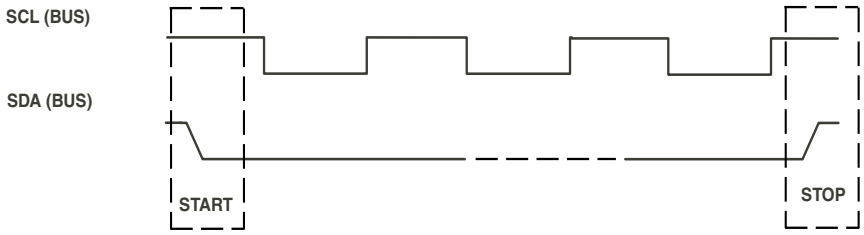


Figure 20-6: Start and Stop Conditions

The TWI controller's special case start and stop conditions 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_ISTAT.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_ISTAT.SCOMP) and indicates a slave transfer error (TWI_ISTAT.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 (as if data transfer count had been reached).

General Call Support

The TWI controller always decodes and acknowledges a general call address if it is enabled as a slave and if general call is enabled. General call addressing (0x00) is configured using the TWI_SLVCTL.GEN bit and only when the TWI controller is a slave-receiver.

If the data associated with the transfer is to be NAK'ed, the TWI_SLVCTL.NAK bit can be set. If the TWI controller is to issue a general call as a master-transmitter the appropriate address (TWI_MSTRADDR register) and transfer direction (TWI_MSTRCTL.DIR bit) can be set along with loading transmit FIFO data.

NOTE: The byte following the General Call address usually defines what action needs to be taken by the slaves in response to the call. The command in the second byte is interpreted based on the value of its LSB. For a TWI slave device, this is not applicable, and 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. It is the electrical specifications and timing that are most affected. When fast mode is enabled (FAST) timing is modified to meet the electrical requirements as described below.

- 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*. These are described in the following sections.

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 following 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.

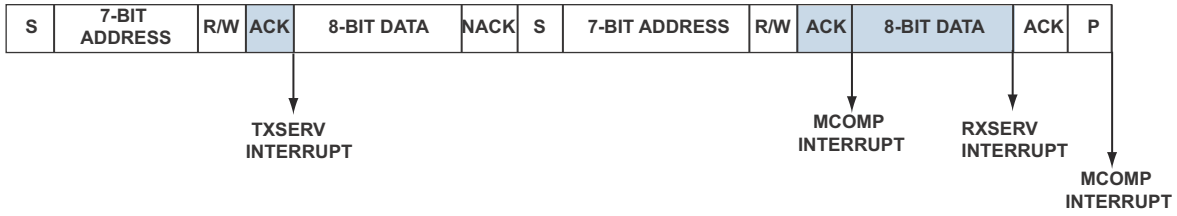


Figure 20-7: Repeated Start Followed by Data Receive

The following tasks are performed at each interrupt.

- Transmit FIFO service (`TWI_ISTAT.TXSERV`) interrupt. This interrupt is generated due to a FIFO access. Since this is the last byte of this transfer, the `TWI_FIFOSTAT` register indicates 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 will be a data receive.
- Master transfer complete (`TWI_ISTAT.MCOMP`) interrupt. This interrupt is generated when all data has been transferred (`TWI_MSTRCTL.DCNT` bit field=0). If no errors are generated, a start condition is initiated. 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_ISTAT.RXSERV`) interrupt. This interrupt is generated due to the arrival of a byte in the receive FIFO. Simple data handling is all that is required.
- Master transfer complete (`TWI_ISTAT.MCOMP`) interrupt. The transfer is complete.

Receive Transmit Repeated Start

The following 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.

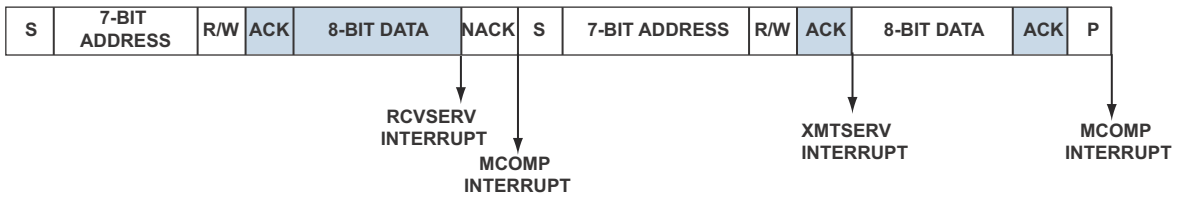


Figure 20-8: Repeated Start Data Receive Followed by Data Transmit

The tasks performed at each interrupt are:

- Receive FIFO service (`TWI_ISTAT.RXSERV`) interrupt. This interrupt 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 will be a data transmit.
- Master transfer complete (`TWI_ISTAT.MCOMP`) interrupt. This interrupt has occurred due to the completion of the data receive transfer. If no errors were generated, a start condition is initiated. 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_ISTAT.TXSERV`) interrupt. This interrupt is generated due to a FIFO access. Simple data handling is all that is required.
- Master transfer complete (`TWI_ISTAT.MCOMP`) interrupt. The transfer is complete.

NOTE: There is no timing constraint to meet the above 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 I ^{2}C clock while waiting to service interrupts. Stretching is done automatically by the hardware and no programming is required. 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 is generated the instant the transmit FIFO becomes empty. At this time, the most recent byte begins transmission. If the TWI_ISTAT.TXSERV interrupt is not serviced, the concluding acknowledge phase of the transfer is stretched.

Stretching of the clock continues until new data bytes are written to the transmit FIFO (TWI_TXDATA8 or TWI_TXDATA16registers). No other action is required to release the clock and continue the transmission. This behavior continues until the transmission is complete (TWI_MSTRCTL.DCNT=0) at which time the transmission is concluded (TWI_ISTAT.MCOMP) as shown in the following figure and table.

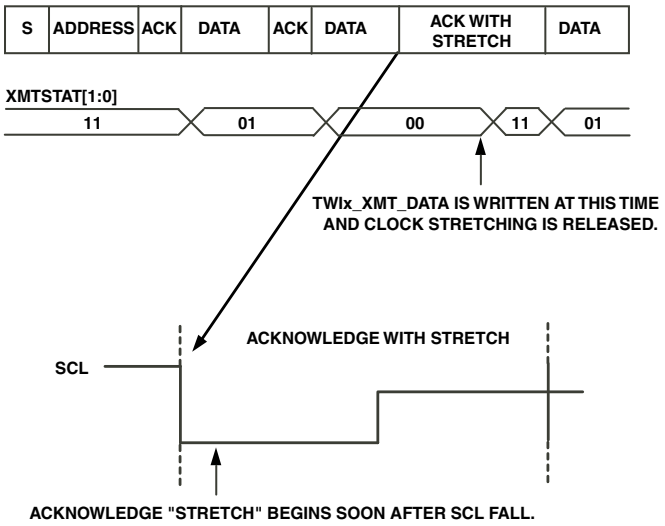


Figure 20-9: Clock Stretching during FIFO Underflow

TWI Controller	Processor
Interrupt: XMTSERV – Transmit FIFO buffer is empty.	Acknowledge: Clear interrupt source bits. Write transmit FIFO buffer.
...	...
Interrupt: MCOMP – Master transmit complete (DCNT= 0x00).	Acknowledge: Clear interrupt source bits.

Clock Stretching During FIFO Overflow

During a master mode receive, an interrupt is generated at the instant the receive FIFO becomes full. It is during the acknowledge phase of this received byte that clock stretching begins. No attempt is made to initiate the reception of an additional byte. Stretching of the clock continues until the data bytes previously received are read from the receive FIFO buffer (TWI_RXDATA8 or TWI_RXDATA16registers). No other action is required to release the clock and continue the reception of data. This behavior continues until the recep-

tion is complete (TWI_MSTRCTL.DCNT=0) at which time the reception is concluded (TWI_ISTAT.MCOMP) as shown in the following figure and table.

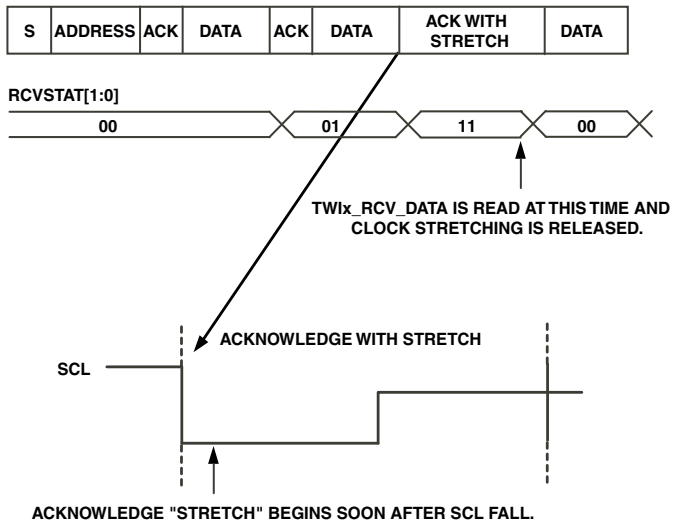


Figure 20-10: Clock Stretching During FIFO Overflow

TWI Controller	Processor
Interrupt: RCVSERV – Receive FIFO buffer is full.	Acknowledge: Clear interrupt source bits. Read receive FIFO buffer.
...	...
Acknowledge: Clear interrupt source bits.	Interrupt: MCOMP – Master receive complete.

Clock Stretching During Repeated Start

The repeated start feature in I^2C protocol requires a transition between two subsequent transfers. With the use of clock stretching, the task of managing transitions becomes simpler and becomes common to all transfer types.

Once an initial TWI master transfer has completed (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 above applies here. With no system related timing constraints the subsequent transfer (receive or transmit) is setup and activated. This sequence can be repeated as many times as required to string a series of repeated start transfers together. This is shown in the following figure and table.

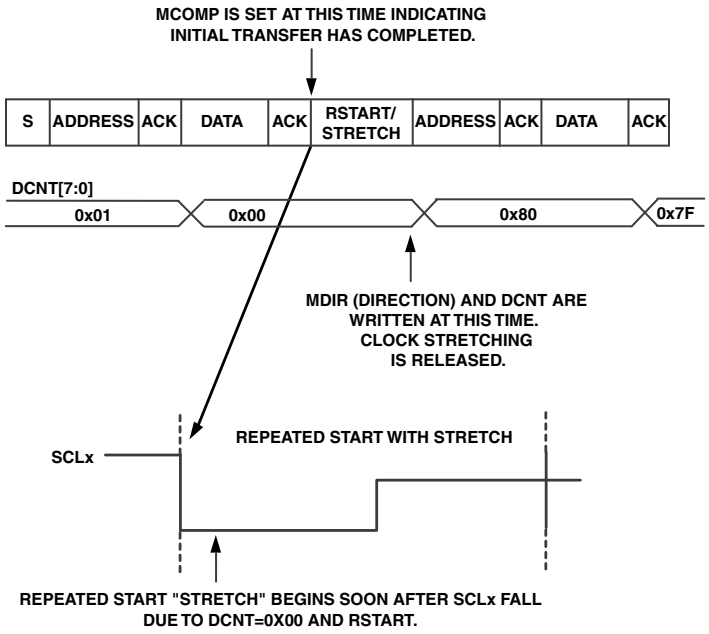


Figure 20-11: Clock Stretching during Repeated Start Condition

TWI Controller	Processor
Interrupt: MCOMP – Initial transmit has completed and DCNT = 0x00. Note: transfer in progress, RSTART previously set.	Acknowledge: Clear interrupt source bits. Write TWIx_MASTER_CTL, setting MDIR (receive), clearing RSTART, and setting new DCNT value (nonzero).
Interrupt: RCVSERV – Receive FIFO is full.	Acknowledge: Clear interrupt source bits. Read receive FIFO buffer.
...	...
Interrupt: MCOMP – Master receive complete	Acknowledge: Clear interrupt 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.

The following sections provide general setup, and master and slave mode programming steps.

General Setup

General setup refers to register writes that are required for both slave mode and master mode operations.

General setup should be performed before either the master or slave enable bits are set.

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_{SCLK}/10 MHz. All values should be rounded up to the next whole number.
3. Set the TWI_CTL.EN bit to enable the controller.

RESULT:

Once the TWI controller is enabled a bus busy condition may be detected. This condition should clear 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. This is reflected in the following setup.

1. Program the TWI_SLVADDR register. The appropriate 7 bits are used in determining a match during the address phase of the transfer.
2. Program the TWI_TXDATA8.VALUE or TWI_TXDATA16 registers. These are the initial data values to be transmitted in the event the slave is addressed and a transmit is required. This is an optional step. If no data is written and the slave is addressed and a transmit is required, the serial clock (TWI_SCL) is stretched and an interrupt is generated until data is written to the transmit FIFO.
3. Program the TWI_IMSK register. Enable bits are associated with the desired interrupt sources. As an example, programming the value 0x000F results in an interrupt output to the processor in the event that a valid address match is detected, a valid slave transfer completes, a slave transfer has an error, or a subsequent transfer has begun yet the previous transfer has not been serviced.
4. Program the TWI_SLVCTL register. This prepares and enables slave mode operation. As an example, programming the value 0x0005 enables slave mode operation, requires 7-bit addressing, and indicates that data in the transmit FIFO buffer is intended for slave mode transmission.

RESULT:

The following table and flow diagram shows what the interaction between the TWI controller and the processor might look like using this example.

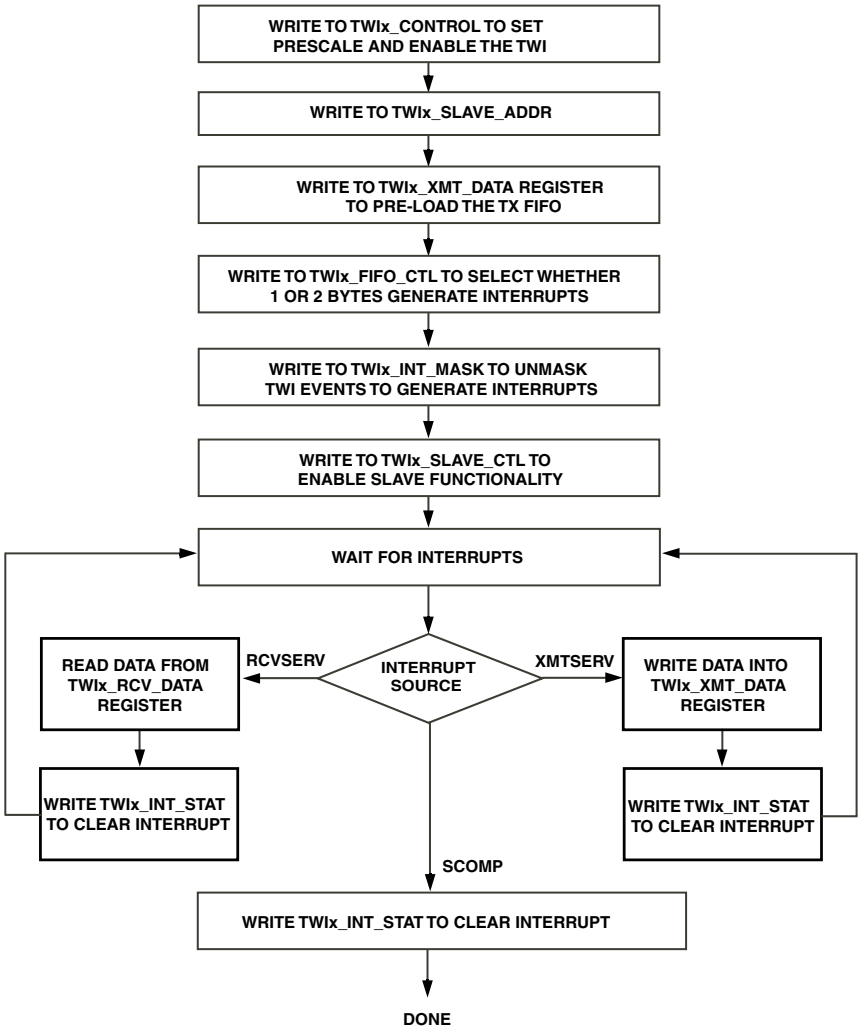


Figure 20-12: TWI Slave Mode Program Flow

Table 20-3: Slave Mode Interaction

TWI Controller	Processor
Interrupt: SINIT – Slave transfer in progress.	Acknowledge: Clear interrupt source bits.
Interrupt: RCVSERV – Receive buffer is full.	Acknowledge: Clear interrupt source bits. Read TWIx_FIFO_STAT. Read receive FIFO buffer.
...	...
Interrupt: SCOMP – Slave transfer complete.	Acknowledge: Clear interrupt source bits. Read receive FIFO buffer.

Master Mode Program Flow

The following figure shows the program for the TWI in master mode.

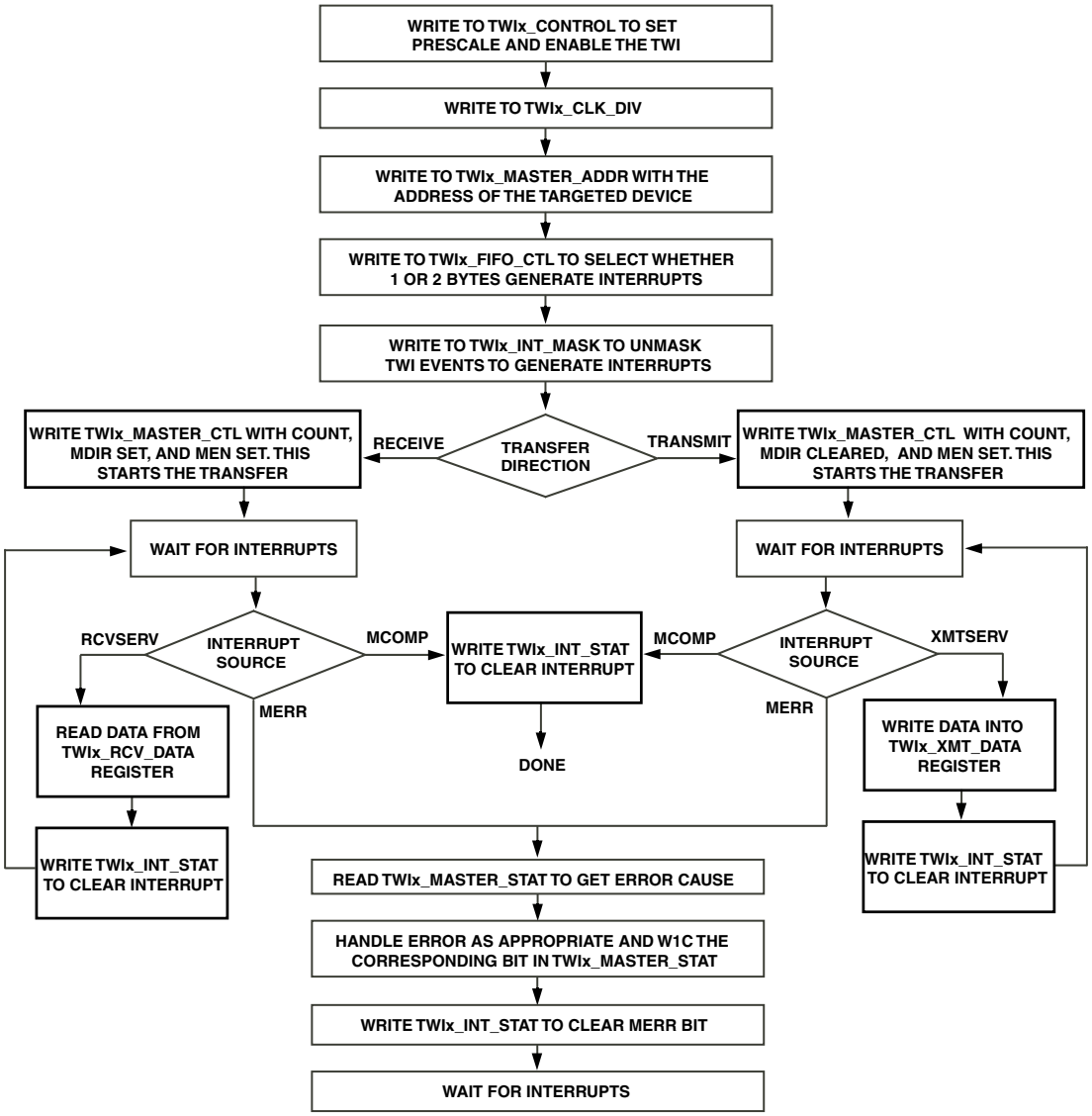


Figure 20-13: Master Mode Program Flow

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 are given separately in following sections. The clock setup programming step listed here is common to both transfer types.

1. Program the `TWI_CLKDIV` register to define the minimum clock high duration and minimum clock low duration.

RESULT:

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 TWI_SCL. Falling edges are controlled by the slew rate, and rising edges are governed by the RC time constant formed by the pull-up resistor and the SCL capacitance. See the "Register Descriptions" section for more details.

Master Mode Transmit

Follow these programming steps for a single master mode transmit:

1. Program the TWI_MSTRADDR register. This defines the address transmitted during the address phase of the transfer.
2. Program the TWI_TXDATA8 or TWI_TXDATA16 register. This is the initial data transmitted. It is considered 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. Indicate if the transmit FIFO buffer interrupts should occur with each byte transmitted (8-bits) or with each two bytes transmitted (16-bits).
4. Program the TWI_IMSK register. Enable the bits associated with the desired interrupt sources. As an example, programming the value 0x0030 results in an interrupt output to the processor in the event that the master transfer completes, and the master transfer has an error.
5. Program the TWI_MSTRCTL register. This prepares and enables master mode operation. As an 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.

RESULT:

The following table shows what the interaction between the TWI controller and the processor might look like using this example.

Table 20-4: Master Mode Transmit Setup Interaction

TWI Controller	Processor
Interrupt: XMTSERV – Transmit buffer is empty.	Acknowledge: Clear interrupt source bits. Write transmit FIFO buffer.
...	...
Interrupt: MCOMP – Master transfer complete.	Acknowledge: Clear interrupt source bits.

Master Mode Receive

Follow these programming steps for a single master mode receive.

1. Program the TWI_MSTRADDR register. This defines the address transmitted during the address phase of the transfer.
2. Program the TWI_FIFOCTL register. Indicate if receive FIFO buffer interrupts should occur with each byte received (8-bits) or with each two bytes received (16-bits).
3. Program the TWI_IMSK register. Enable bits associated with the desired interrupt sources. For example, programming the value 0x0030 results in an interrupt output to the processor in the event that the master transfer completes, and the master transfer has an error.
4. Program the TWI_MSTRCTL register. This prepares and enables master mode operation. As an 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.

RESULT:

The following table shows what the interaction between the TWI controller and the processor might look like using this example.

Table 20-5: Master Mode Receive Setup Interaction

TWI Controller	Processor
Interrupt: RCVSERV – Receive buffer is full.	Acknowledge: Clear interrupt source bits. Read receive FIFO buffer.
...	...
Interrupt: MCOMP – Master transfer complete.	Acknowledge: Clear interrupt source bits. Read receive FIFO buffer.

NOTE: After the TWI_DCNT bit is decremented to zero, the TWI Master device sends a NAK to indicate to the slave transmitter that the bus should be released. This allows the master to send the STOP signal to terminate the transfer."

ADSP-BF60x TWI Register Descriptions

2-Wire Interface (TWI) contains the following registers.

Table 20-6: ADSP-BF60x TWI Register List

Name	Description
TWI_CLKDIV	SCL Clock Divider Register
TWI_CTL	Control Register
TWI_SLVCTL	Slave Mode Control Register
TWI_SLVSTAT	Slave Mode Status Register
TWI_SLVADDR	Slave Mode Address Register
TWI_MSTRCTL	Master Mode Control Registers
TWI_MSTRSTAT	Master Mode Status Register
TWI_MSTRADDR	Master Mode Address Register
TWI_ISTAT	Interrupt Status Register
TWI_IMSK	Interrupt Mask Register
TWI_FIFOCTL	FIFO Control Register
TWI_FIFOSTAT	FIFO Status Register
TWI_TXDATA8	Tx Data Single-Byte Register
TWI_TXDATA16	Tx Data Double-Byte Register
TWI_RXDATA8	Rx Data Single-Byte Register
TWI_RXDATA16	Rx Data Double-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.

CLKDIV = TWI SCL period / 10 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):

CLKDIV = 2500 ns / 100 ns = 25

For an SCL with a 30% duty cycle, use `TWI_CLKDIV.CLKLO` = 17 and `TWI_CLKDIV.CLKHI` = 8.

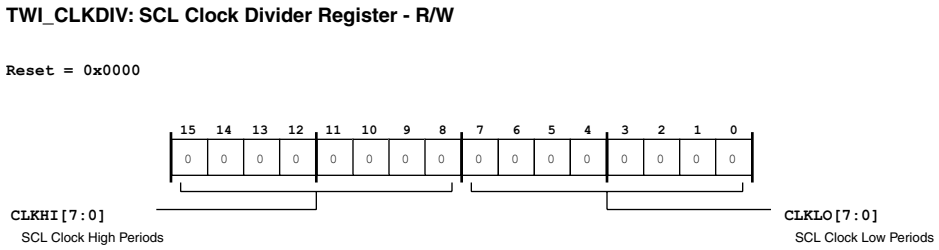


Figure 20-14: TWI_CLKDIV Register Diagram

Table 20-7: TWI_CLKDIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	CLKHI	SCL Clock High Periods. The `TWI_CLKDIV.CLKHI` 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 `TWI_CLKDIV.CLKLO` 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.

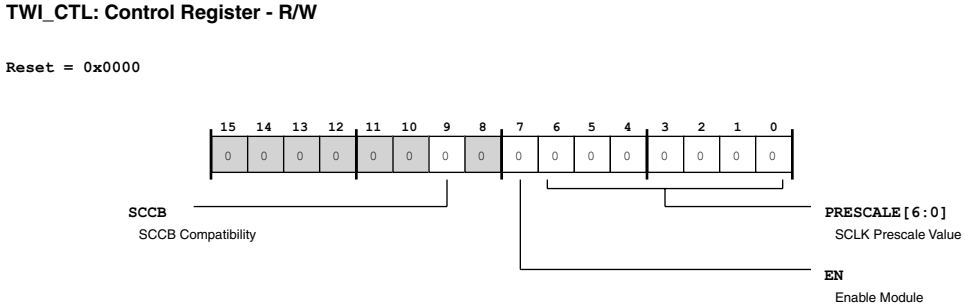


Figure 20-15: TWI_CTL Register Diagram

Table 20-8: TWI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	SCCB	SCCB Compatibility. The TWI_CTL.SCCB enables SCCB compatible operation for the TWI. SCCB compatibility is an optional feature and should not be used in an I^2C 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 TWI_CTL.EN enables TWI controller operation for either master and/or slave mode of operation. It is recommended that this bit be set at the time TWI_CTL.PRESCALE is initialized and remain set. This method guarantees accurate operation of bus busy detection logic.	
		0	Disable
		1	Enable
6:0 (R/W)	PRESCALE	SCLK Prescale Value. The TWI_CTL.PRESCALE holds the pre-scaled value for the TWI internal time reference. This reference is derived from SCLK according to the formula: TWI_CTL.PRESCALE = f_{SCLK}/10MHz The TWI_CTL.PRESCALE specifies the number of system clock (SCLK) periods used in the generation of one internal time reference. The value of TWI_CTL.PRESCALE must be set to create an internal time reference with a period of 10 MHz. It is represented as a 7-bit binary value.	

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.

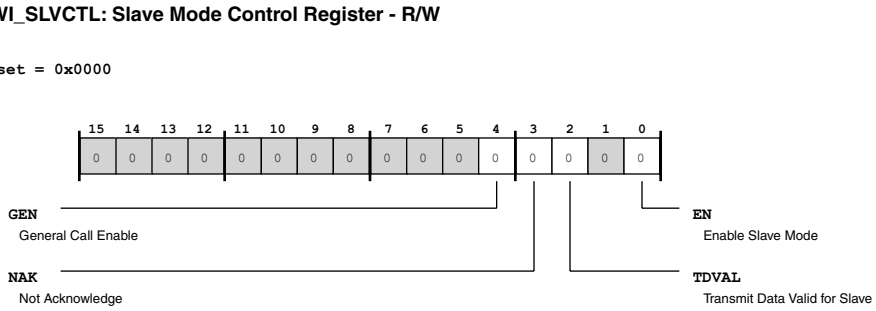


Figure 20-16: TWI_SLVCTL Register Diagram

Table 20-9: TWI_SLVCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	GEN	General Call Enable. The TWI_SLVCTL.GEN 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 TWI_SLVCTL.NAK 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

Table 20-9: TWI_SLVCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	TDVAL	Transmit Data Valid for Slave. The TWI_SLVCTL.TDVAL selects whether the data in the transmit FIFO is available (valid) for slave transmission (TWI_SLVCTL.TDVAL set). If the FIFO data is not available (invalid) for slave transmission (TWI_SLVCTL.TDVAL 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
0 (R/W)	EN	Enable Slave Mode. The TWI_SLVCTL.EN 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 TWI_SLVCTL.EN 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 interrupts. Master mode operation does not affect slave mode status bits.

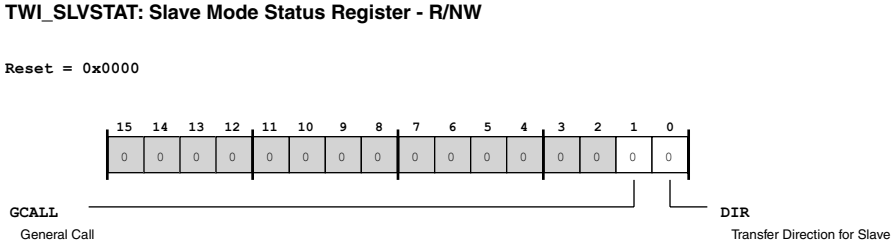


Figure 20-17: TWI_SLVSTAT Register Diagram

Table 20-10: TWI_SLVSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	GCALL	General Call. The TWI_SLVSTAT.GCALL 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 (TWI_SLVCTL.EN =0).	
		0	Not a General Call Address
		1	General Call Address
0 (R/NW)	DIR	Transfer Direction for Slave. The TWI_SLVSTAT.DIR 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 (TWI_SLVCTL.EN =0).	
		0	Slave Receive
		1	Slave Transmit

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.

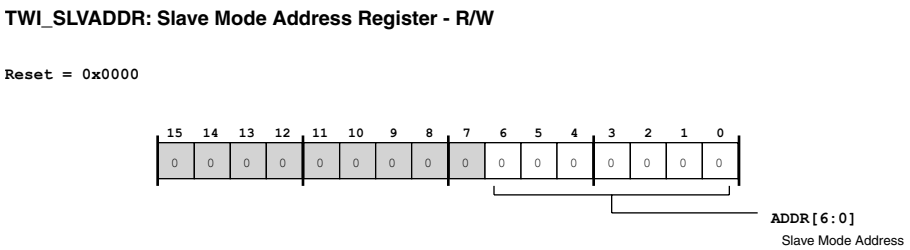


Figure 20-18: TWI_SLVADDR Register Diagram

Table 20-11: TWI_SLVADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	ADDR	Slave 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.

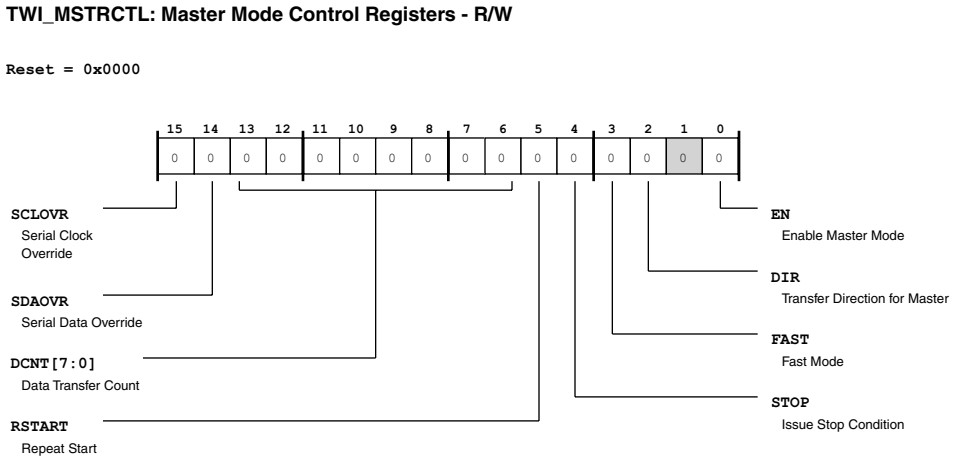


Figure 20-19: TWI_MSTRCTL Register Diagram

Table 20-12: TWI_MSTRCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	SCLOVR	Serial Clock Override. The TWI_MSTRCTL.SCLOVR provides direct control of the serial clock line when required. Normal master and slave mode operation should not require override operation. When TWI_MSTRCTL.SCLOVR 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 TWI_MSTRCTL.SCLOVR 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

Table 20-12: TWI_MSTRCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	SDAOVR	Serial Data Override. The TWI_MSTRCTL.SDAOVR provides direct control of the serial data line when required. Normal master and slave mode operation should not require override operation. When TWI_MSTRCTL.SDAOVR 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 TWI_MSTRCTL.SDAOVR is cleared, the TWI permits normal serial data operation.	
		0	Permit Normal SDA Operation
		1	Override Normal SDA Operation
13:6 (R/W)	DCNT	Data Transfer Count. The TWI_MSTRCTL.DCNT indicates the number of data bytes to transfer. As each data word is transferred, the TWI decrements this counter. When TWI_MSTRCTL.DCNT decrements to 0, a stop condition is generated. Setting TWI_MSTRCTL.DCNT to 0xFF disables the counter. In this transfer mode, data continues to be transferred until it is concluded by setting the TWI_MSTRCTL.STOP bit. In the event a master transmit is aborted due to a slave data NAK, the value of TWI_MSTRCTL.DCNT equals the number of bytes not sent. The byte which was NAK'ed by the slave is counted as a sent byte.	
5 (R/W)	RSTART	Repeat Start. The TWI_MSTRCTL.RSTART enables the TWI to issue a repeat start condition at the conclusion of the current transfer (TWI_MSTRCTL.DCNT =0) 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 (TWI_MSTRCTL.EN) does not self clear on a repeat start.	
		0	Disable Repeat Start
		1	Enable Repeat Start
4 (R/W)	STOP	Issue Stop Condition. The TWI_MSTRCTL.STOP 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 TWI_IMSK is updated along with any associated status bits.	
		0	Permit Normal Operation
		1	Issue Stop

Table 20-12: TWI_MSTRCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	FAST	Fast Mode. The TWI_MSTRCTL.FAST 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.	
		0	Select Standard Mode
		1	Select Fast Mode
2 (R/W)	DIR	Transfer Direction for Master. The TWI_MSTRCTL.DIR selects the transfer direction for the TWI as master initiated receive or transmit.	
		0	Master Transmit
		1	Master Receive
0 (R/W)	EN	Enable Master Mode. The TWI_MSTRCTL.EN enables master mode functionality. A start condition is generated if the bus is idle. This bit self clears at the completion of a transfer, including transfers terminated due to errors. If disabled (bit cleared) 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.	
		0	Disable
		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 interrupts, 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.

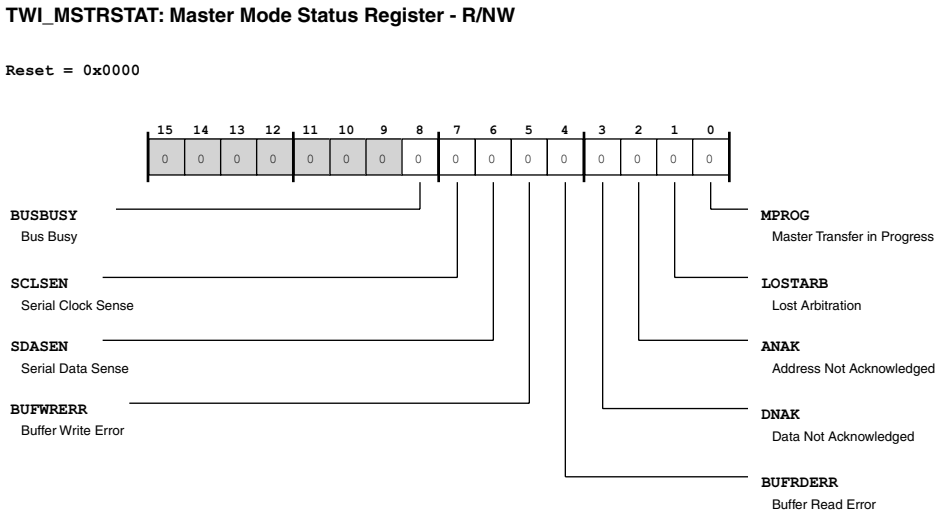


Figure 20-20: TWI_MSTRSTAT Register Diagram

Table 20-13: TWI_MSTRSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	BUSBUSY	Bus Busy. The TWI_MSTRSTAT.BUSBUSY 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 20-13: TWI_MSTRSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	SCLSEN	Serial Clock Sense. The TWI_MSTRSTAT.SCLSEN 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 TWI_MSTRSTAT.SDASEN 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 TWI_MSTRSTAT.BUFWRERR 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

Table 20-13: TWI_MSTRSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W1C)	BUFRDERR	Buffer Read Error. The TWI_MSTRSTAT.BUFRDERR 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	Buffer Read Error
3 (R/W1C)	DNAK	Data Not Acknowledged. The TWI_MSTRSTAT.DNAK 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 TWI_MSTRSTAT.ANAK 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 TWI_MSTRSTAT.LOSTARB 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 TWI_MSTRSTAT.MPROG indicates whether or not a master transfer is in progress. If clear (TWI_MSTRSTAT.MPROG =0), 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

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.

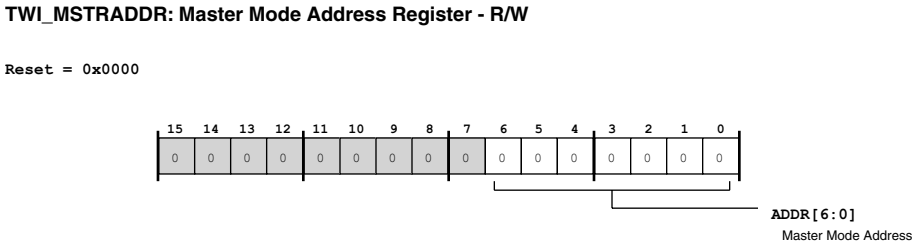


Figure 20-21: TWI_MSTRADDR Register Diagram

Table 20-14: TWI_MSTRADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	ADDR	Master Mode Address.

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.

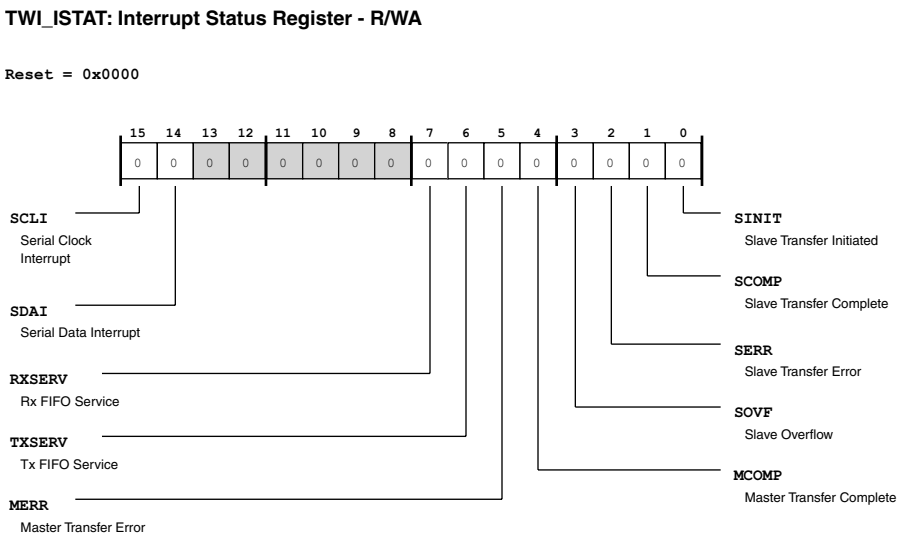


Figure 20-22: TWI_ISTAT Register Diagram

Table 20-15: TWI_ISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	SCLI	Serial Clock Interrupt. If the TWI is enabled (TWI_CTL.EN), SCLI is set on a high-to-low transition of the serial clock pin (SCLx). Normally, this bit is not required for I^2C bus transfers. It will be initially set on an I^2C transfer and does not require clearing.	
		0	No Interrupt No transition was detected on the SCLx pin.
		1	Interrupt Detected A high-to-low transition was detected on the SCLx pin. This bit is W1C.
14 (R/W1C)	SDAI	Serial Data Interrupt. If the TWI is enabled (TWI_CTL.EN), SDAI is set on a high-to-low transition of the serial data pin (SDAx). Normally, this bit is not required for I^2C bus transfers. It will be initially set on an I^2C transfer and does not require clearing.	
		0	No Interrupt No transition was detected on the SDAx pin.
		1	Interrupt Detected A high-to-low transition was detected on the SDAx pin. This bit is W1C.
7 (R/W1C)	RXSERV	Rx FIFO Service. If TWI_FIFOCTL.RXILEN =0, the TWI_ISTAT.RXSERV is set each time the TWI_FIFOSTAT.RXSTAT field is updated to either 01 or 11. If TWI_FIFOCTL.RXILEN =1, the TWI_ISTAT.RXSERV is set each time TWI_FIFOSTAT.RXSTAT is updated to 11.	
		0	No Interrupt The FIFO does not require servicing, or the TWI_FIFOSTAT.RXSTAT 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.

Table 20-15: TWI_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1C)	TXSERV	Tx FIFO Service. If TWI_FIFOCTL.TXILEN =0, the TWI_ISTAT.TXSERV is set each time the TWI_FIFOSTAT.TXSTAT field is updated to either 01 or 00. If TWI_FIFOCTL.TXILEN =1, the TWI_ISTAT.TXSERV is set each time TWI_FIFOSTAT.TXSTAT is updated to 00.	
		0	No Interrupt FIFO does not require servicing, or the TWI_FIFOSTAT.TXSTAT 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 TWI_ISTAT.MERR indicates that a master error has occurred. The conditions surrounding the error are indicated by the master status register (TWI_MSTRSTAT).	
		0	No Interrupt
		1	Interrupt Detected
4 (R/W1C)	MCOMP	Master Transfer Complete. The TWI_ISTAT.MCOMP 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 TWI_ISTAT.SOVF indicates that the TWI_ISTAT.SCOMP 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 20-15: TWI_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	SERR	Slave Transfer Error. The TWI_ISTAT.SERR 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 TWI_ISTAT.SCOMP 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 TWI_ISTAT.SINIT 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.

Interrupt Mask Register

The TWI_IMSK enables interrupt sources to assert the interrupt output. Each mask bit corresponds with one interrupt source bit in TWI_ISTAT. Reading and writing TWI_IMSK does not affect the contents of the TWI_ISTAT.

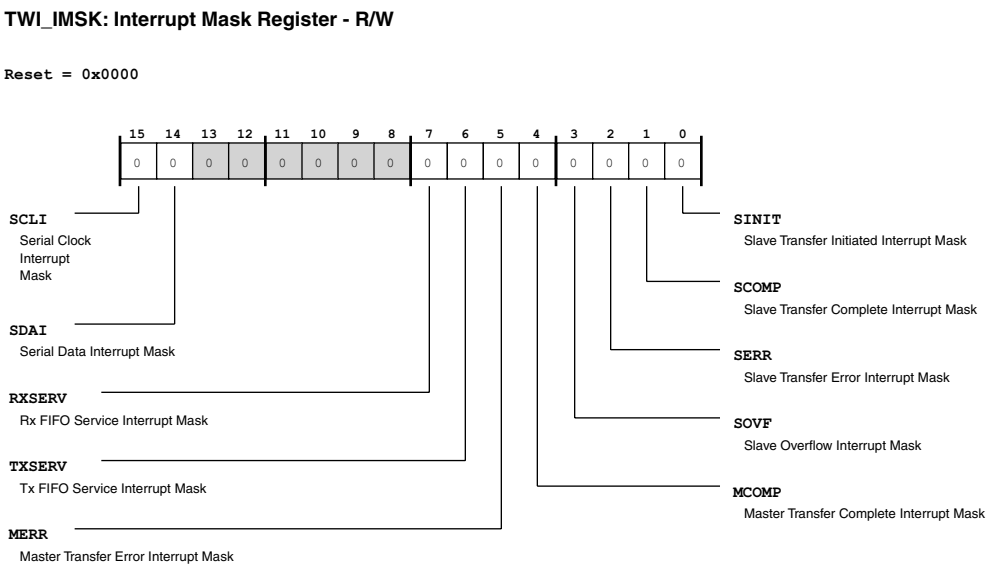


Figure 20-23: TWI_IMSK Register Diagram

Table 20-16: 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
5 (R/W)	MERR	Master Transfer Error Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt

Table 20-16: TWI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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

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.

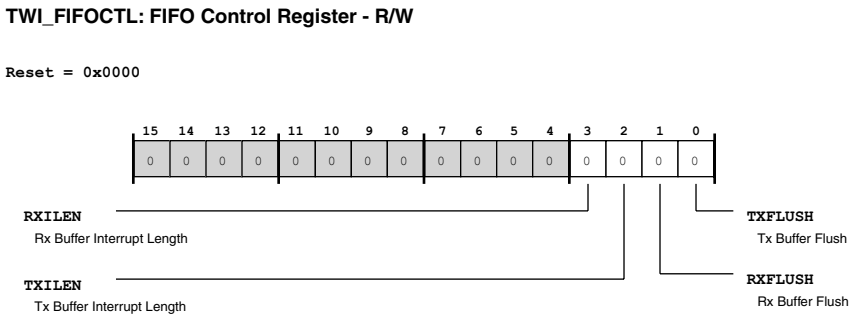


Figure 20-24: TWI_FIFOCTL Register Diagram

Table 20-17: TWI_FIFOCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	RXILEN	Rx Buffer Interrupt Length. The TWI_FIFOCTL.RXILEN 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 TWI_FIFOSTAT.RXSTAT.	
		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 TWI_FIFOCTL.TXILEN 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 TWI_FIFOSTAT.TXSTAT.	
		0	TXSERVI on 1 Byte of FIFO Empty
		1	TXSERVI on 2 Bytes of FIFO Empty
1 (R/W)	RXFLUSH	Rx Buffer Flush. The TWI_FIFOCTL.RXFLUSH directs the TWI to flush the contents of the receive buffer and update TWI_FIFOSTAT.RXSTAT 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
0 (R/W)	TXFLUSH	Tx Buffer Flush. The TWI_FIFOCTL.TXFLUSH directs the TWI to flush the contents of the transmit buffer and update TWI_FIFOSTAT.TXSTAT 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.

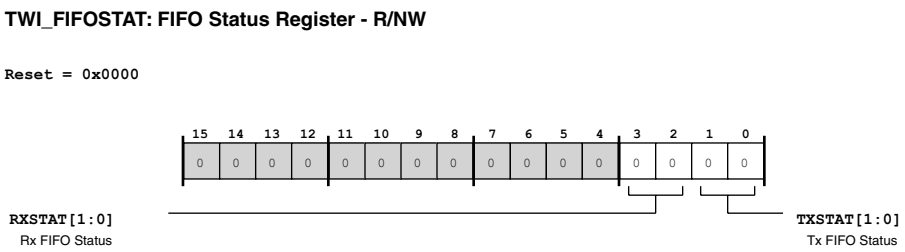


Figure 20-25: TWI_FIFOSTAT Register Diagram

Table 20-18: TWI_FIFOSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3:2 (R/NW)	RXSTAT	Rx FIFO Status. The read-only `TWI_FIFOSTAT.RXSTAT` 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.

Table 20-18: TWI_FIFOSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/NW)	TXSTAT	Tx FIFO Status. The read-only TWI_FIFOSTAT.TXSTAT 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.

Tx Data Single-Byte Register

The TWI_TXDATA8 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 TWI_TXDATA8 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.

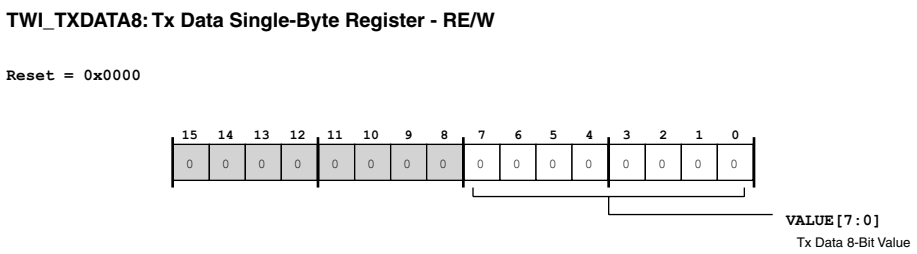


Figure 20-26: TWI_TXDATA8 Register Diagram

Table 20-19: TWI_TXDATA8 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W)	VALUE	Tx Data 8-Bit Value.

Tx Data Double-Byte Register

The TWI_TXDATA16 holds a 16-bit data value written into the FIFO buffer. To reduce interrupt 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.

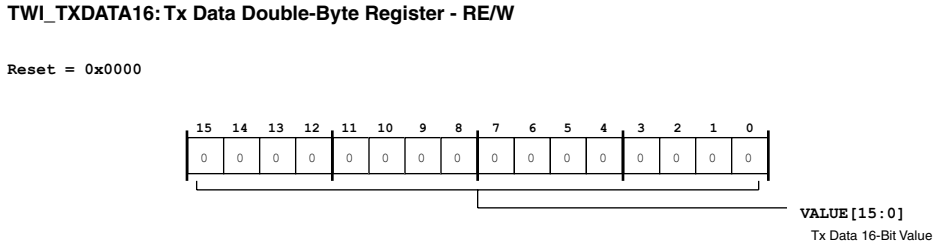


Figure 20-27: TWI_TXDATA16 Register Diagram

Table 20-20: TWI_TXDATA16 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R0/W)	VALUE	Tx 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.

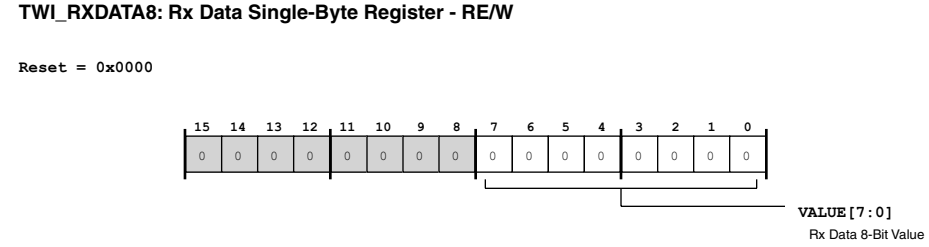


Figure 20-28: TWI_RXDATA8 Register Diagram

Table 20-21: TWI_RXDATA8 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W)	VALUE	Rx Data 8-Bit Value.

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.

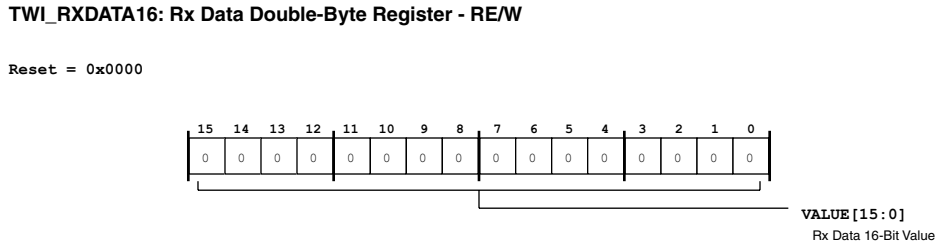


Figure 20-29: TWI_RXDATA16 Register Diagram

Table 20-22: TWI_RXDATA16 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R0/W)	VALUE	Rx Data 16-Bit Value.

21 Controller Area Network (CAN)

The processor contains a controller area network (CAN) module based on the CAN 2.0B (active) protocol. This protocol is an asynchronous communications protocol used in both industrial and automotive control systems. The CAN protocol is well suited for control applications because it can communicate reliably over a network and incorporates CRC checking, message error tracking, and fault node confinement.

NOTE: This document assumes familiarity with the CAN standard. For more information, refer to Version 2.0 of the CAN Specification from Robert Bosch GmbH.

CAN Features

Key features of the CAN module include:

- Conformity to the CAN 2.0B (active) standard
- Dedicated acceptance mask for each mailbox
- Support for data rates of up to 1M bit/s
- Support for standard (11-bit) and extended (29-bit) identifiers
- 32 mailboxes (8 transmit, 8 receive, 16 configurable)
- Data filtering (first 2 bytes) can be used for acceptance filtering (DeviceNet™ mode)
- Error status and warning registers
- Universal counter module
- Readable receive and transmit pin values
- Support for remote frames
- Active or passive network support
- Interrupts, including transmit/receive complete, error, and global
- Clock derived from *SCLK* through a programmable divider, eliminating the need for an additional crystal

CAN Functional Description

The following sections provide information on the functional operation of the CAN module. This section also provides listings of the CAN registers and interrupts.

ADSP-BF60x CAN Register List

The controller area network (CAN) module implements the CAN 2.0B (active) protocol. This protocol is an asynchronous communications protocol used in both industrial and automotive control systems. A set of registers govern CAN operations. For more information on CAN functionality, see the CAN register descriptions.

Table 21-1: ADSP-BF60x CAN Register List

Name	Description
CAN_MC1	Mailbox Configuration 1 Register
CAN_MD1	Mailbox Direction 1 Register
CAN_TRS1	Transmission Request Set 1 Register
CAN_TRR1	Transmission Request Reset 1 Register
CAN_TA1	Transmission Acknowledge 1 Register
CAN_AA1	Abort Acknowledge 1 Register
CAN_RMP1	Receive Message Pending 1 Register
CAN_RML1	Receive Message Lost 1 Register
CAN_MBTIF1	Mailbox Transmit Interrupt Flag 1 Register
CAN_MBRIF1	Mailbox Receive Interrupt Flag 1 Register
CAN_MBIM1	Mailbox Interrupt Mask 1 Register
CAN_RFH1	Remote Frame Handling 1 Register
CAN_OPSS1	Overwrite Protection/Single Shot Transmission 1 Register
CAN_MC2	Mailbox Configuration 2 Register
CAN_MD2	Mailbox Direction 2 Register

Table 21-1: ADSP-BF60x CAN Register List (Continued)

Name	Description
CAN_TRS2	Transmission Request Set 2 Register
CAN_TRR2	Transmission Request Reset 2 Register
CAN_TA2	Transmission Acknowledge 2 Register
CAN_AA2	Abort Acknowledge 2 Register
CAN_RMP2	Receive Message Pending 2 Register
CAN_RML2	Receive Message Lost 2 Register
CAN_MBTIF2	Mailbox Transmit Interrupt Flag 2 Register
CAN_MBRIF2	Mailbox Receive Interrupt Flag 2 Register
CAN_MBIM2	Mailbox Interrupt Mask 2 Register
CAN_RFH2	Remote Frame Handling 2 Register
CAN_OPSS2	Overwrite Protection/Single Shot Transmission 2 Register
CAN_CLK	Clock Register
CAN_TIMING	Timing Register
CAN_DBG	Debug Register
CAN_STAT	Status Register
CAN_CEC	Error Counter Register
CAN_GIS	Global CAN Interrupt Status Register
CAN_GIM	Global CAN Interrupt Mask Register
CAN_GIF	Global CAN Interrupt Flag Register
CAN_CTL	CAN Master Control Register
CAN_INT	Interrupt Pending Register
CAN_MBTD	Temporary Mailbox Disable Register

Table 21-1: ADSP-BF60x CAN Register List (Continued)

Name	Description
CAN_EWR	Error Counter Warning Level Register
CAN_ESR	Error Status Register
CAN_UCCNT	Universal Counter Register
CAN_UCRC	Universal Counter Reload/Capture Register
CAN_UCCNF	Universal Counter Configuration Mode Register
CAN_AMnnL	Acceptance Mask (L) Register
CAN_AMnnH	Acceptance Mask (H) Register
CAN_MBnn_DATA0	Mailbox Word 0 Register
CAN_MBnn_DATA1	Mailbox Word 1 Register
CAN_MBnn_DATA2	Mailbox Word 2 Register
CAN_MBnn_DATA3	Mailbox Word 3 Register
CAN_MBnn_LENGTH	Mailbox Length Register
CAN_MBnn_TIMESTAMP	Mailbox Timestamp Register
CAN_MBnn_ID0	Mailbox ID 0 Register
CAN_MBnn_ID1	Mailbox ID 1 Register

ADSP-BF60x CAN Interrupt List

Table 21-2: ADSP-BF60x CAN Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
CAN0 Receive	40		LEVEL
CAN0 Transmit	41		LEVEL
CAN0 Status	42		LEVEL

External Interface

The interface to the CAN bus is a simple two-wire line. The following figure shows a symbolic representation of the CAN transceiver interconnection. Typically, the processor's CAN_TX output and CAN_RX input pins are connected to an external CAN transceiver's CAN_TX and CAN_RX pins (respectively). The CAN_TX and CAN_RX pins operate with TTL levels and are appropriate for operation with CAN bus transceivers according to ISO/DIS 11898.

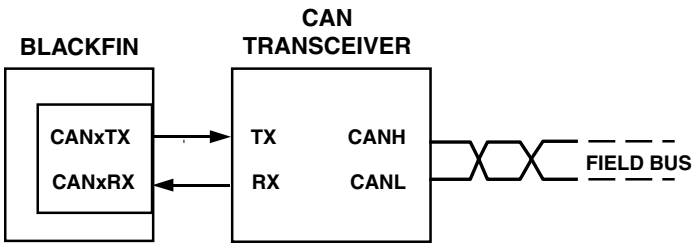


Figure 21-1: Representation of CAN Transceiver Interconnection

CAN data is defined to be either dominant (logic 0) or recessive (logic 1). The default state of the CAN_TX output is recessive.

ADSP-BF60x Specific External Interface

The CAN_TX and CAN_RX signals can be found on GPIO port G, pins PA_01 and PA_04 respectively. By default, these pins are in GPIO mode. The SPORT_2A and the timer signals are also multiplexed on this pin. To enable CAN functionality, the appropriate bits must be set in the PORT_FER and PORT_MUX registers.

The PA_04 pin (CAN_RX input pin) is also internally routed to the alternate capture input TACI2 of GP timer 2. This way, GP timer 2 can be used to auto-detect or adjust the bit rate on the CAN bus.

NOTE: The CAN pad does not support 5V operation.

Architectural Concepts

The full-CAN controller features 32 message buffers, which are called mailboxes. Eight mailboxes are dedicated for message transmission, eight are for reception, and 16 are programmable in direction.

The CAN module architecture is based around a 32-entry mailbox RAM. The mailbox is accessed sequentially by the CAN serial interface or the Blackfin core. Each mailbox consists of eight 16-bit control and data registers and two optional 16-bit acceptance mask registers, all of which must be configured before the mailbox itself is enabled.

Since the mailbox area (shown in the following figure) is implemented as RAM, the reset values of these registers are undefined. The data is divided into fields, which includes a message identifier, a time stamp, a byte count, up to 8 bytes of data, and several control bits

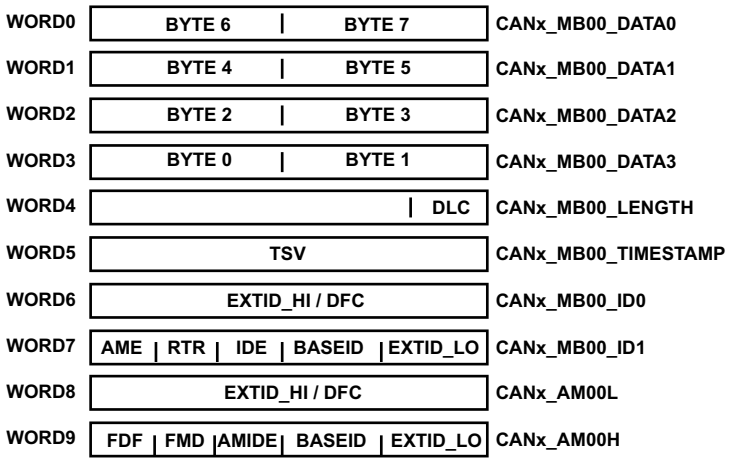


Figure 21-2: CAN Mailbox Area

The CAN mailbox identification (CAN_MBnn_ID0/1) register pair includes:

- The 29 bit identifier (base part CAN_AMnnH.BASEID plus extended part CAN_AMnnL.EXTID/CAN_AMnnH.EXTID)
- The acceptance mask enable bit (CAN_MBnn_ID1.AME)
- The remote transmission request bit (CAN_MBnn_ID1.RTR)
- The identifier extension bit (CAN_MBnn_ID1.IDE)

NOTE: Do not write to the identifier of a message object while the mailbox is enabled for the CAN module (the corresponding bit in CAN_MC1 is set).

The other mailbox area registers/bits are:

- The data length code bit (CAN_MBnn_LENGTH.DLC). The upper 12 bits of this register of each mailbox are marked as reserved. These 12 bits should always be set to 0.
- The mailbox word registers (CAN_MBnn_DATA0/1/2/3) supply up to eight bytes for the data field, sent MSB first from based on the number of bytes defined in the CAN_MBnn_LENGTH.DLC bit. For example, if only one byte is transmitted or received (CAN_MBnn_LENGTH.DLC=1), then it is stored in the most significant byte of the CAN_MBnn_DATA3 register.
- The time stamp value bits (CAN_MBnn_TIMESTAMP.TSV).

The final registers in the mailbox area are the acceptance mask registers (CAN_AMnnH and CAN_AMnnL). The acceptance mask is enabled when the CAN_MBnn_ID1.AME bit is set.

The *filtering on data field* option can be enabled by setting the CAN_CTL.DNM and CAN_AMnnH.FDF bits. When enabled, the CAN_MBnn_ID0.EXTID[15:0] bits are reused as acceptance code (DFC) for the data field filtering.

Block Diagram

The following figure shows a block diagram of the CAN module.

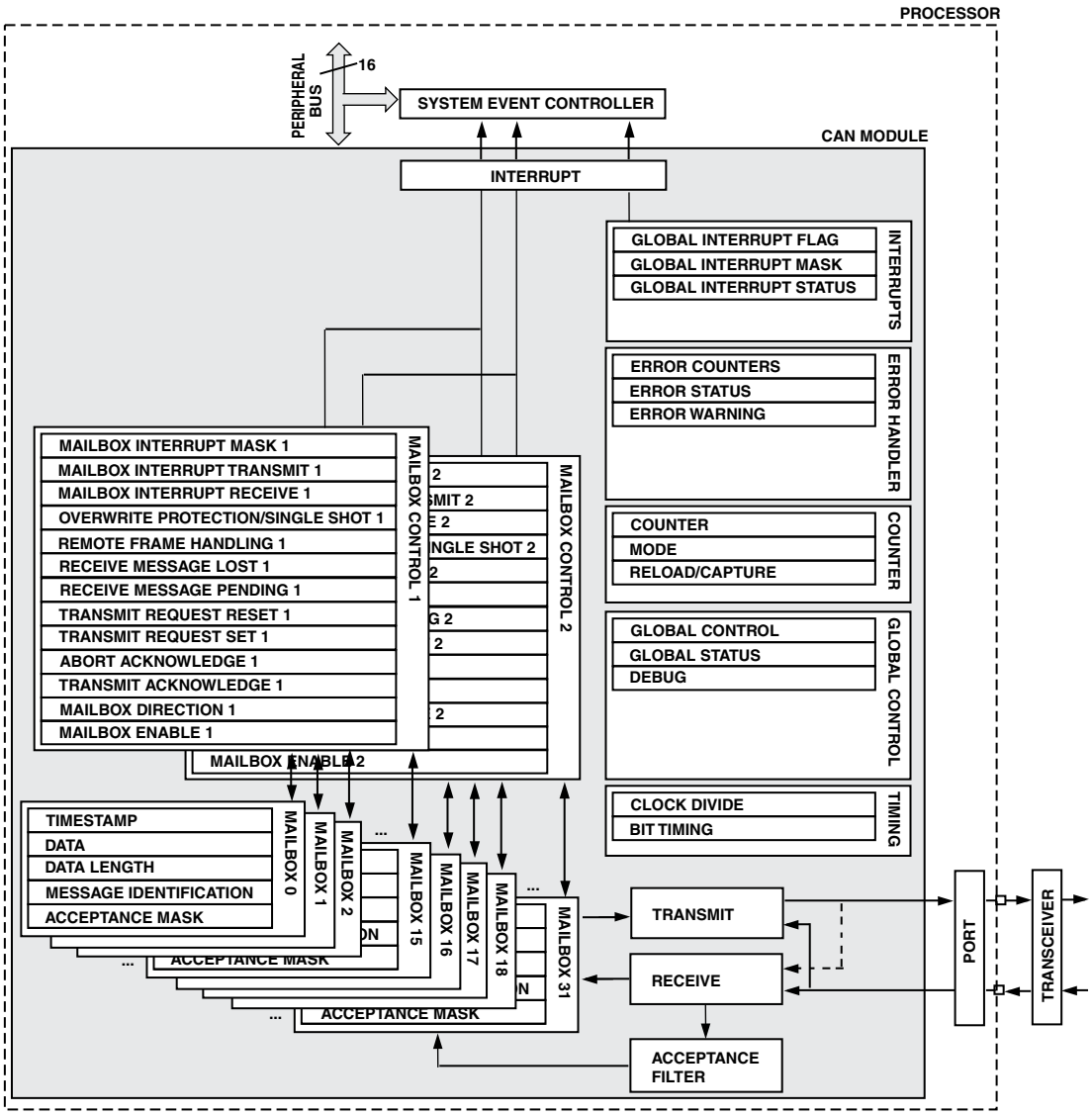


Figure 21-3: CAN Controller Block Diagram

Mailbox Control

Mailbox control memory-mapped registers (MMRs) function as control and status registers for the 32 mailboxes. Each bit in these registers represents one specific mailbox. Since CAN MMRs are all 16 bits wide, pairs of registers are required to manage certain functionality for all 32 individual mailboxes. Mailboxes 0–15 are configured/monitored in registers with a suffix of 1. Similarly, mailboxes 16–31 use the same named register with a suffix of 2. For example, the CAN mailbox direction registers (`CAN_MD1`/ `CAN_MD2`) control mailboxes as shown in the figure below.The Mailbox Control registers are shown in *ADSP-BF60x CAN Register List*.

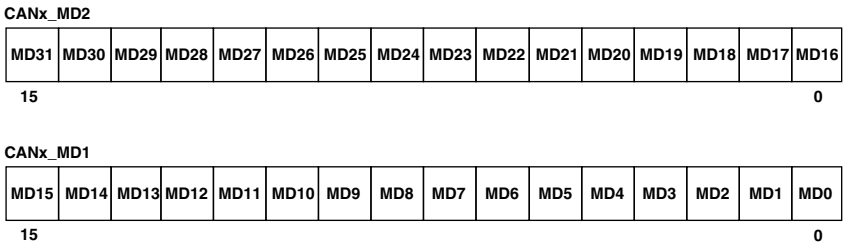


Figure 21-4: CAN Mailbox Register Pair

Since mailboxes 24–31 support transmit operation only and mailboxes 0–7 are receive-only mailboxes, the lower eight bits in the 1 registers and the upper eight bits in the 2 registers are sometimes reserved or are restricted in their use.

Protocol Fundamentals

Although the CAN_RX and CAN_TX pins are TTL-compliant signals, the CAN signals beyond the transceiver have asymmetric drivers. A low state on the CAN_TX pin activates strong drivers while a high state is driven weakly. Consequently, the active low is called the *dominant* state and the active high is the *recessive* state. If the CAN module is passive, the CAN_TX pin is always high. If two CAN nodes transmit at the same time, dominant bits overwrite recessive bits.

The CAN protocol specifies that all nodes trying to send a message on the CAN bus attempt to send a frame (shown in the figure below) once the CAN bus becomes available. The start of frame indicator (SOF) signals the beginning of a new frame. Each CAN node then begins transmitting its message starting with the message ID.

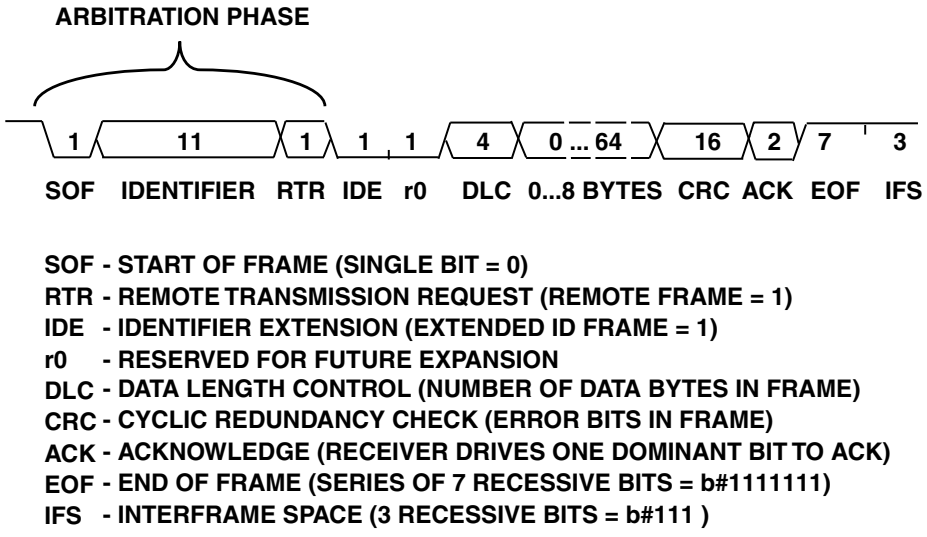


Figure 21-5: Standard CAN Frame

While transmitting, the CAN controller samples the CAN_RX pin to verify that the logic level being driven is the value it just placed on the CAN_TX pin. This is where the names for the logic levels apply. If a trans-

mitting node places a recessive 1 on the CAN_TX pin and detects a dominant 0 on the CAN_RX pin, it knows that another node has placed a dominant bit on the bus, which means another node is a higher priority.

Therefore, if the value sensed on the CAN_RX pin is the value driven on the CAN_TX pin, transmission continues, otherwise the CAN controller senses that it has lost arbitration and module configuration determines the next course of action.

The figure above shows a basic 11-bit identifier frame. After the SOF and identifier is the CAN_MBnn_ID1.RTR bit, which indicates whether the frame contains data (data frame) or is a request for data associated with the message identifier in the frame being sent (remote frame).

NOTE: In the CAN protocol, a dominant bit in the CAN_MBnn_ID1.RTR field wins arbitration against a remote frame request (CAN_MBnn_ID1.RTR=1) for the same message ID. This allows a remote request to be a lower priority than a data frame.

The next field of interest in the frame is the CAN_MBnn_ID1.IDE bit. When set, it indicates that the message is an extended frame with a 29-bit identifier instead of an 11-bit identifier. In an extended frame, the first part of the message resembles the following figure.

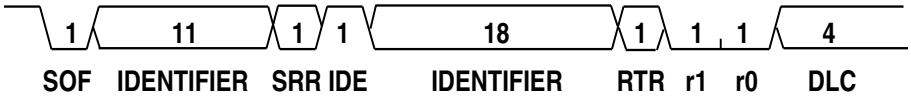


Figure 21-6: Extended CAN Frame

Therefore a dominant bit in the CAN_MBnn_ID1.IDE field wins arbitration against an extended frame with the same lower 11-bits and standard frames are higher priority than extended frames.

The substitute remote request (SRR, always sent as recessive), the reserved bits r0 and r1 (always sent as dominant), and the checksum (CRC) are generated automatically by the internal logic.

Data Transfer Modes

The following sections provide information on the data transfer modes supported by the CAN controller.

Transmit Operations

The following figure shows the CAN transmit operation. Mailboxes 24–31 are dedicated transmitters. Mailboxes 8–23 can be configured as transmitters by writing 0 to the corresponding bit in the CAN_MD1/CAN_MD2registers. After writing the data and the identifier into the mailbox area, the message is sent after mailbox n is enabled (CAN_MC1.MB=1) and, subsequently, the corresponding transmit request bit is set (CAN_TRS1.MB=1).

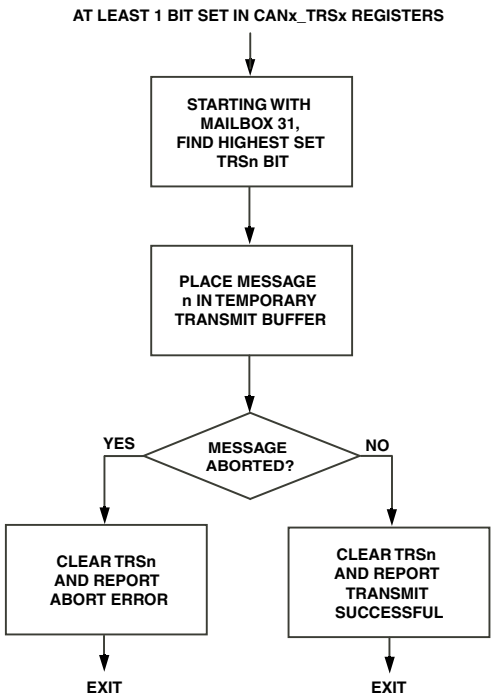


Figure 21-7: CAN Transmit Operation Flow Chart

When a transmission completes, the corresponding bits in the CAN_TRS1/CAN_TRS2 and CAN_TRR1/CAN_TRR2registers are cleared. If the transmission was successful, the corresponding bit in the CAN_TA1/CAN_TA2register is set. If the transmission was aborted due to lost arbitration or a CAN error, the corresponding bit in the CAN_AA1/CAN_AA2register is set. A requested transmission can also be manually aborted by setting the corresponding bit in the CAN_TRR1/CAN_TRR2register.

Multiple CAN_TRS1.MB bits can be set simultaneously by software, and these bits are reset after either a successful or an aborted transmission.

These bits are also set by the CAN hardware in the following cases:

- When using the auto-transmit mode of the universal counter,
- When a message loses arbitration and the single-shot CAN_OPSS1.MB bit is not set, or
- In the event of a remote frame request (only possible for receive/transmit mail-boxes if the automatic remote frame handling feature is enabled (CAN_RFH1.MB=1).

NOTE: Special care should be given to mailbox area management when a CAN_TRS1/CAN_TRS2 bit is set. Write access to the mailbox is permissible with a bit set, but changing data in such a mailbox may lead to unexpected data during transmission.

Enabling and disabling mailboxes has an impact on transmit requests. Setting the CAN_TRS1/CAN_TRS2 bit associated with a disabled mailbox may result in erroneous behavior. Similarly, disabling a mailbox before the associated CAN_TRS1/CAN_TRS2 bit is reset by the internal logic can cause unpredictable results.

Retransmission

Normally, the current message object is resent after arbitration is lost or an error frame is detected on the CAN bus line. If there is more than one transmit message object pending, the message object with the highest mailbox is sent first (see figure below). The currently aborted transmission is restarted after any messages with higher priority are sent.

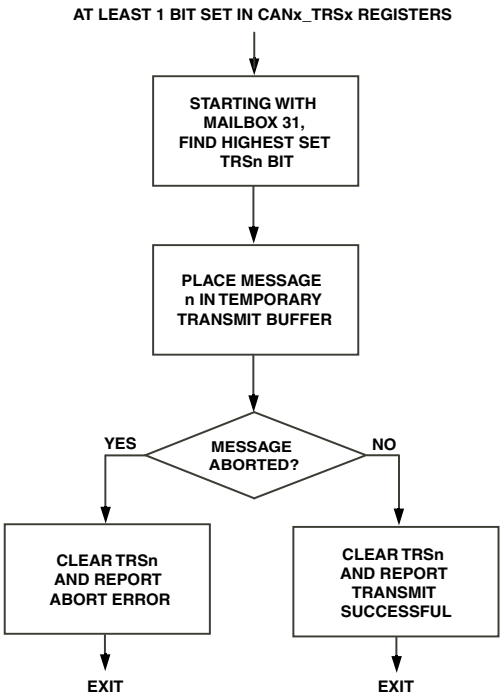


Figure 21-8: Transmit Flow

A message which is currently under preparation is not replaced by another message which is written into the mailbox. The message under preparation is one that is copied into the temporary transmit buffer when the internal transmit request for the CAN core module is set. The message in the buffer is not replaced until it is sent successfully, the arbitration on the CAN bus line is lost, or there is an error frame on the CAN bus line.

Single-Shot Transmission

If the single shot transmission feature is used (`CAN_OPSS1.MB`=1), the corresponding `CAN_TRS1` bit is cleared after the message is successfully sent or even if the transmission is aborted due to a lost arbitration or an error frame on the CAN bus line. Therefore, there is no further attempt to transmit the message again if the initial try failed, and the Abort error is reported (`CAN_AA1.MB`=1).

Auto-Transmission

In Auto-Transmit mode, the message in mailbox 11 (MB11) can be sent periodically using the universal counter. This mode is often used to broadcast heartbeats to all CAN nodes. Accordingly, messages sent this way usually have high priority.

The period value is written to the CAN_UCRC register. The Auto-Transmission mode is enabled by setting the CAN_UCCNF.UCCNF field to 0x03. When enabled, the counter CAN_UCCNT is loaded with the value in the CAN_UCRC register. The counter decrements to 0 at the CAN bit clock rate and is then reloaded from CAN_UCRC. Each time the counter reaches a value of 0, the CAN_TRS1.MB bit is automatically set by internal logic, and the corresponding message from mailbox 11 is sent.

For proper auto-transmit operation, mailbox 11 must be configured as a transmit mailbox and must contain valid data (identifier, control bits, and data) before the counter first expires after this mode is enabled.

Receive Operation

The CAN hardware autonomously receives messages and discards invalid messages. Once a valid message is successfully received, the receive logic interrogates all enabled receive mailboxes sequentially, from mailbox 23 down to mailbox 0, whether the message is of interest to the local node or not.

Each incoming data frame is compared to all identifiers stored in active receive mailboxes (respective mailbox indices of CAN_MD1 and CAN_MC1 registers set to 1) and to all active transmit mailboxes with the remote frame handling feature enabled (=1). The message identifier of the received message, along with the identifier extension (CAN_MBnn_ID1.IDE) and remote transmission request (CAN_MBnn_ID1.RTR) bits, are compared against each mailbox's register settings. In standard mode, the message is compared to the content of the CAN_MBnn_ID1register. In extended mode, the content of the CAN_MBnn_ID0register must also match.

If the acceptance mask enable CAN_MBnn_ID1.AME bit is not set, a match is signaled only if CAN_MBnn_ID1.IDE, CAN_MBnn_ID1.RTR, and all (11 or 29) identifier bits are exact. If, however, the CAN_MBnn_ID1.AME bit is set, the acceptance mask registers (CAN_AMnnH/L) determine which of the CAN_MBnn_ID1.IDE and CAN_MBnn_ID1.RTR bits need to match.

The following logic applies:

[(Received Message ID) XNOR (CAN_MBnn_ID0/1)] OR [(CAN_MBnn_ID1.AME) AND (CAN_AMnnH/L)].

This logic appears graphically in the figure below.

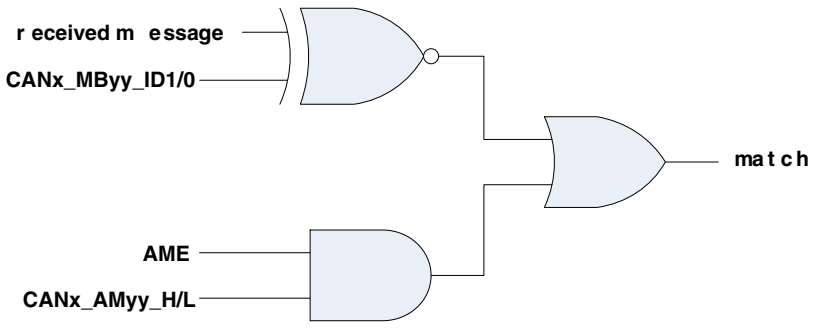


Figure 21-9: CAN Message Receive Logic

A one (1) at the respective bit position in the CAN_AMnnH/CAN_AMnnL mask registers means that the bit does not need to match when CAN_MBnn_ID1.AME=1. This way, a mailbox can accept a group of messages.

Table 21-3: Mailbox Used for Acceptance Filtering

MCn	MDn	RFHn	Mailbox n	Comment
0	X	X	Ignored	Mailbox n disabled
1	0	0	Ignored	Mailbox n enabled, Mailbox n configured for transmit, Remote frame handling disabled
1	0	1	Used	Mailbox n enabled, Mailbox n configured for transmit, Remote frame handling enabled
1	1	X	Used	Mailbox n enabled, Mailbox n configured for receive

If the acceptance filter finds a matching identifier, the content of the received data frame is stored in that mailbox. A received message is stored only once, even if multiple receive mailboxes match its identifier. If the current identifier does not match any mailbox, the message is not stored.

The figure below illustrates the decision tree of the receive logic when processing the individual mailboxes.

If a message is received for a mailbox and that mailbox still contains unread data (CAN_RMP1.MB), then the program has to decide whether the old message should be overwritten or not. If the CAN_OPSS1.MB bit is cleared, the corresponding CAN_RML1.MB bit is set, and the stored message is overwritten. This results in the receive message lost interrupt being raised (CAN_GIS.RMLIS is set). If, however, the CAN_OPSS1.MB bit is set, the next mailboxes are checked for another matching identifier. If no match is found, the message is discarded, and the next message is checked.

NOTE: If a receive mailbox is disabled, an ongoing receive message for that mailbox is lost even if a second mailbox is configured to receive the same identifier.

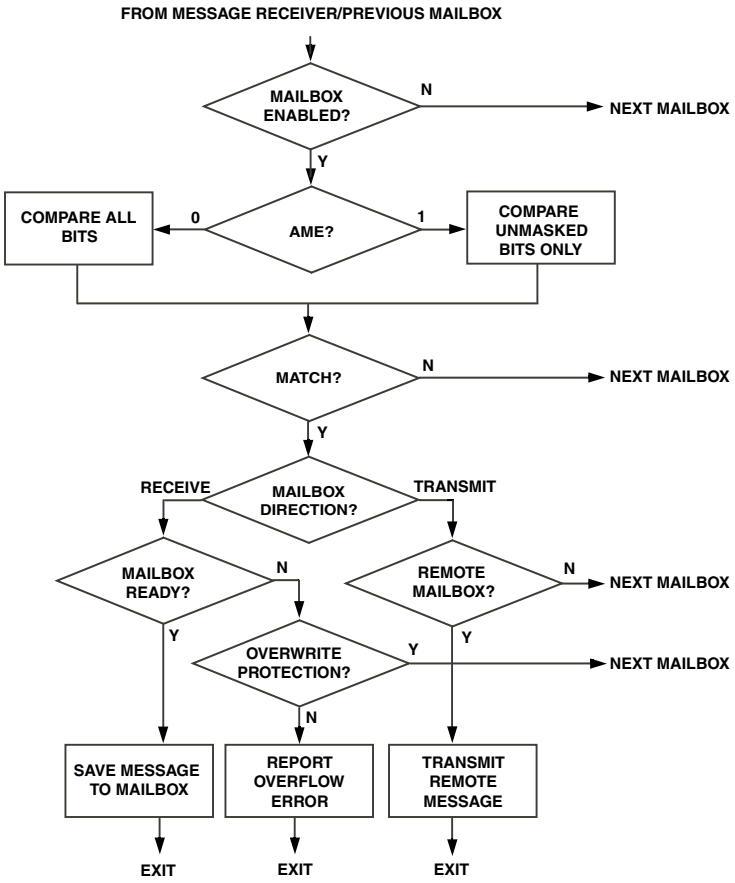


Figure 21-10: CAN Receive Operation Flow Chart

Data Acceptance Filtering

If DeviceNet mode is enabled (`CAN_CTL.DNM` = 1) and the mailbox is set up for filtering on data field, the filtering is done on the standard ID of the message and data fields. The data field filtering can be programmed for either the first byte only or the first two bytes, as shown the table below.

If the `CAN_AMnnH.FDF`bit is set, the corresponding `CAN_AMnnL`register holds the data field mask (DFM bits 15–0]). If the `CAN_AMnnH.FDF`bit is cleared, the corresponding `CAN_AMnnL`register holds the extended identifier mask (`CAN_AMnnH.EXTID`bits 15–0).

Table 21-4: Data Field Filtering

FDF (Filter on Data Field)	FMD (Full Mask Data Field)	Description
0	0	Do not allow filtering on the data field
0	1	Not allowed. FMF must be 0 if FDF is 0
1	0	Filter on first data byte only
1	1	Filter on first two data bytes

Watchdog Mode

Watchdog mode is used to ensure that messages are received periodically. It is often used to observe whether or not a certain node on the network is alive and functioning properly, and, if not, to detect and manage its failure case accordingly.

This mode can be enabled by programming the universal counter to watchdog mode by setting the CAN_UCCNF.UCCNF to 0x2. Once enabled, the CAN_UCCNT register is loaded with the predefined value contained in CAN_UCRC. This counter then decrements at the CAN bit rate.

If the CAN_UCCNF.UCCT and CAN_UCCNF.UCRC bits are set and a message is received in mailbox 4 before the counter counts down to 0, the counter is reloaded with the CAN_UCRC contents. If the counter has counted down to 0 without receiving a message in mailbox 4, then the CAN_GIS.UCEIS bit is set, and the counter is automatically reloaded with the contents of the CAN_UCRC register. If an interrupt is desired for this event, the CAN_GIM.UCEIM bit must also be set. With the mask bit set, when a watchdog interrupt occurs, the CAN_GIF.UCEIF bit is also set.

The counter can be reloaded with the contents of CAN_UCRC or disabled by writing to the CAN_UCCNF register.

The time period it takes for the watchdog interrupt to occur is controlled by the value written into the CAN_UCRC register.

Time Stamps

To get an indication of the time of the receive or transmit time for each message, program the CAN universal counter to Time Stamp mode. This mode can be enabled by setting the CAN_UCCNF.UCCNF field to 0x01.

If enabled, the value of the 16-bit free-running counter (CAN_UCCNT) is written into the CAN_MBnn_TIMESTAMP register of the corresponding mailbox when a received message is stored or a message is transmitted.

The time stamp value is captured at the sample point of the Start Of Frame (SOF) bit of each incoming or outgoing message. Afterwards, this time stamp value is copied to the CAN_MBnn_TIMESTAMP register of the corresponding mailbox.

If the mailbox is configured for automatic remote frame handling (CAN_RFH1.MB = 1), the time stamp value is written for transmission of a data frame (mailbox configured as transmit) or the reception of the requested data frame (mailbox configured as receive).

The counter can be cleared by setting the CAN_UCCNF.UCRC bit to 1, or disabled by clearing the CAN_UCCNF.UCE bit. The counter can also be loaded with a value by writing to the CAN_UCCNT register itself.

It is also possible to clear the counter (CAN_UCCNT) by reception of a message in mailbox number 4 (synchronization of all time stamp counters in the system). This is accomplished by setting the CAN_UCCNF.UCCT bit.

An overflow of the counter sets the CAN_GIS.UCEIS bit. A global CAN interrupt can optionally occur by unmasking the CAN_GIM.UCEIM bit. If the interrupt source is unmasked, the CAN_GIF.UCEIF bit is also set.

Remote Frame Handling

Automatic handling of remote frames can be enabled for a transmit mailbox by setting the corresponding CAN_RFH1.MB bit of a transmit mailbox.

Remote frames are data frames with no data field and the CAN_MBnn_ID1.RTR bit set. The data length code (DLC) of the responding data frame is overruled by the DLC of the requesting remote frame. A DLC can be programmed with values in the range of 0 to 15, but DLC values greater than 8 are considered as 8.

A remote frame contains:

- The identifier bits
- The control field DLC (data length count)
- The remote transmission request (CAN_MBnn_ID1.RTR) bit

Only configurable mailboxes, MB8–MB23, can process remote frames, but all mailboxes can receive and transmit remote frame requests. When setup for automatic remote frame handling, the CAN_OPSS1 register has no effect. All content of a mailbox is always overwritten by an incoming message.

NOTE: If a remote frame is received, the DLC of the corresponding mailbox is overwritten with the received value.

Erroneous behavior may result when the CAN_RFH1.MB bit is changed while the corresponding mailbox is currently being processed. To avoid the risk of inconsistent messages, programs should temporarily disable the mailbox while its data registers are updated.

Temporarily Disabling CAN Mailbox

If a mailbox is enabled and configured to transmit, write accesses to the data field should be guarded to avoid transmitting inconsistent messages. Special care must be taken if the mailbox is transmitting (or attempting to transmit) repeatedly. Also, if this mailbox is used for Automatic remote frame handling, the data field must be updated without losing an incoming remote request frame and without sending inconsistent data. Therefore, the CAN controller allows for temporary disabling the mailbox using the mailbox temporary disable register (CAN_MBTD).

The pointer to the requested mailbox must be written to the CAN_MBTD.TDPTR field, and the CAN_MBTD.TDR bit must be set. The corresponding CAN_MBTD.TDA flag is subsequently set by the internal logic.

If a mailbox is configured as transmit (CAN_MD1 = 0) and the CAN_MBTD.TDA bit is set, the content of the data field of that mailbox can be updated. If there is an incoming remote Request Frame while the mailbox is temporarily disabled, the corresponding transmit request set bit (CAN_TRS1.MB) is set by the internal logic and the data length code (DLC) of the incoming message is written to the corresponding mailbox. However, the message being requested is not sent until the CAN_MBTD.TDR bit is cleared. Similarly, all transmit requests for temporarily disabled mailboxes are ignored until the CAN_MBTD.TDR bit is cleared.

Additionally, transmission of a message is immediately aborted if the mailbox is temporarily disabled and the corresponding transmission request reset (CAN_TRR1.MB) bit for this mailbox is set.

If a mailbox is configured to receive (CAN_MD1 = 1), then after issuing a temporary disable request, the CAN_MBTD.TDA flag is set, and the mailbox is not processed. If there is an incoming message for a mailbox being temporarily disabled, the internal logic waits until the reception is complete or there is an error on the CAN bus before setting CAN_MBTD.TDA. Once this flag is set, the mailbox can then be completely disabled (CAN_MC1 = 0) without the risk of losing an incoming frame. The CAN_MBTD.TDR bit must then be reset as soon as possible.

When the CAN_MBTD.TDA flag is set for a given mailbox, only the data field of that mailbox can be updated. Accesses to the control bits and the identifier are denied.

CAN Operating Modes

The CAN controller is in configuration mode when coming out of processor reset or hibernate. It is only when the CAN is in configuration mode that hardware behavior can be altered. Before initializing the mailboxes themselves, the CAN bit timing must be set up to work on the CAN bus to which the controller is expected to connect.

Bit Timing

The CAN controller does not have a dedicated clock. Instead, the CAN clock is derived from the system clock based on a configurable number of time quanta. The Time Quantum (TQ) is derived from the formula:

TQ = (BRP + 1)/SCLK,

where BRP is the 10-bit bit rate prescaler field in the CAN_CLK register.

Although the CAN_CLK.BRP field can be set to any value, it is recommended that the value be greater than or equal to 4, as restrictions apply to the bit timing configuration when BRP is less than 4.

The CAN_CLK register defines the TQ value, and multiple time quanta make up the duration of a CAN bit on the bus. The CAN_TIMING register controls the nominal bit time and the sample point of the individual bits in the CAN protocol. The figure below shows the three phases of a CAN bit—the synchronization segment, the segment before the sample point, and the segment after the sample point.

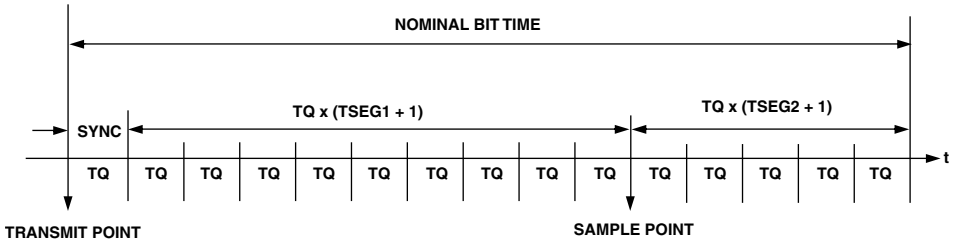


Figure 21-11: Three Phases of a CAN Bit

The synchronization segment is fixed to one TQ. It is required to synchronize the nodes on the bus. All signal edges are expected to occur within this segment.

The CAN_TIMING.TSEG1 and CAN_TIMING.TSEG2 fields control how many TQs the CAN bits consist of, resulting in the CAN bit rate. The nominal bit time is given by the following formula.

$t_{BIT} = TQ \times [1 + (1 + TSEG1) + (1 + TSEG2)]$

For safe receive operations on given physical networks, the sample point is programmable by the CAN_TIMING.TSEG1 field. The CAN_TIMING.TSEG2 field holds the number of TQs needed to complete the bit time. Often, best sample reliability is achieved with sample points in the high 80% range of the bit time. Never use sample points lower than 50%. Therefore, CAN_TIMING.TSEG1 should always be greater than or equal to CAN_TIMING.TSEG2.

The CAN module does not distinguish between the Propagation Segment and the phase segment-1 as defined by the standard. The CAN_TIMING.TSEG1 value is intended to cover both of them. The CAN_TIMING.TSEG2 value represents the phase segment-2.

If the CAN module detects a recessive-to-dominant edge outside the synchronization segment, it can automatically move the sampling point such that the CAN bit is still handled properly. The synchronization jump width (CAN_TIMING.SJW) field specifies the maximum number of TQs, ranging from 1 to 4 (SJW + 1), allowed for such a re-synchronization attempt. The SJW value should not exceed CAN_TIMING.TSEG2 or CAN_TIMING.TSEG1. Therefore, the fundamental rule for writing CAN_TIMING is:

$SJW \leq TSEG2 \leq TSEG1$

In addition to this fundamental rule, CAN_TIMING.TSEG2 must also be greater than or equal to the information processing time (IPT). This is the time required by the logic to sample the CAN_RX input, which is 3 system clock cycles.

Because of this, restrictions apply to the minimal value of CAN_TIMING.TSEG2 if CAN_CLK.BRP is lower than 2. If CAN_CLK.BRP is set to 0, the CAN_TIMING.TSEG2 field must be greater than or equal to 2. If CAN_CLK.BRP is set to 1, the minimum CAN_TIMING.TSEG2 value is 1.

NOTE: All nodes on a CAN bus should use the same nominal bit rate.

With all the timing parameters set, the final consideration is how sampling is performed. The default behavior of the CAN controller is to sample the CAN bit once at the sampling point described by the CAN_TIMING register, controlled by the CAN_TIMING.SAM bit. If this bit is set, however, the input signal is oversampled three times at the system clock rate. The resulting value is generated by a majority decision of the three sample values. Always keep the CAN_TIMING.SAM bit cleared if the BRP value is less than 4.

Do not modify the CAN_CLK and CAN_TIMING registers during normal operation. Always enter configuration mode first. Writes to these registers have no effect if not in configuration or debug mode. If not coming out of processor reset or hibernate, enter configuration mode by setting the CAN_CTL.CCR bit and poll the CAN_STAT register until CAN_STAT.CCA is set.

NOTE: If the CAN_TIMING.TSEG1 field is programmed to 0, the module doesn't leave the configuration mode.

During configuration mode, the module is not active on the CAN bus line. The CAN_TX output pin remains recessive and the module does not receive/transmit messages or error frames. After leaving the configuration mode, all CAN internal core registers and the CAN error counters are set to their initial values.

A soft reset does not change the values of CAN_CLK and CAN_TIMING. Therefore, an ongoing transfer through the CAN bus cannot be corrupted by changing the bit timing parameter or initiating the soft reset (by setting the CAN_CTL.SRS bit).

CAN Low Power Features

The CAN module includes built-in sleep and suspend modes to save power.

It also responds to the hibernate power state on processors that support that state.

The behavior of the CAN module in these modes is described in the following sections.

Built-In Suspend Mode

The most modest of power savings mode is the suspend mode. This mode is entered by setting the CAN_CTL.CSR bit. The module enters the suspend mode after the current operation of the CAN bus is finished, at which point the internal logic sets the CAN_STAT.CSA bit. Once this mode is entered, the module is no longer active on the CAN bus line, slightly reducing power consumption.

In suspend mode the CAN_TX output pin remains in a recessive state, and the module does not receive/transmit messages or error frames. The content of the CAN_CEC register remains unchanged. Suspend mode can subsequently be exited by clearing CAN_CTL.CSR.

The only differences between suspend mode and configuration mode are that writes to the CAN_CLK and CAN_TIMING registers are locked in suspend mode, and the CAN_CTL and CAN_STAT registers are not reset when exiting suspend mode.

Built-In Sleep Mode

The next level of power savings can be realized by using the module's built-in sleep mode. This mode is entered by setting the CAN_CTL.SMR bit. The module enters the sleep mode after the current operation of the CAN bus is finished. Once this mode is entered, many of the internal CAN module clocks are shut off, reducing power consumption, and the CAN_INT.SMACK bit is set.

When the CAN module is in sleep mode, all register reads return the contents of CAN_INT instead of the usual contents. All register writes, except to CAN_INT, are ignored in sleep mode. A small part of the module is clocked continuously to allow for wake up out of sleep mode.

A write to the CAN_INT register ends sleep mode. If the CAN_CTL.WBA bit is set before entering sleep mode, a dominant bit on the CAN_RX pin also ends sleep mode. When software sets the CAN_CTL.SMR bit, hardware sets the CAN_CTL.CSR bit as well, making sleep mode a super set of suspend mode. When the controller wakes up from sleep mode, hardware automatically clears CAN_CTL.SMR and CAN_CTL.CSR. If,

however, the controller never enters sleep mode because the wake-up condition was met before CAN_INT.SMACK bit turns to one, the CAN_CTL.SMR and CAN_CTL.CSR bits may not be automatically cleared. Therefore, it is good programming practice to always clear those two bits by software when returning from sleep mode.

Wake-Up From Hibernate State

Many processors provide a hibernate state, where the internal voltage regulator shuts off the internal power supply to the chip, turning off the core and system clocks in the process. In this mode, the only power drawn is that used by the regulator circuitry awaiting any of the possible hibernate wake-up events. One such event is a wake-up due to CAN bus activity.

After hibernation, the CAN module must be re-initialized. For low power designs, the external CAN bus transceiver is typically put into standby mode through one of the processor's general purpose I/O pins. While in standby mode, the CAN transceiver continually drives the recessive logic 1 level onto the CAN_RX pin. If the transceiver then senses CAN bus activity, it drives the CAN_RX pin to the dominant logic 0 level. This signals the processor that CAN bus activity is detected. If the internal voltage regulator is programmed to recognize CAN bus activity as an event to exit the hibernate state, the part responds appropriately. Otherwise, the activity on the CAN_RX pin has no effect on the processor state.

Soft Reset

The CAN controller features a build-in reset mechanism called Soft Reset. Soft reset is entered immediately after software has set the CAN_CTL.SRSbit. Soft reset brings all control registers to a defined state and mailbox and error registers remain unaffected. Soft reset does not alter the CAN_TIMING and CAN_CLKregisters and does not disturb the on-going transmission of a currently pending message, acknowledge bit or error frame. However, when recovering from soft reset, software may lose track of transmission or reception reports and interrupts.

CAN Event Control

The following is a description of how CAN events are generated and controlled.

CAN Interrupt Signals

The CAN module provides three independent interrupts: two mailbox interrupts (mailbox receive interrupt (MBRIRQ) and mailbox transmit interrupt (MBTIRQ) and a global CAN status interrupt (GIRQ). The values of these three interrupts can also be read back through the CAN_GISregisters.

Mailbox Interrupts

Each of the 32 mailboxes in the CAN module may generate a receive or transmit interrupt, depending on the mailbox configuration. To enable a mailbox to generate an interrupt, set the corresponding CAN_MBIM1 bit.

If a mailbox is configured as a receive mailbox, the corresponding CAN_MBRIF1 bit and CAN_RMP1 bit are set after a received message is stored in mailbox n. If the automatic remote frame handling feature is used (CAN_RFH1=1), the receive interrupt flag is set after the requested data frame is stored in the mailbox.

If any CAN_MBRIF1 bits are set, the CAN_INT.MBRIRQ interrupt is generated. In order to clear the CAN_INT.MBRIRQ interrupt request, all of the set CAN_MBRIF1 bits must be cleared by software by writing a 1 to those set bit locations in CAN_MBRIF1. Prior to this, the corresponding CAN_RMP1 bit must also be cleared by software.

If a mailbox is configured as a transmit mailbox, the corresponding CAN_MBTIF1 bit in the transmit interrupt flag is set after the message in mailbox n is sent correctly, and the corresponding CAN_TA1 bit also gets set. The CAN_TA1 bits maintain their state even after the corresponding mailbox n is disabled (CAN_MC1=0). If the automatic remote frame handling feature is used, then transmit interrupt flag is set after the requested data frame is sent from the mailbox.

If any CAN_MBTIF1.MB, bits are set the MBTIRQ interrupt output is raised in the CAN_INT register. In order to clear the MBTIRQ interrupt request, all of the bits set in the CAN_MBTIF1 register must be cleared by software by writing a 1 to those set bit locations. Additionally, software must clear the associated CAN_TA1 bit or set the associated CAN_TRS1 bit to clear the interrupt source that asserts the CAN_MBTIF1 bit.

Global Interrupt

The global CAN interrupt logic is implemented with three registers:

- The CAN_GIMregister, where each interrupt source can be enabled or disabled separately
- The CAN_GIS register
- The CAN_GIF register

The interrupt mask bits only affect the content of the CAN_GIF register. If the mask bit is not set in the CAN_GIM register, the corresponding flag bit is not set when the event occurs. The interrupt status bits in the CAN_GIS register, however, are always set if the corresponding interrupt event occurs, independent of the mask bits. Thus, the interrupt status bits can be used for polling of interrupt events.

The CAN_INT.GIRQbit is only asserted if a bit in the CAN_GIFregister is set. The read-only CAN_INT.GIRQbit remains set as long as at least one bit in CAN_GIF is set. All bits in the interrupt status and interrupt flag registers remain set until cleared by software or a soft reset has occurred.

NOTE: The CAN_GIF register is read-only (RO). In the global CAN interrupt ISR, the interrupt latch should be cleared by writing to 1 to the corresponding bit of the CAN_GIS register, which clears the related bits of the CAN_GIS and CAN_GIF registers, as well as the CAN_INT.GIRQ bit.

There are several interrupt events that can activate this GIRQ interrupt:

- Access denied interrupt (CAN_GIM.ADIM, CAN_GIS.ADIS, CAN_GIF.ADIF): At least one access to the mailbox RAM occurred during a data update by internal logic.
- Universal counter exceeded interrupt (CAN_GIM.UCEIM, CAN_GIS.UCEIS, CAN_GIF.UCEIF): There was an overflow of the universal counter (in Time Stamp mode or Event Counter mode) or the counter has reached the value 0x0000 (in Watchdog mode).
- Receive message lost interrupt (CAN_GIM.RMLIM, CAN_GIS.RMLIS, CAN_GIF.RMLIF): A message is received for a mailbox that currently contains unread data. At least one bit in the CAN_RMLn register is set. If the bit in CAN_GIS and CAN_GIF registers is cleared and there is at least one bit in CAN_RML1 still set, then the bit in the CAN_GIS and CAN_GIF registers is not set again. The internal interrupt source signal is only active if a new bit in CAN_RML1 is set.
- Abort acknowledge interrupt (CAN_GIM.AAIM, CAN_GIS.AAIS, CAN_GIF.AAIF): At least one CAN_AA1.MB bit in the CAN_AA1 registers is set. If the bit in the CAN_GIS and CAN_GIF registers is cleared and there is at least one bit in CAN_AA1 still set, then the bit in the CAN_GIS and CAN_GIF registers is not set again. The internal interrupt source signal is only active if a new bit in CAN_AA1 is set. The CAN_AA1.MB bits maintain state even after the corresponding mailbox n is disabled (CAN_MC1 = 0).
- Access to un implemented address interrupt (CAN_GIM.UIAIM, CAN_GIS.UIAIS, CAN_GIF.UIAIF): There was a CPU access to an address which is not implemented in the controller module.
- Wake-up interrupt (CAN_GIM.WUIM, CAN_GIS.WUIS, CAN_GIF.WUIF): The CAN module has left the sleep mode because of detected activity on the CAN bus line.
- Bus-Off interrupt (CAN_GIM.BOIM, CAN_GIS.BOIS, CAN_GIF.BOIF): The CAN module has entered the bus-off state. This interrupt source is active if the status of the CAN core changes from normal operation mode to the bus-off mode. If the bit in the CAN_GIS and CAN_GIF registers is cleared and the bus-off mode is still active, then this bit is not set again. If the module leaves the bus-off mode, the bit in the CAN_GIS and CAN_GIF registers remains set, if not explicitly cleared.
- Error-passive interrupt (CAN_GIM.EPIM, CAN_GIS.EPIS, CAN_GIF.EPIF): The CAN module has entered the error-passive state. This interrupt source is active if the status of the CAN module changes from the error-active mode to the error-passive mode. If the bit in the CAN_GIS and CAN_GIF registers is cleared and the error-passive mode is still active, then this bit is not set again. If the module leaves the error-passive mode, the bit in the CAN_GIS and CAN_GIF registers remains set, if not explicitly cleared.
- Error warning receive interrupt (CAN_GIM.EWRIM, CAN_GIS.EWRIS, CAN_GIF.EWRIF): The CAN receive error counter (CAN_CEC.RXECNT) has reached the warning limit. If the bit in the CAN_GIS and CAN_GIF registers) is cleared and the error warning mode is still active, this bit is not set again. If the module leaves the error warning mode, the bit in the CAN_GIS and CAN_GIF registers remains set, if not explicitly cleared.
- Error warning transmit interrupt (CAN_GIM.EWTIM, CAN_GIS.EWTIS, CAN_GIF.EWTIF): The CAN transmit error counter (CAN_CEC.TXECNT) has reached the warning limit. If the bit in the CAN_GIS and CAN_GIF registers is cleared and the error warning mode is still active, this bit is not set again. If the

module leaves the error warning mode, the bit in the CAN_GIS and CAN_GIF registers remains set, if not explicitly cleared.

Event Counter

For diagnostic functions, it is possible to use the universal counter as an event counter. The counter can be programmed in the CAN_UCCNF[3:0] field to increment on one of these conditions:

- 0x6 – CAN error frame. Counter is incremented if there is an error frame on the CAN bus line.
- 0x7 – CAN overload frame. Counter is incremented if there is an overload frame on the CAN bus line.
- 0x8 – Lost arbitration. Counter is incremented every time arbitration on the CAN line is lost during transmission.
- 0x9 – Transmission aborted. Counter is incremented every time arbitration is lost or a transmit request is canceled (CAN_AA1 is set).
- 0xA – Transmission succeeded. Counter is incremented every time a message sends without detected errors (CAN_TA1 is set).
- 0xB – Receive message rejected. Counter is incremented every time a message is received without detected errors but not stored in a mailbox because there is no matching identifier found.
- 0xC – Receive message lost. Counter is incremented every time a message is received without detected errors but not stored in a mailbox because the mailbox contains unread data (CAN_RML1 is set).
- 0xD – Message received. Counter is incremented every time a message is received without detected errors, whether the received message is rejected or stored in a mailbox.
- 0xE – Message stored. Counter is incremented every time a message is received without detected errors, has an identifier that matches an enabled receive mailbox, and is stored in the receive mailbox (CAN_RMP1 is set).
- 0xF – Valid message. Counter is incremented every time a valid transmit or receive message is detected on the CAN bus line.

CAN Warnings and Errors

CAN warnings and errors are controlled using the error counter (CAN_CEC) register, the error status (CAN_ESR) register, and the error counter warning level (CAN_EWR) register. Error handling is described in the following sections.

Programmable Warning Limits

Programs can set the warning level for CAN_GIS.EWTISand CAN_GIS.EWRISseparately by writing to the CAN_EWR.EWLRECand CAN_EWR.EWLTECfields. After power-on reset, the CAN_EWRregister is set to the

default warning level of 96 for both error counters. After a soft reset, the contents of this register remain unchanged.

Error Handling

Error management is an integral part of the CAN standard. Five different kinds of bus errors may occur during transmissions:

- Bit error – This error is only detected by the transmitting node. Whenever a node is transmitting, it continuously monitors its receive pin (CAN_RX) and compares the received bit with the transmitted bit. During the arbitration phase, the node postpones the transmission if the received and transmitted bits do not match. However, after the arbitration phase (that is, once the CAN_MBnn_ID1.RTR bit is sent successfully), a bit error is signaled any time the value on CAN_RX does not equal what is being transmitted on the CAN_TX pin.
- Form error – Occurs any time a fixed-form bit position in the CAN frame contains one or more illegal bits--that is, when a dominant bit is detected at a delimiter or end-of-frame bit position.
- Acknowledge error – Occurs whenever a message is sent and no receivers drive an acknowledge bit.
- CRC error – Occurs whenever a receiver calculates the CRC on the data it received and finds it different than the CRC that was transmitted on the bus itself.
- Stuff error – The CAN specification requires the transmitter to insert an extra stuff bit of opposite value after 5 bits have been transmitted with the same value. The receiver disregards the value of these stuff bits. However, it takes advantage of the signal edge to re-synchronize itself. A stuff error occurs on receiving nodes whenever the 6th consecutive bit value is the same as the previous five bits.

Once the CAN module detects any of the above errors, it updates the CAN_ESR and CAN_CEC registers. In addition to the standard errors, the CAN_ESR.SAO flag signals when the CAN_RX pin sticks at dominant level, indicating a possibility of shorted wires.

Error Frames

It is very important that all nodes on the CAN bus ignore data frames that any single node failed to receive. To accomplish this, every node sends an error frame as soon as it has detected an error as shown in the figure below.

A device has detected an error still completes the ongoing bit and initiates an error frame by sending six dominant and eight recessive bits to the bus. Since this is a violation of the bit stuffing rule, all nodes are informed that the ongoing frame needs to be discarded. (All receivers that did not detect the transmission error in the first instance now detect a stuff bit error.)

The transmitter may detect a normal bit error sooner. It aborts the transmission of the ongoing frame and tries resending it later.

When all nodes on the bus have detected the error, they also send 6 dominant and 8 recessive bits to the bus. The resulting error frame consists of two different fields. The first field is given by the superposition

of error flags contributed from the different stations, which is a sequence of 6 to 12 dominant bits. The second field is the error delimiter and consists of 8 recessive bits indicating the end of frame.

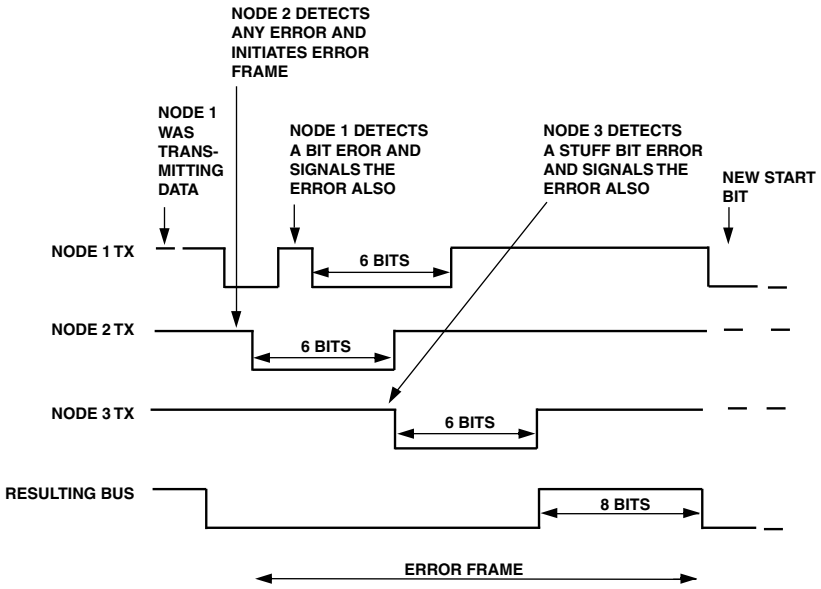


Figure 21-12: CAN Error Example

For CRC errors, the error frame is initiated at the end of the frame, rather than immediately after the failing bit.

After having received 8 recessive bits, every node knows that the error condition is resolved and, if messages are pending, starts transmission. The transmitter that had to abort its operation must win the new arbitration again; otherwise its message is delayed as determined by priority.

Because the transmission of an error frame destroys the frame under transmission, a faulty node erroneously detecting an error can block the bus. Because of this, there are two node states which determine a nodes right to signal an error—error-active and error-passive.

- *Error-active* nodes are those which have an error detection rate below a certain limit. These nodes drive an Active Error Flag of 6 dominant bits.
- *Error-passive* nodes have a higher error detection rate and are suspected of having a local problem and therefore have a limited right to signal errors. These nodes drive a passive error flag consisting of 6 recessive bits. Therefore an error-passive transmitting node is still able to inform the other nodes about the aborting of a self-transmitted frame, but it is no longer able to destroy correctly received frames of other nodes.

Error Levels

The CAN specification requires each node in the system to operate in one of three levels which are shown in the table below. This prevents nodes with high error rates from blocking the entire network, as the errors may be caused by local hardware. The CAN module provides an error counter for transmit (TEC) and an error counter for receive (REC). The `CAN_CEC` register contains each of these 8-bit counters.

After initialization, both the TEC and the REC counters are 0. Each time a bus error occurs, one of the counters is incremented by either 1 or 8, depending on the error situation (documented in Version 2.0 of the CAN Specification). Successful transmit or receive operations decrement the respective counter by 1.

If either of the error counters exceeds 127, the CAN module goes into an error-passive state and the CAN_STAT.EP bit is set. Once this occurs, the module is not allowed to send any more active error frames. However, the module is still allowed to transmit messages and to signal passive error frames in case the transmission fails due to bit errors.

If one of the counters exceeds 255 (that is, when an 8-bit counter overflows), the CAN module is disconnected from the bus and it goes into bus-off mode. In this mode the CAN_STAT.EBO bit is set. Software intervention is required to recover from this state, unless the CAN_CTL.ABO bit is enabled, which puts the module into active mode after the bus-off recovery sequence.

Table 21-5: CAN Error Level Description

Level	Condition	Description
Error active	Transmit and receive error counters <128	This is the initial condition level. As long as errors stay below 128, the node will drive active error flags during error frames.
Error passive	Transmit or receive error counter value between 128 and 255, inclusive	Errors have accumulated to a level that requires the node to drive passive error flags during error frames.
Bus off	Transmit or receive error counters greater than 255	CAN module goes into bus-off mode

In addition to the three levels in the table, the CAN module also generates separate transmit and receive warnings (CAN specification enhancement). By default, when one of the error counters exceeds 96, a warning is signaled and is reported in the CAN_STAT register. The CAN receive warning flag (CAN_STAT.WR) bit is set when CAN_CEC.RXECNT exceeds 96. The CAN transmit warning flag (CAN_STAT.WT) bit is set when CAN_CEC.TXECNT exceeds 96. The error warning level can be programmed using the error warning register (CAN_EWR).

Additionally, interrupts can occur for all of these levels by unmasking them in the global CAN interrupt mask register (CAN_GIM). These interrupts include the bus-off interrupt (CAN_GIM.BOIM), the Error-Passive interrupt (CAN_GIM.EPIM), the error warning receive interrupt (CAN_GIM.EWRIM), and the Error Warning Transmit interrupt (CAN_GIM.EWTIM).

During the bus-off recovery sequence, the configuration mode request CAN_CTL.CCR bit is set by the internal logic, and the CAN core module does not automatically come out of the bus-off mode. The CAN_CTL.CCR bit cannot be reset until the bus-off recovery sequence has completed.

NOTE: This behavior can be overridden by setting the CAN_CTL.ABO bit. After exiting the bus-off or configuration modes, the CAN error counters are reset.

CAN Debug and Test Modes

The CAN module contains test mode features that aid in the debugging of the CAN software and system.

NOTE: When these features are used, the CAN module may not be compliant to the CAN specification. All test modes should be enabled or disabled only when the module is in Configuration mode (CAN_STAT.CCA=1) or in Suspend mode (CAN_STAT.CSA=1).

The CAN_DBG.CDE bit is used to gain access to all of the debug features. This bit must be set to enable the test mode, and it must be written first before any other writes to the CAN_DBG register. When the CAN_DBG.CDE bit is cleared, all debug features are disabled.

When the CAN_DBG.CDE bit is set, it enables writes to the other bits of the CAN_DBG register. It also enables these features, which are not compliant with the CAN standard:

- Bit timing registers can be changed anytime, not only during configuration mode. This includes the CAN_CLK and CAN_TIMING registers.
- Write access is allowed to the normally read-only CAN_CEC register.

The other bits in the debug register are described below.

- The CAN_DBG.MRB bit is used to enable the read back mode. In this mode, a message transmitted on the CAN bus (or through an internal loop back mode) is received back directly to the internal receive buffer. After a correct transmission, the internal logic treats this as a normal receive message. This feature allows the user to test most of the CAN features without an external device.
- The CAN_DBG.MAA bit allows the CAN module to generate its own acknowledge during the ACK slot of the CAN frame. No external devices or connections are necessary to read back a transmit message. In this mode, the message that is sent is automatically stored in the internal receive buffer. In Auto Acknowledge mode, the module itself transmits the acknowledge. This acknowledge can be programmed to appear on the CAN_TX pin if CAN_DBG.DIL=1 and CAN_DBG.DTO= 0. If the acknowledge is only going to be used internally, then these test mode bits should be set to CAN_DBG.DIL= 0 and CAN_DBG.DTO=1.
- The CAN_DBG.DIL bit is used to internally enable the transmit output to be routed back to the receive input.
- The CAN_DBG.DTO bit is used to disable the CAN_TX output pin. When this bit is set, the CAN_TX pin continuously drives recessive bits.
- The CAN_DBG.DRI bit is used to disable the CAN_RX input. When set, the internal logic receives recessive bits or receives the internally generated transmit value in the case of the internal loop enabled (CAN_DBG.DIL= 0). In either case, the value on the CAN_RX input pin is ignored.
- The CAN_DBG.DEC bit is used to disable the transmit and receive error counters in the CAN_CEC register. When this bit is set, the CAN_CEC holds its current contents and is not allowed to increment or decrement the error counters. This mode does not conform to the CAN specification.

NOTE: Writes to the error counter registers should be performed in debug mode only. Write access during reception may lead to undefined values. The maximum value which can be written into the error counters is 255. Therefore, the error counter value of 256, which forces the module into the bus off state, cannot be written into the error counter registers.

Table 21-6: Common CAN Test Mode Bit Combinations

MRB	MAA	DIL	DTO	DRI	CDE	Functional Description
X	X	X	X	X	0	Normal mode, not debug mode
0	X	X	X	X	X	No readback of transmit message
1	0	1	0	0	1	Normal transmission on CAN bus line. Read back. External acknowledge from external device required.
1	1	1	0	0	1	Normal transmission on CAN bus line. Read back. No external acknowledge required. Transmit message and acknowledge are transmitted on CAN bus line. CAN_RX input is enabled.
1	1	0	0	0	1	Normal transmission on CAN bus line. Read back. No external acknowledge required. Transmit message and acknowledge are transmitted on CAN bus line. CAN_RX input and internal loop are enabled (internal OR of TX and RX)
1	1	0	0	1	1	Normal transmission on CAN bus line. Read back. No external acknowledge required. Transmit message and acknowledge are transmitted on CAN bus line. CAN_RX input is ignored. Internal loop is enabled.
1	1	0	1	1	1	No transmission on CAN bus line. Read back. No external acknowledge required. Nether transmit message nor acknowledge are transmitted on CAN_TX. CAN_RX input is ignored. Internal loop is enabled.

ADSP-BF60x CAN Register Descriptions

Controller Area Network (CAN) contains the following registers.

Table 21-7: ADSP-BF60x CAN Register List

Name	Description
CAN_MC1	Mailbox Configuration 1 Register
CAN_MD1	Mailbox Direction 1 Register
CAN_TRS1	Transmission Request Set 1 Register
CAN_TRR1	Transmission Request Reset 1 Register
CAN_TA1	Transmission Acknowledge 1 Register
CAN_AA1	Abort Acknowledge 1 Register
CAN_RMP1	Receive Message Pending 1 Register
CAN_RML1	Receive Message Lost 1 Register
CAN_MBTIF1	Mailbox Transmit Interrupt Flag 1 Register
CAN_MBRIF1	Mailbox Receive Interrupt Flag 1 Register
CAN_MBIM1	Mailbox Interrupt Mask 1 Register
CAN_RFH1	Remote Frame Handling 1 Register
CAN_OPSS1	Overwrite Protection/Single Shot Transmission 1 Register
CAN_MC2	Mailbox Configuration 2 Register
CAN_MD2	Mailbox Direction 2 Register
CAN_TRS2	Transmission Request Set 2 Register
CAN_TRR2	Transmission Request Reset 2 Register
CAN_TA2	Transmission Acknowledge 2 Register
CAN_AA2	Abort Acknowledge 2 Register
CAN_RMP2	Receive Message Pending 2 Register

Table 21-7: ADSP-BF60x CAN Register List (Continued)

Name	Description
CAN_RML2	Receive Message Lost 2 Register
CAN_MBTIF2	Mailbox Transmit Interrupt Flag 2 Register
CAN_MBRIF2	Mailbox Receive Interrupt Flag 2 Register
CAN_MBIM2	Mailbox Interrupt Mask 2 Register
CAN_RFH2	Remote Frame Handling 2 Register
CAN_OPSS2	Overwrite Protection/Single Shot Transmission 2 Register
CAN_CLK	Clock Register
CAN_TIMING	Timing Register
CAN_DBG	Debug Register
CAN_STAT	Status Register
CAN_CEC	Error Counter Register
CAN_GIS	Global CAN Interrupt Status Register
CAN_GIM	Global CAN Interrupt Mask Register
CAN_GIF	Global CAN Interrupt Flag Register
CAN_CTL	CAN Master Control Register
CAN_INT	Interrupt Pending Register
CAN_MBTD	Temporary Mailbox Disable Register
CAN_EWR	Error Counter Warning Level Register
CAN_ESR	Error Status Register
CAN_UCCNT	Universal Counter Register
CAN_UCRC	Universal Counter Reload/Capture Register
CAN_UCCNF	Universal Counter Configuration Mode Register

Table 21-7: ADSP-BF60x CAN Register List (Continued)

Name	Description
CAN_AMnnL	Acceptance Mask (L) Register
CAN_AMnnH	Acceptance Mask (H) Register
CAN_MBnn_DATA0	Mailbox Word 0 Register
CAN_MBnn_DATA1	Mailbox Word 1 Register
CAN_MBnn_DATA2	Mailbox Word 2 Register
CAN_MBnn_DATA3	Mailbox Word 3 Register
CAN_MBnn_LENGTH	Mailbox Length Register
CAN_MBnn_TIMESTAMP	Mailbox Timestamp Register
CAN_MBnn_ID0	Mailbox ID 0 Register
CAN_MBnn_ID1	Mailbox ID 1 Register

Mailbox Configuration 1 Register

The CAN_MC1 register enables mailboxes 0 through 15. Each bit in this register enables or disables the corresponding mailbox. For all bits, set the bit (=1) to enable the mailbox, and clear the bit (=0) to disable the mailbox.

Enabling and disabling mailboxes has an impact on transmit requests. Setting the CAN_TRS1 bit associated with a disabled mailbox may result in erroneous behavior. Similarly, disabling a mailbox before the associated CAN_TRS1 bit is reset by the internal logic can cause unpredictable results.

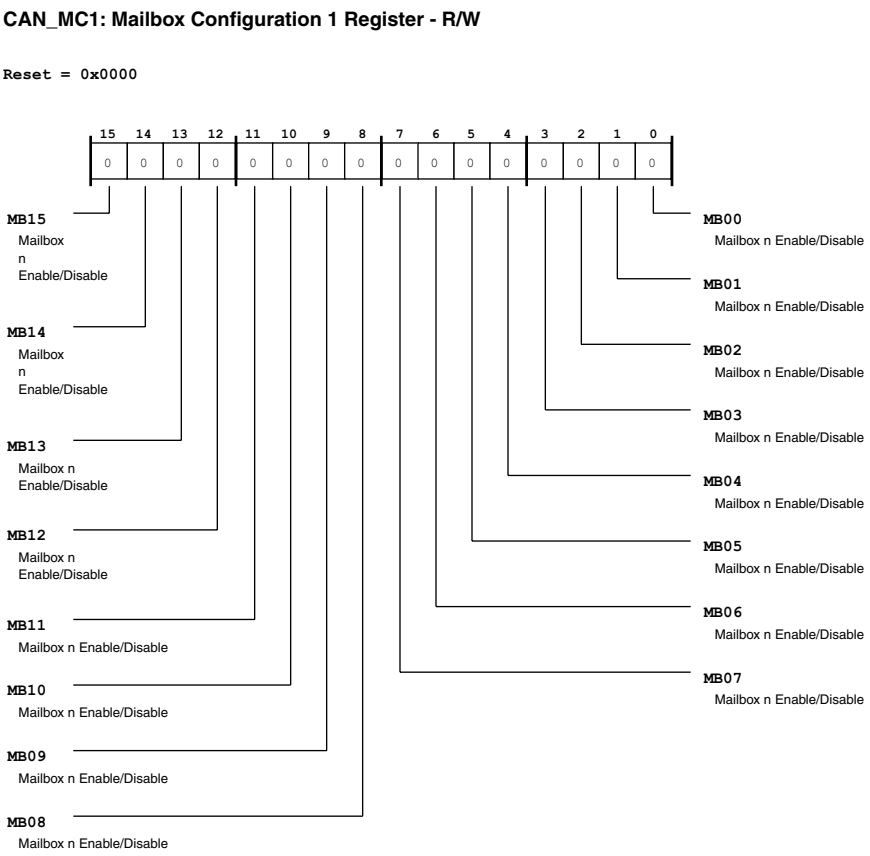


Figure 21-13: CAN_MC1 Register Diagram

Table 21-8: CAN_MC1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Enable/Disable.

Mailbox Direction 1 Register

The CAN_MD1 register selects the data transfer direction for mailboxes 0 through 15. Each bit in this register selects receive mode or transmit mode for the corresponding mailbox. For all bits, set the bit (=1) for receive mode from the mailbox, and clear the bit (=0) for transmit mode to the mailbox. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

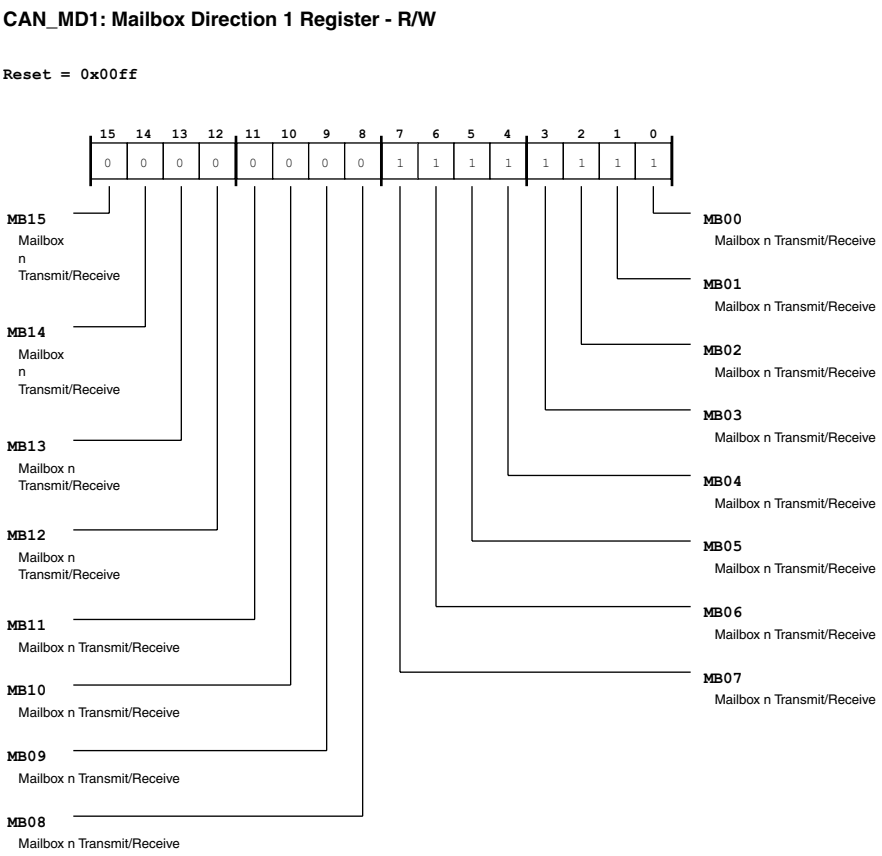


Figure 21-14: CAN_MD1 Register Diagram

Table 21-9: CAN_MD1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	MB	Mailbox n Transmit/Receive.
7:0 (R/NW)	MB	Mailbox n Transmit/Receive.

Transmission Request Set 1 Register

The CAN_TRS1 register requests transmit for mailboxes 8 through 15. Bits in this register request transmit for the corresponding mailbox when set (=1). After writing the data and the identifier into the mailbox area, the message is sent after mailbox n is enabled (with the corresponding bit in CAN_MC1 = 1}, and (subsequently) the corresponding transmit request bit is set (in CAN_TRS1). When a transmission completes, the corresponding bits in CAN_TRS1 and in the transmit request reset register (CAN_TRR1) are cleared. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

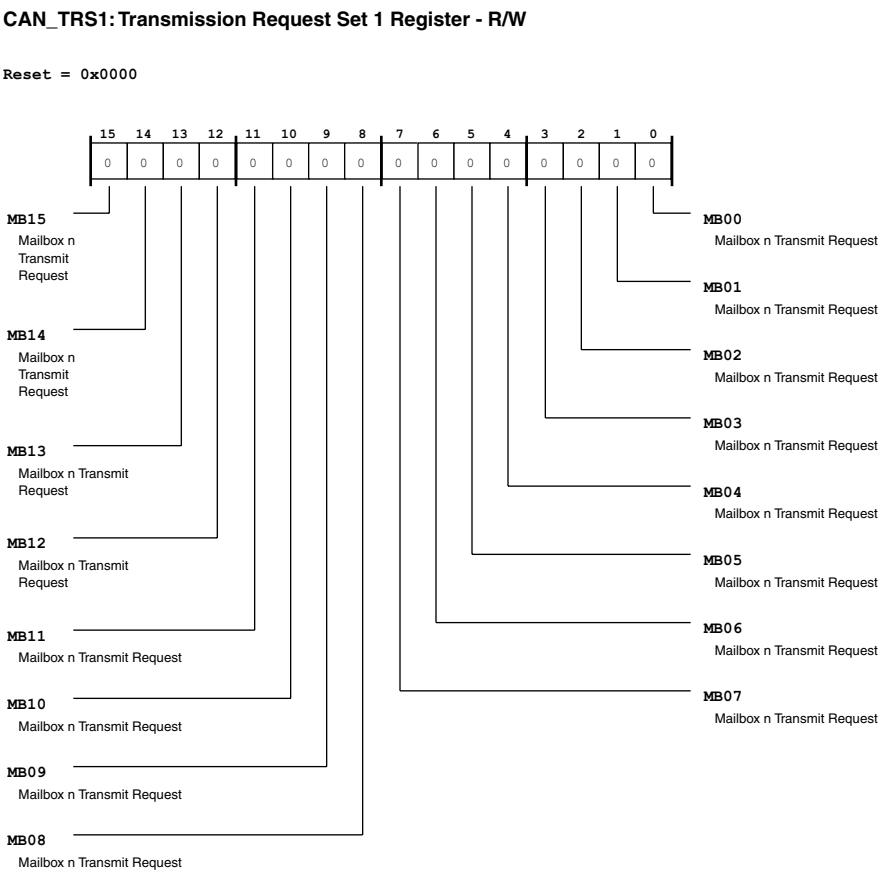


Figure 21-15: CAN_TRS1 Register Diagram

Table 21-10: CAN_TRS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	MB	Mailbox n Transmit Request.
7:0 (R/NW)	MB	Mailbox n Transmit Request.

Transmission Request Reset 1 Register

The CAN_TRR1 register requests transmit abort for mailboxes 8 through 15. Bits in this register request transmit abort for the corresponding mailbox when set (=1). When a transmission completes, the corresponding bits in the transmit request set register (CAN_TRS1) and in the CAN_TRR1 are cleared. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

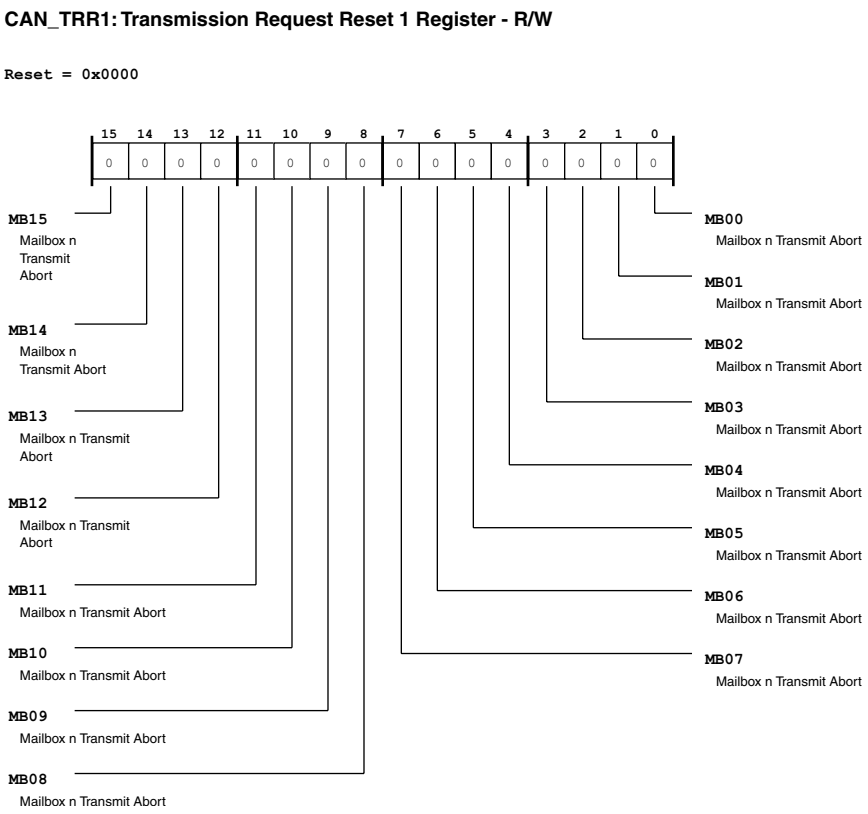


Figure 21-16: CAN_TRR1 Register Diagram

Table 21-11: CAN_TRR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	MB	Mailbox n Transmit Abort.
7:0 (R/NW)	MB	Mailbox n Transmit Abort.

Transmission Acknowledge 1 Register

The CAN_TA1 register indicates transmission success for mailboxes 8 through 15. Each bit in this register indicates transmission success for the corresponding mailbox when set (=1). Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

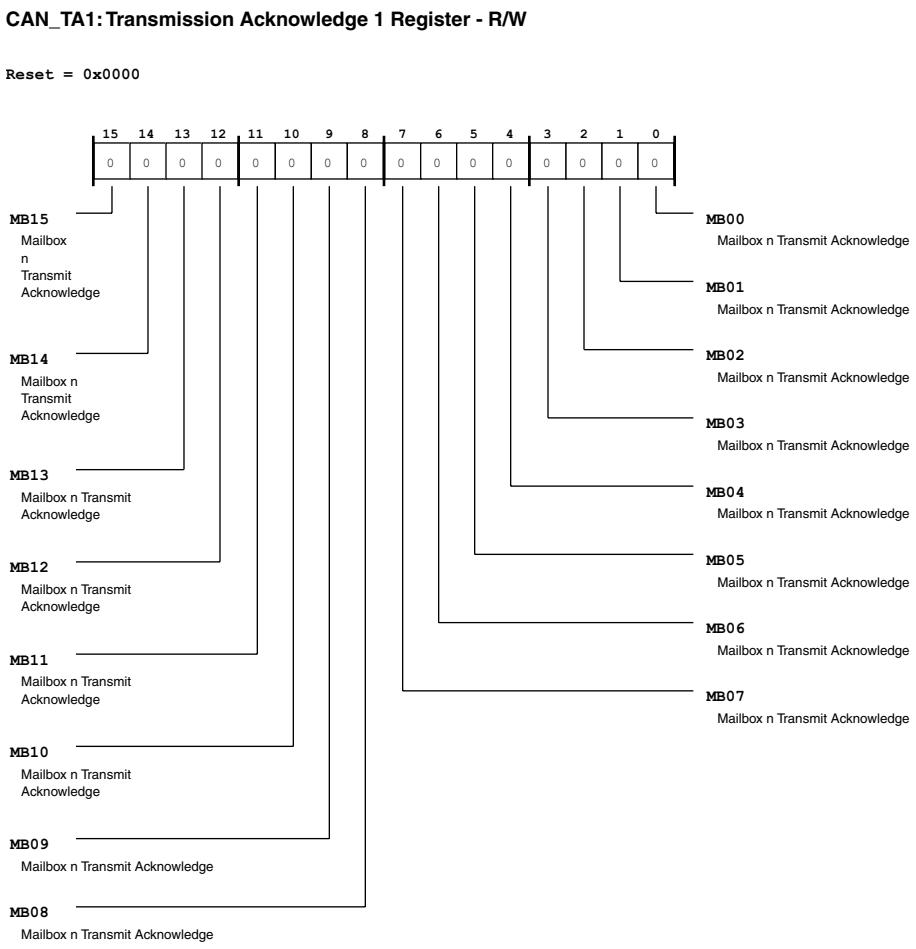


Figure 21-17: CAN_TA1 Register Diagram

Table 21-12: CAN_TA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W1C)	MB	Mailbox n Transmit Acknowledge.
7:0 (R/NW)	MB	Mailbox n Transmit Acknowledge.

Abort Acknowledge 1 Register

The `CAN_AA1` register indicates transmission abort (due to lost arbitration or a CAN error) for mailboxes 8 through 15. Each bit in this register indicates transmission abort for the corresponding mailbox when set (=1). Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

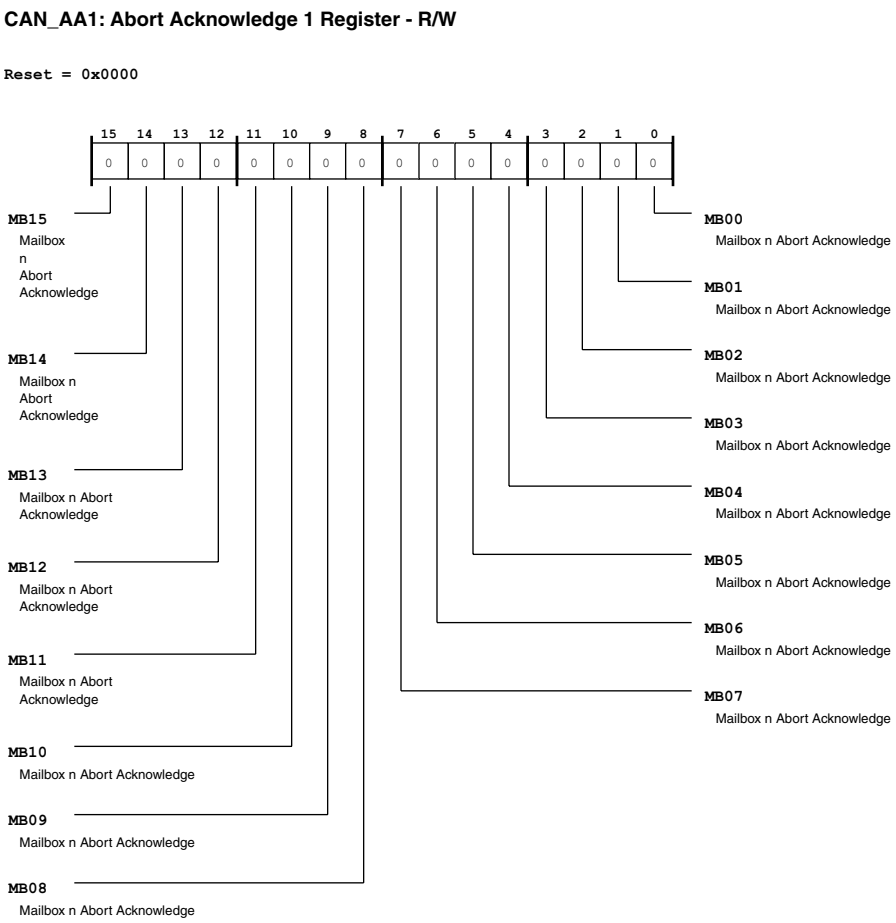


Figure 21-18: CAN_AA1 Register Diagram

Table 21-13: CAN_AA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W1C)	MB	Mailbox n Abort Acknowledge.
7:0 (R/NW)	MB	Mailbox n Abort Acknowledge.

Receive Message Pending 1 Register

The CAN_RMP1 register indicates when a message is pending (unread data) for mailboxes 0 through 15. Each bit in this register indicates the message pending status for the corresponding mailbox when set (=1).

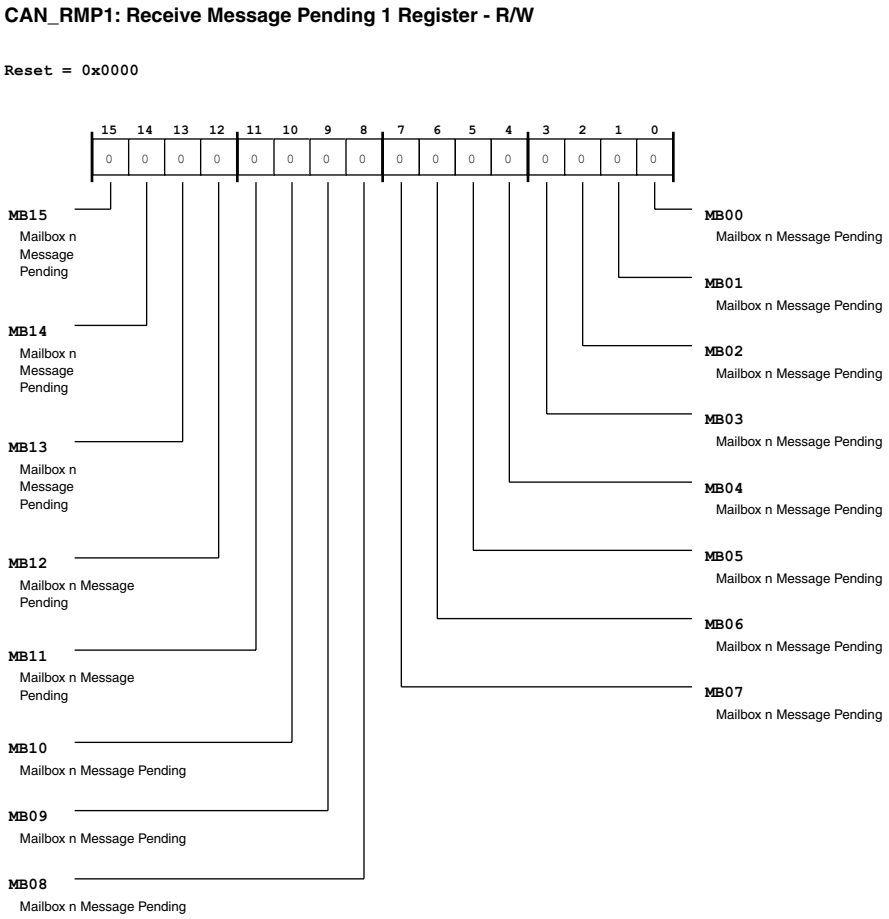


Figure 21-19: CAN_RMP1 Register Diagram

Table 21-14: CAN_RMP1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Message Pending.

Receive Message Lost 1 Register

The CAN_RML1 register indicates when a message is lost---due to a message coming while there is pending data (corresponding CAN_RMP1 bit set) and overwrite protection is disabled (CAN_OPSS1 bit cleared)---for mailboxes 0 through 15. Each bit in this register indicates the message lost status for the corresponding mailbox when set (=1).

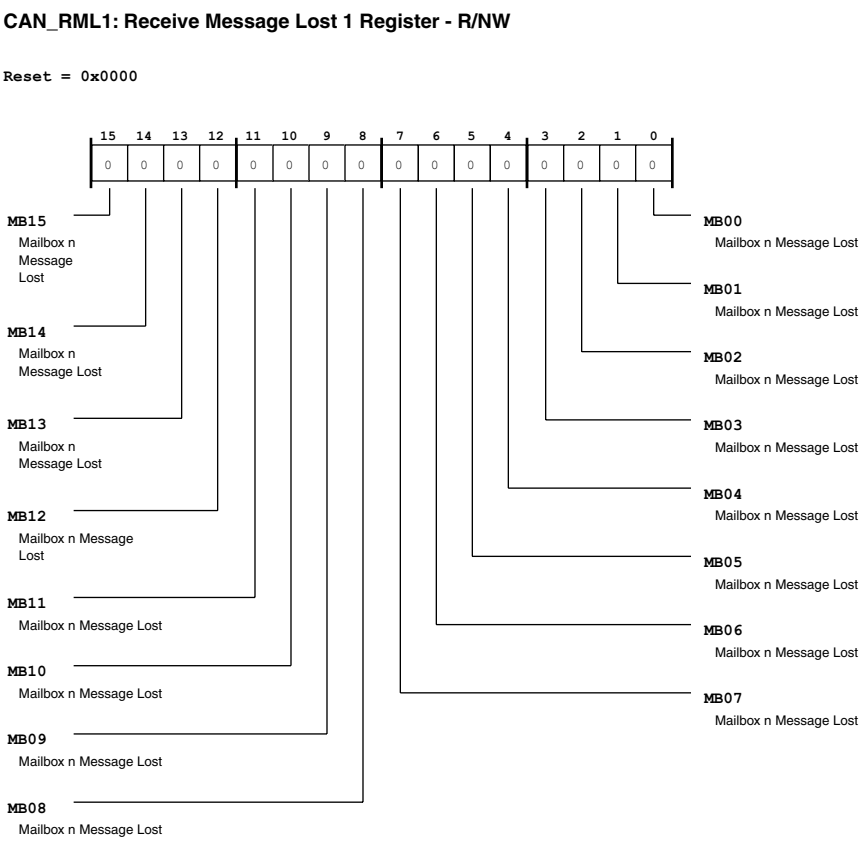


Figure 21-20: CAN_RML1 Register Diagram

Table 21-15: CAN_RML1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	MB	Mailbox n Message Lost.

Mailbox Transmit Interrupt Flag 1 Register

The CAN_MBTIF1 register indicates when a transmit interrupt is pending---due to successful transmission (corresponding CAN_TA1 bit set) and the interrupt is enabled (corresponding CAN_MBIM1 bit set)---for mailboxes 8 through 15. Each bit in this register indicates the transmit interrupt pending status for the corresponding mailbox when set (=1). When any bit in CAN_MBTIF1 is set, the CAN transmit interrupt request is raised (CAN_INT.MBTIRQ bit set). To clear the interrupt request, all of the set bits in CAN_MBTIF1 must be cleared by software (W1C). Also, software must clear the associated bits set in CAN_TA1 or set the associated bits in CAN_TRS1 bit to clear the interrupt source asserting the bits in CAN_MBTIF1. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

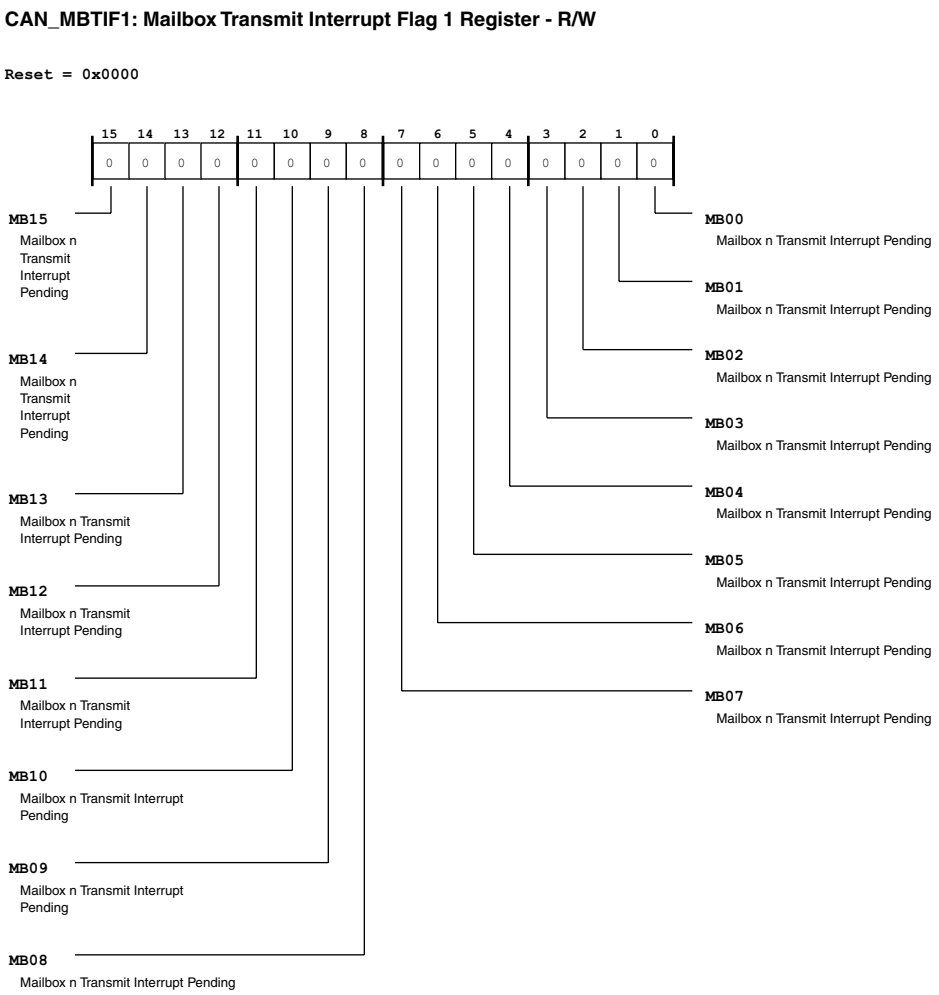


Figure 21-21: CAN_MBTIF1 Register Diagram

Table 21-16: CAN_MBTIF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W1C)	MB	Mailbox n Transmit Interrupt Pending.
7:0 (R/NW)	MB	Mailbox n Transmit Interrupt Pending.

Mailbox Receive Interrupt Flag 1 Register

The CAN_MBRIF1 register indicates when a receive interrupt is pending---due to successful reception (corresponding CAN_RMP1 bit set) and the interrupt is enabled (corresponding CAN_MBIM1 bit set)---for mailboxes 0 through 15. Each bit in this register indicates the receive interrupt pending status for the corre-

sponding mailbox when set (=1). When any bit in CAN_MBRIF1 is set, the CAN receive interrupt request is raised (CAN_INT.MBRIRQ bit set). To clear the interrupt request, all of the set bits in CAN_RMP1 must be cleared by software, then the associated bits set in CAN_MBRIF1 must be cleared (W1C).

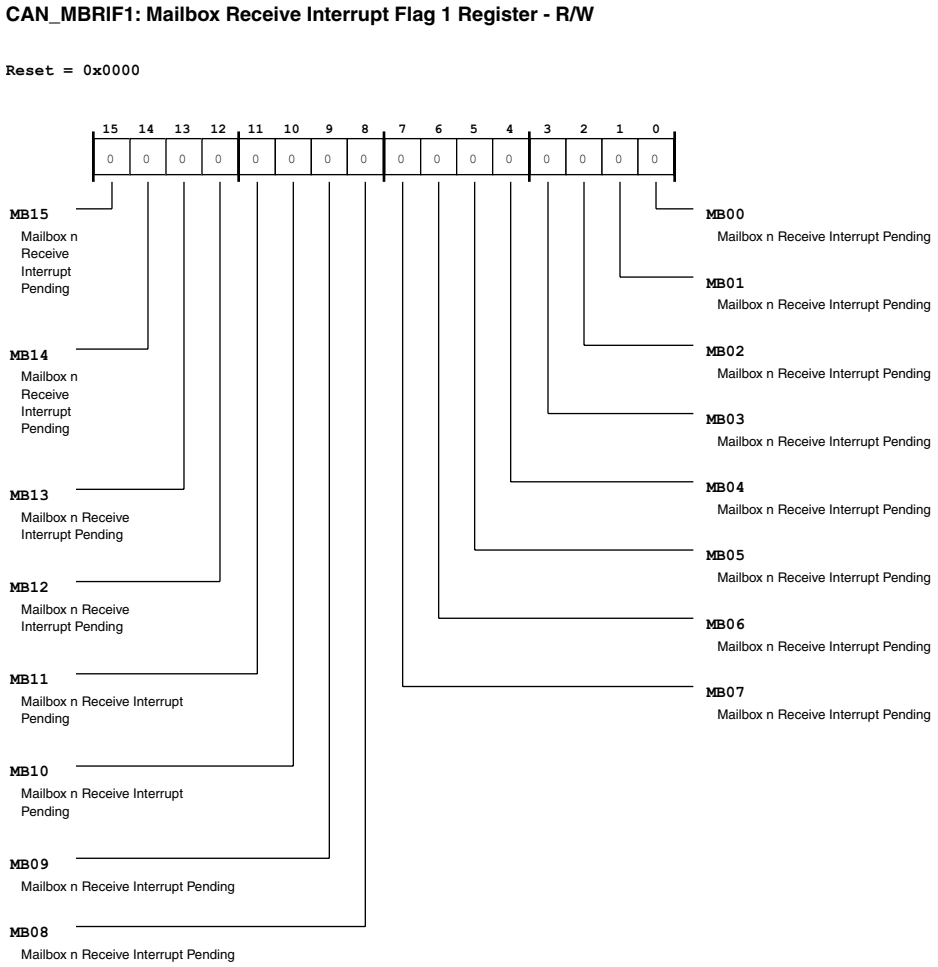


Figure 21-22: CAN_MBRIF1 Register Diagram

Table 21-17: CAN_MBRIF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Receive Interrupt Pending.

Mailbox Interrupt Mask 1 Register

The CAN_MBIM1 register enables transmit and receive interrupts for mailboxes 0 through 15. Each bit in this register requests enables the transmit or receive interrupt for the corresponding mailbox when set (=1).

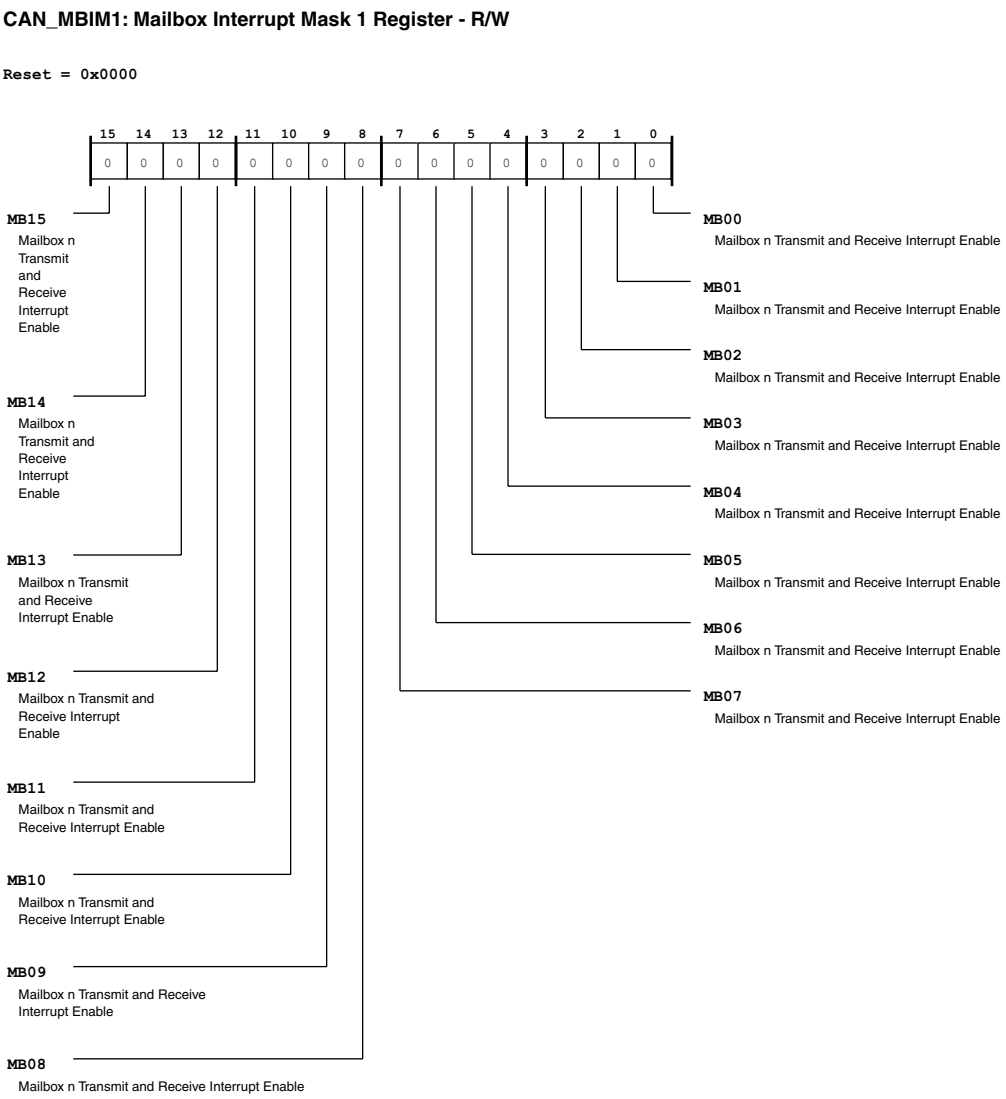


Figure 21-23: CAN_MBIM1 Register Diagram

Table 21-18: CAN_MBIM1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Transmit and Receive Interrupt Enable.

Remote Frame Handling 1 Register

The CAN_RFH1 register enables remote frame handling for mailboxes 8 through 15. Each bit in this register enables remote frame handling for the corresponding mailbox when set (=1). Note that enabling this bit affects transmit and receive operations for mailboxes. For more information about remote frame handling,

see the CAN Operating Modes sections, describing transmit and receive operations. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

CAN_RFH1: Remote Frame Handling 1 Register - R/W

Reset = 0x0000

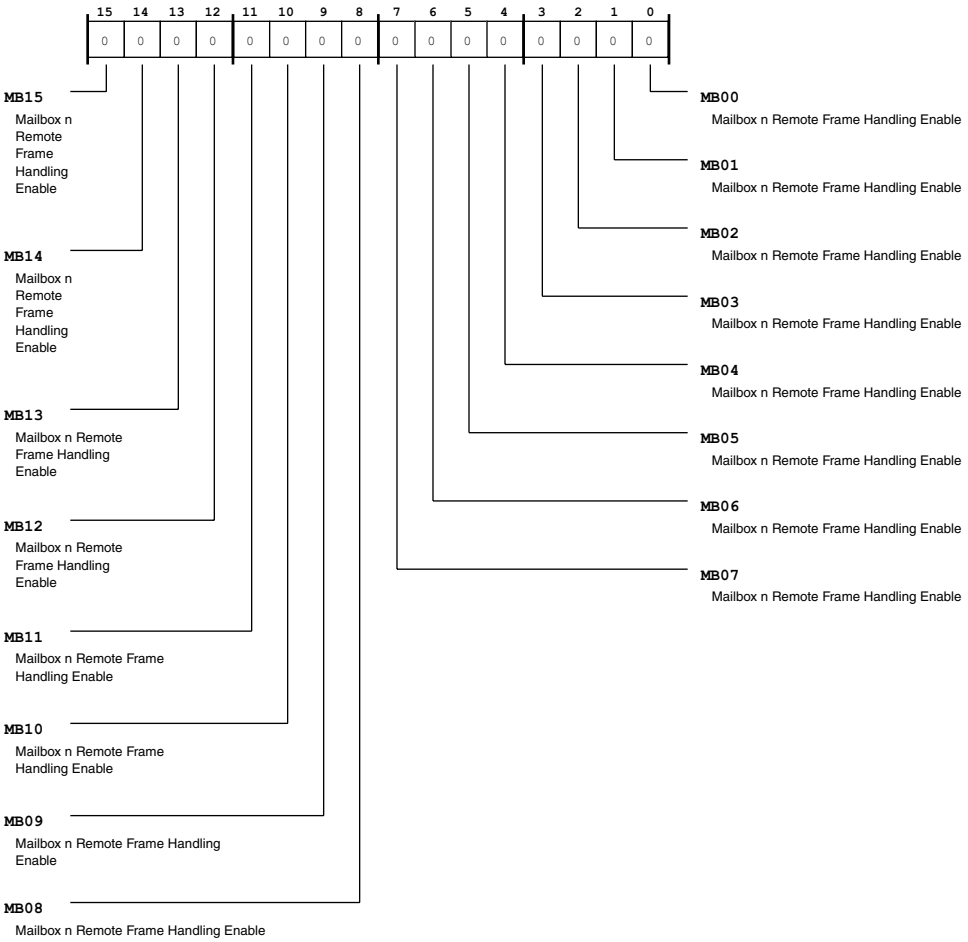


Figure 21-24: CAN_RFH1 Register Diagram

Table 21-19: CAN_RFH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	MB	Mailbox n Remote Frame Handling Enable.
7:0 (R/NW)	MB	Mailbox n Remote Frame Handling Enable.

Overwrite Protection/Single Shot Transmission 1 Register

The CAN_OPSS1 register enables overwrite protection for mailboxes 0 through 15. Each bit in this register enables overwrite protection for the corresponding mailbox when set (=1). Note that enabling this bit affects transmit and receive operations for mailboxes. For more information about remote overwrite protection, see the detailed feature description in the CAN Functional Description section. For more information about how this feature affects transmit and receive operations, see the CAN Operating Modes sections, describing transmit and receive operations.

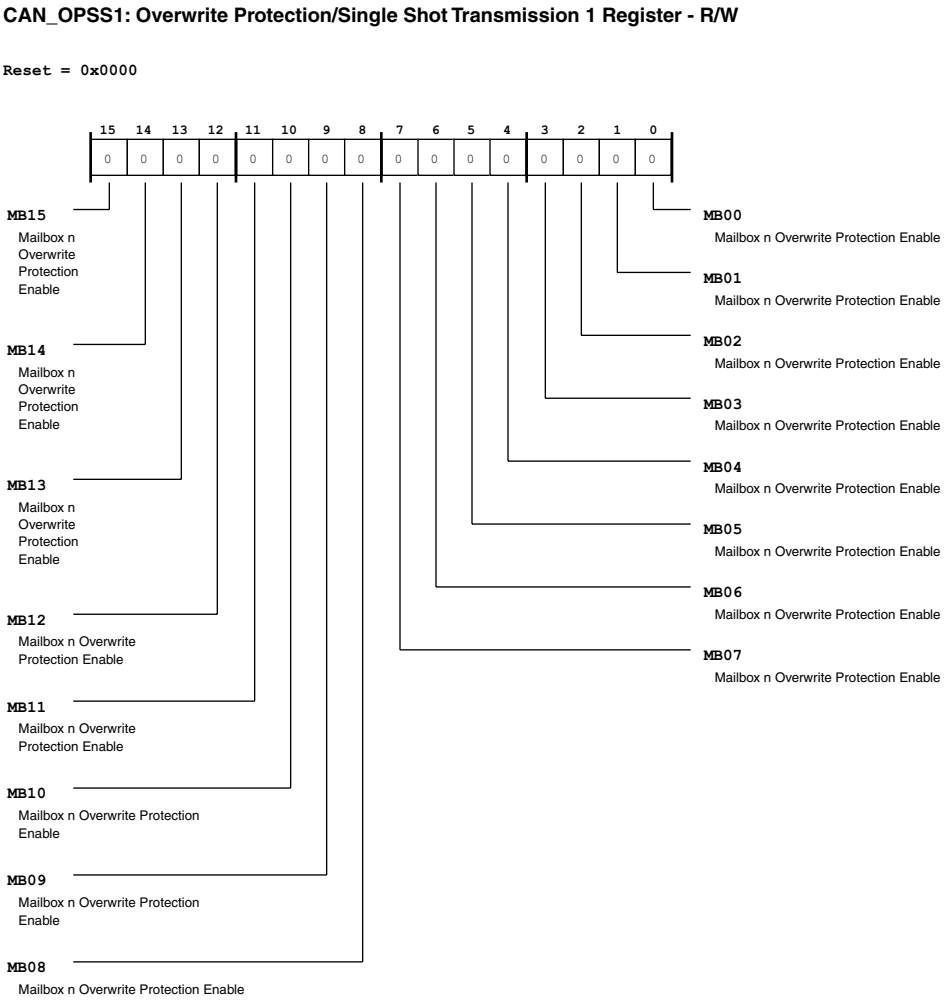


Figure 21-25: CAN_OPSS1 Register Diagram

Table 21-20: CAN_OPSS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Overwrite Protection Enable.

Mailbox Configuration 2 Register

The CAN_MC2 register enables mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register enables or disables the corresponding mailbox. For all bits, set the bit (=1) to enable the mailbox, and clear the bit (=0) to disable the mailbox.

Enabling and disabling mailboxes has an impact on transmit requests. Setting the CAN_TRS2 bit associated with a disabled mailbox may result in erroneous behavior. Similarly, disabling a mailbox before the associated CAN_TRS2 bit is reset by the internal logic can cause unpredictable results.

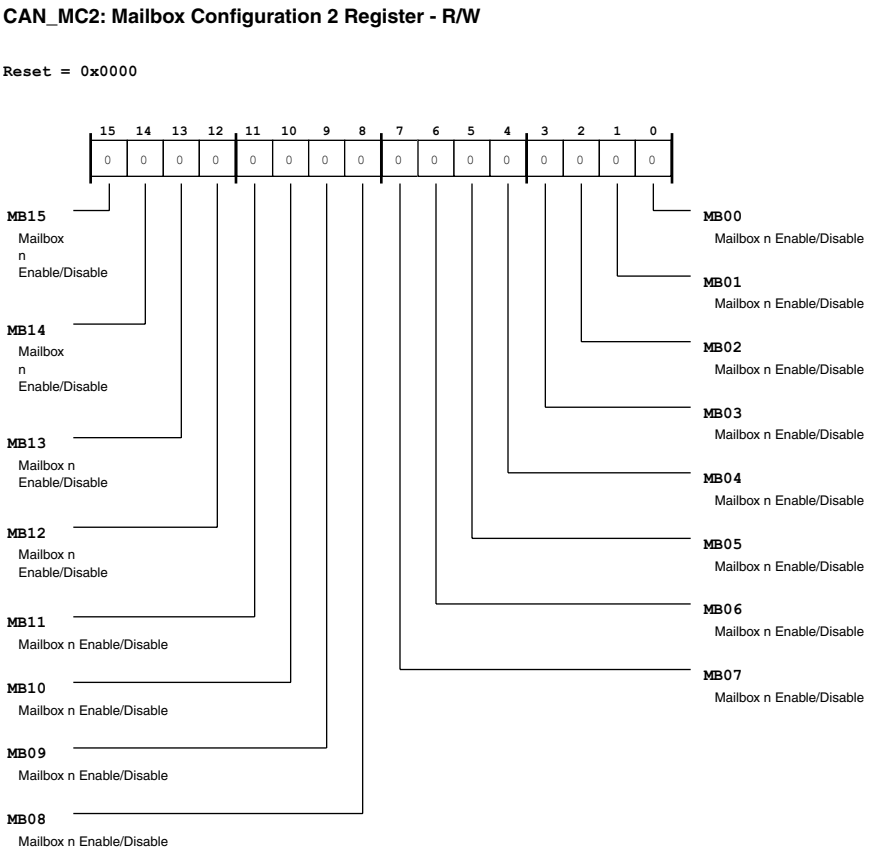


Figure 21-26: CAN_MC2 Register Diagram

Table 21-21: CAN_MC2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Enable/Disable.

Mailbox Direction 2 Register

The CAN_MD2 register selects the data transfer direction for mailboxes 16 (bit 0) through 23 (bit 7). Each bit in this register selects receive mode or transmit mode for the corresponding mailbox. For all bits, set the bit (=1) for receive mode from the mailbox, and clear the bit (=0) for transmit mode to the mailbox. Bits 8 through 15 are read-only, as the corresponding mailboxes (24 through 31) are transmit-only mailboxes.

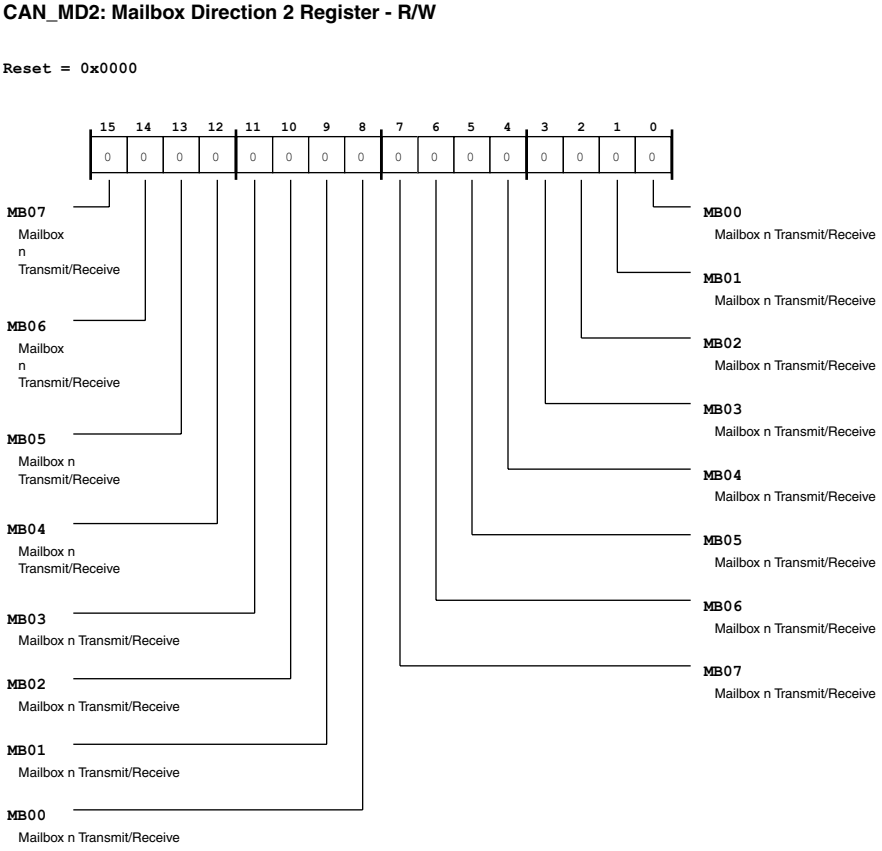


Figure 21-27: CAN_MD2 Register Diagram

Table 21-22: CAN_MD2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/NW)	MB	Mailbox n Transmit/Receive.
7:0 (R/W)	MB	Mailbox n Transmit/Receive.

Transmission Request Set 2 Register

The CAN_TRS2 register requests transmit for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register requests transmit for the corresponding mailbox when set (=1). After writing the data and the identifier into the mailbox area, the message is sent after mailbox n is enabled (with the corresponding bit in CAN_MC2 = 1}, and (subsequently) the corresponding transmit request bit is set (in CAN_TRS2). When a transmission completes, the corresponding bits in CAN_TRS2 and in the transmit request reset register (CAN_TRR2) are cleared.

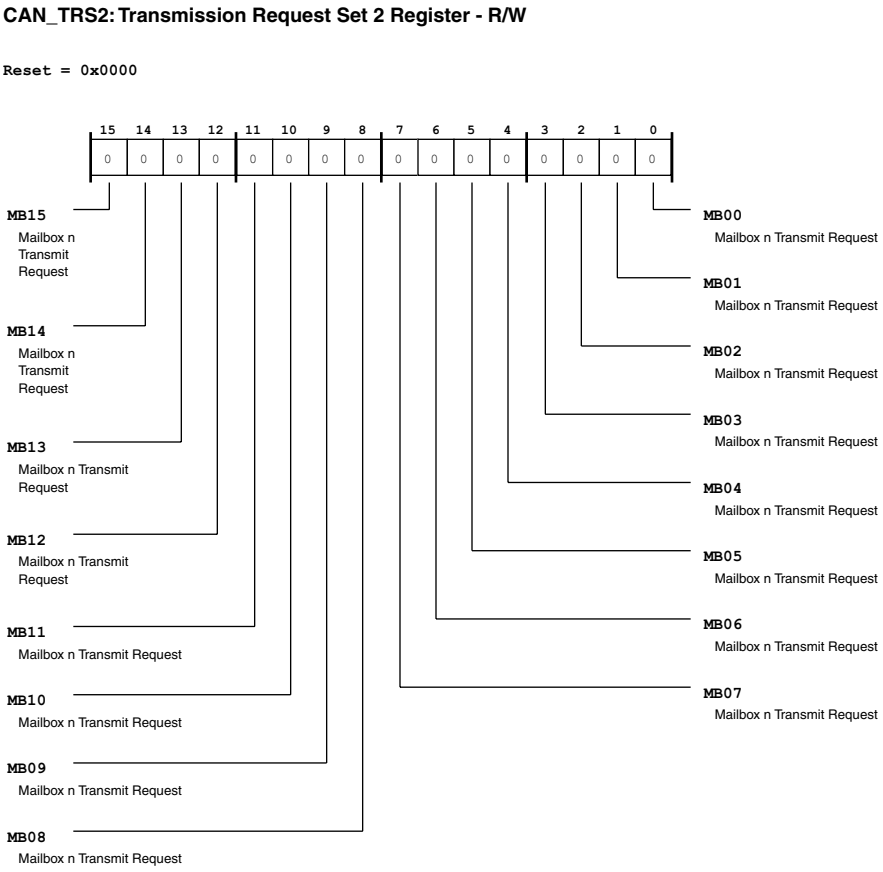


Figure 21-28: CAN_TRS2 Register Diagram

Table 21-23: CAN_TRS2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Transmit Request.

Transmission Request Reset 2 Register

The CAN_TRR2 register requests transmit abort for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register requests transmit abort for the corresponding mailbox when set (=1). When a transmission completes, the corresponding bits in the transmit request set register (CAN_TRS2) and in the CAN_TRR2 are cleared.

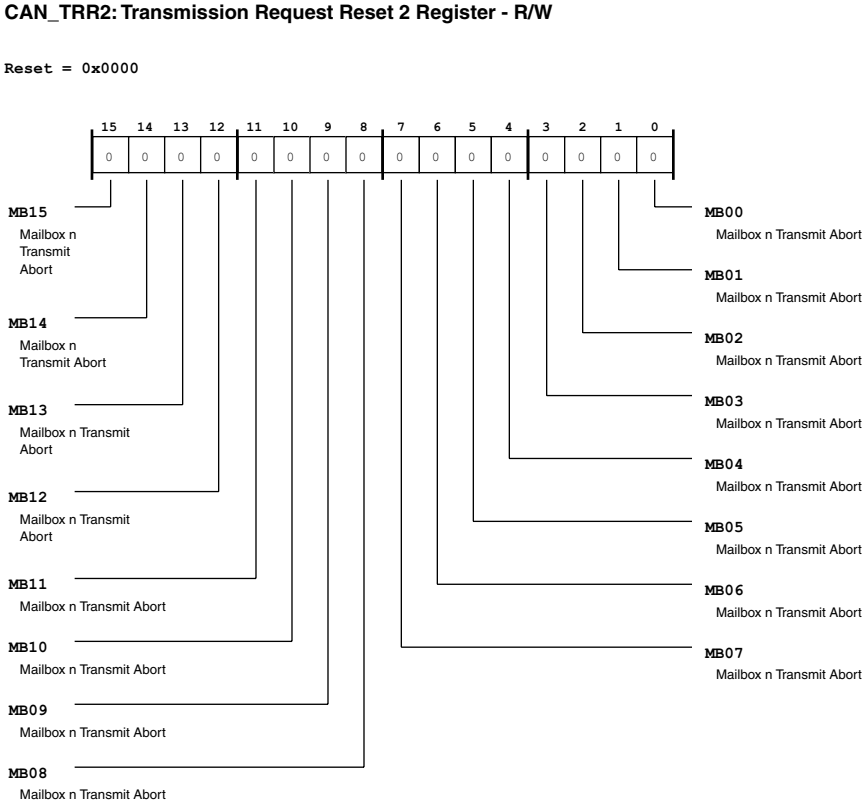


Figure 21-29: CAN_TRR2 Register Diagram

Table 21-24: CAN_TRR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Transmit Abort.

Transmission Acknowledge 2 Register

The CAN_TA2 register indicates transmission success for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register indicates transmission success for the corresponding mailbox when set (=1).

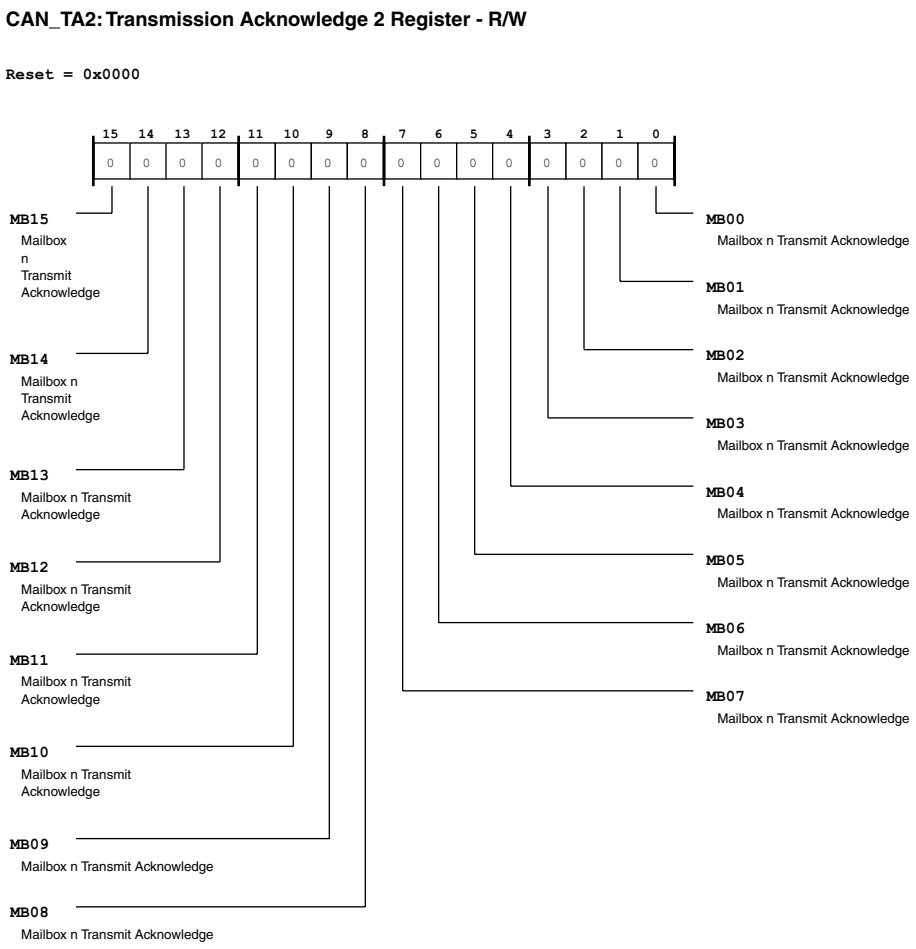


Figure 21-30: CAN_TA2 Register Diagram

Table 21-25: CAN_TA2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Transmit Acknowledge.

Abort Acknowledge 2 Register

The CAN_AA2 register indicates transmission abort (due to lost arbitration or a CAN error) for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register indicates transmission abort for the corresponding mailbox when set (=1).

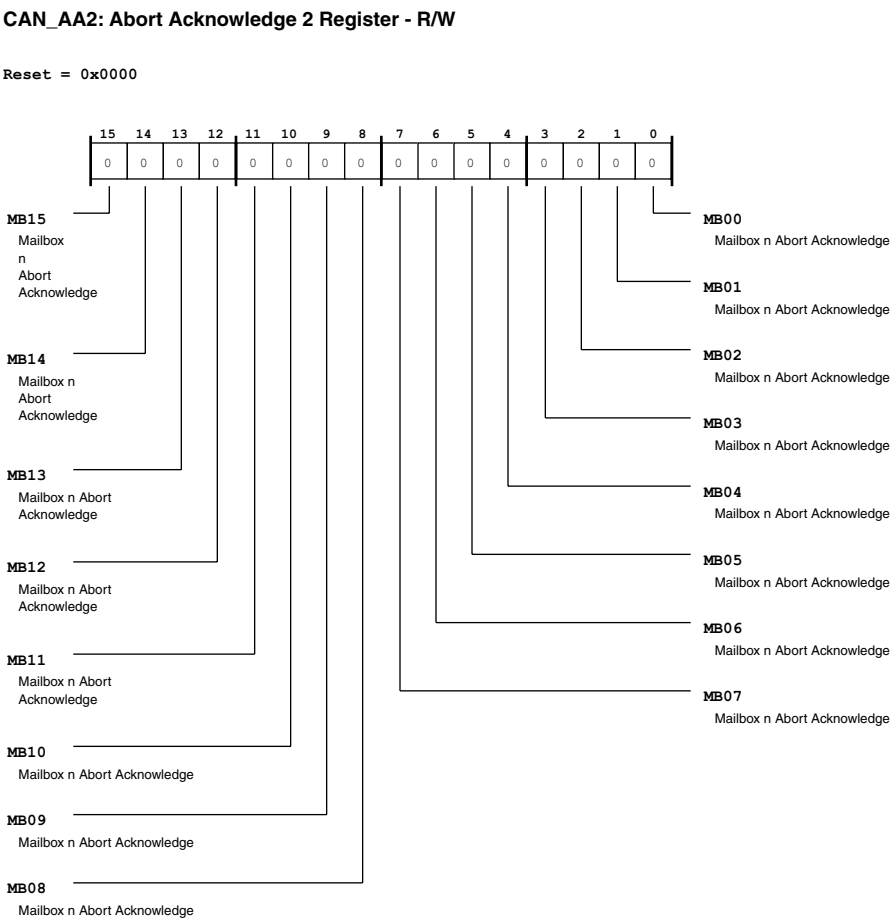


Figure 21-31: CAN_AA2 Register Diagram

Table 21-26: CAN_AA2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Abort Acknowledge.

Receive Message Pending 2 Register

The CAN_RMP2 register indicates when a message is pending (unread data) for mailboxes 16 (bit 0) through 23 (bit 7). Each bit in this register indicates the message pending status for the corresponding mailbox when set (=1). Bits 8 through 15 are reserved, as the corresponding mailboxes (24 through 31) are transmit-only mailboxes.

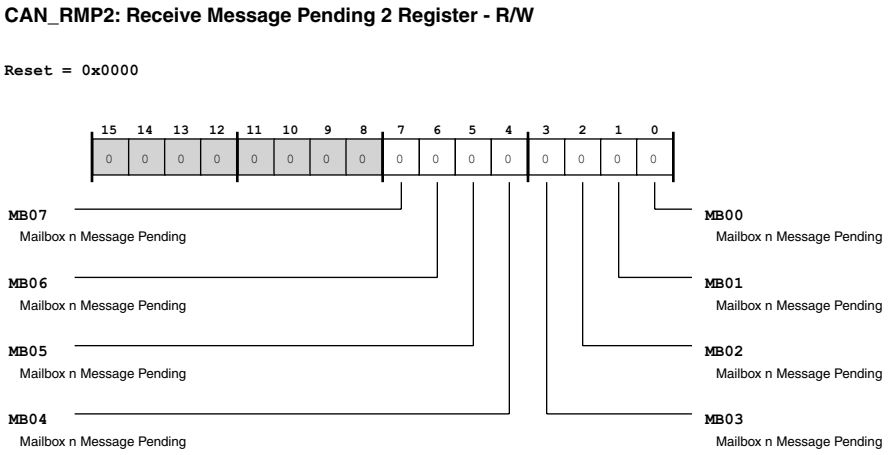


Figure 21-32: CAN_RMP2 Register Diagram

Table 21-27: CAN_RMP2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W1C)	MB	Mailbox n Message Pending.

Receive Message Lost 2 Register

The CAN_RML2 register indicates when a message is lost---due to a message coming while there is pending data (corresponding CAN_RMP2 bit set) and overwrite protection is disabled (CAN_OPSS2 bit cleared)---for mailboxes 16 (bit 0) through 23 (bit 7). Each bit in this register indicates the message lost status for the corresponding mailbox when set (=1). Bits 8 through 15 are reserved, as the corresponding mailboxes (24 through 31) are transmit-only mailboxes.

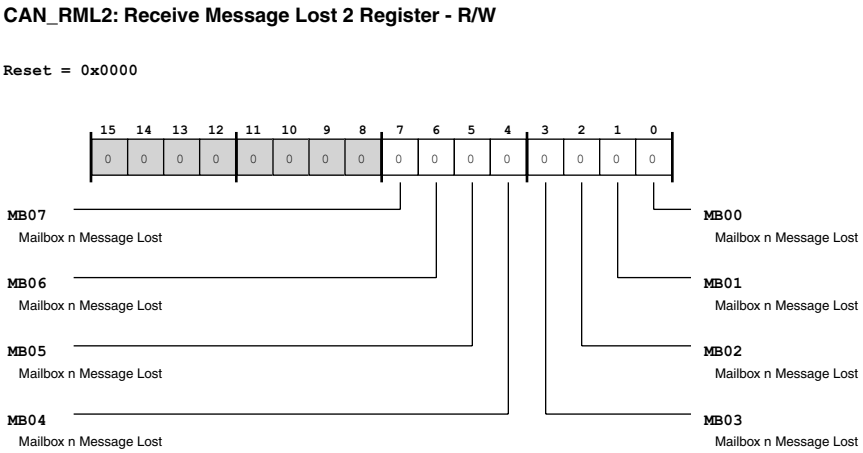


Figure 21-33: CAN_RML2 Register Diagram

Table 21-28: CAN_RML2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	MB	Mailbox n Message Lost.

Mailbox Transmit Interrupt Flag 2 Register

The CAN_MBTIF2 register indicates when a transmit interrupt is pending---due to successful transmission (corresponding CAN_TA2 bit set) and the interrupt is enabled (corresponding CAN_MBIM2 bit set)---for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register indicates the transmit interrupt pending status for the corresponding mailbox when set (=1). When any bit in CAN_MBTIF2 is set, the CAN transmit interrupt request is raised (CAN_INT.MBTIRQ bit set). To clear the interrupt request, all of the set bits in CAN_MBTIF2 must be cleared by software (W1C). Also, software must clear the associated bits set in CAN_TA2 or set the associated bits in CAN_TRS2 bit to clear the interrupt source asserting the bits in CAN_MBTIF2.

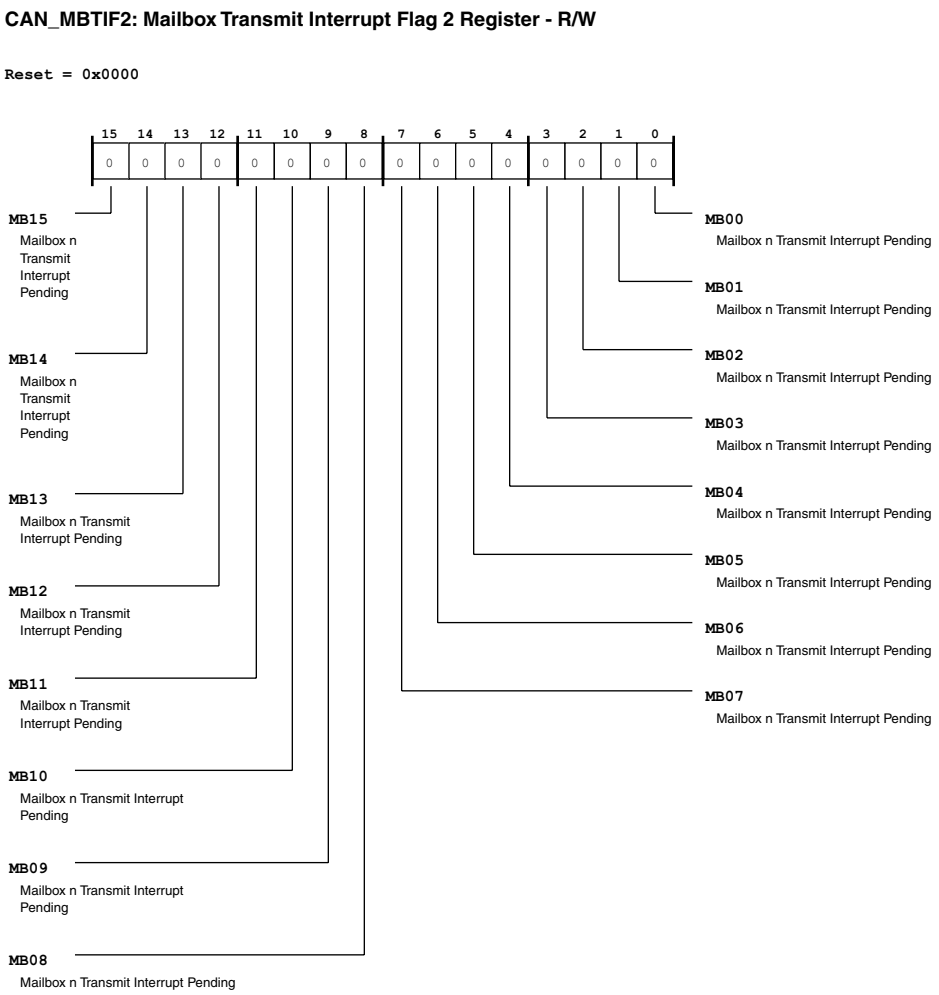


Figure 21-34: CAN_MBTIF2 Register Diagram

Table 21-29: CAN_MBTIF2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Transmit Interrupt Pending.

Mailbox Receive Interrupt Flag 2 Register

The CAN_MBRIF2 register indicates when a receive interrupt is pending---due to successful reception (corresponding CAN_RMP2 bit set) and the interrupt is enabled (corresponding CAN_MBIM2 bit set)---for mailboxes 16 (bit 0) through 23 (bit 7). Each bit in this register indicates the receive interrupt pending status for the corresponding mailbox when set (=1). When any bit in CAN_MBRIF2 is set, the CAN receive interrupt request is raised (CAN_INT.MBRIRQ bit set). To clear the interrupt request, all of the set bits in

CAN_RMP2 must be cleared by software, then the associated bits set in CAN_MBRIF2 must be cleared (W1C). Bits 8 through 15 are reserved and read-only, as the corresponding mailboxes (24 through 31) are transmit-only mailboxes.

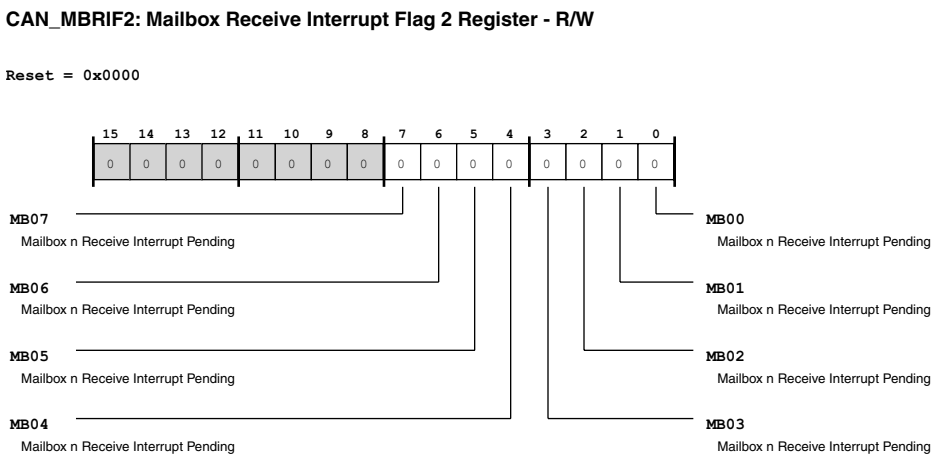


Figure 21-35: CAN_MBRIF2 Register Diagram

Table 21-30: CAN_MBRIF2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W1C)	MB	Mailbox n Receive Interrupt Pending.

Mailbox Interrupt Mask 2 Register

The CAN_MBIM2 register enables transmit and receive interrupts for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register requests enables the transmit or receive interrupt for the corresponding mailbox when set (=1).

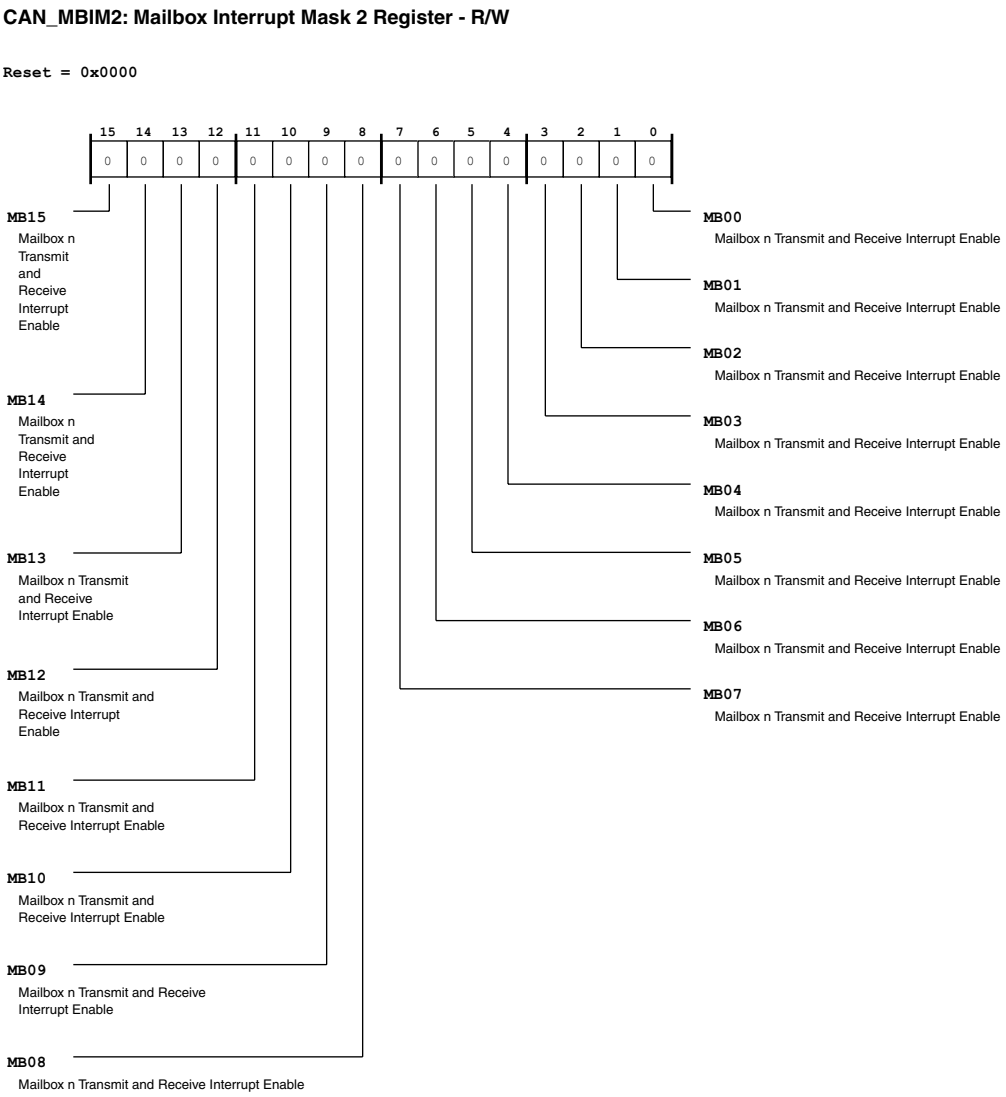


Figure 21-36: CAN_MBIM2 Register Diagram

Table 21-31: CAN_MBIM2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Transmit and Receive Interrupt Enable.

Remote Frame Handling 2 Register

The CAN_RFH2 register enables remote frame handling for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register enables remote frame handling for the corresponding mailbox when set (=1). Note that

enabling this bit affects transmit and receive operations for mailboxes. For more information about remote frame handling, see the CAN Operating Modes sections, describing transmit and receive operations.

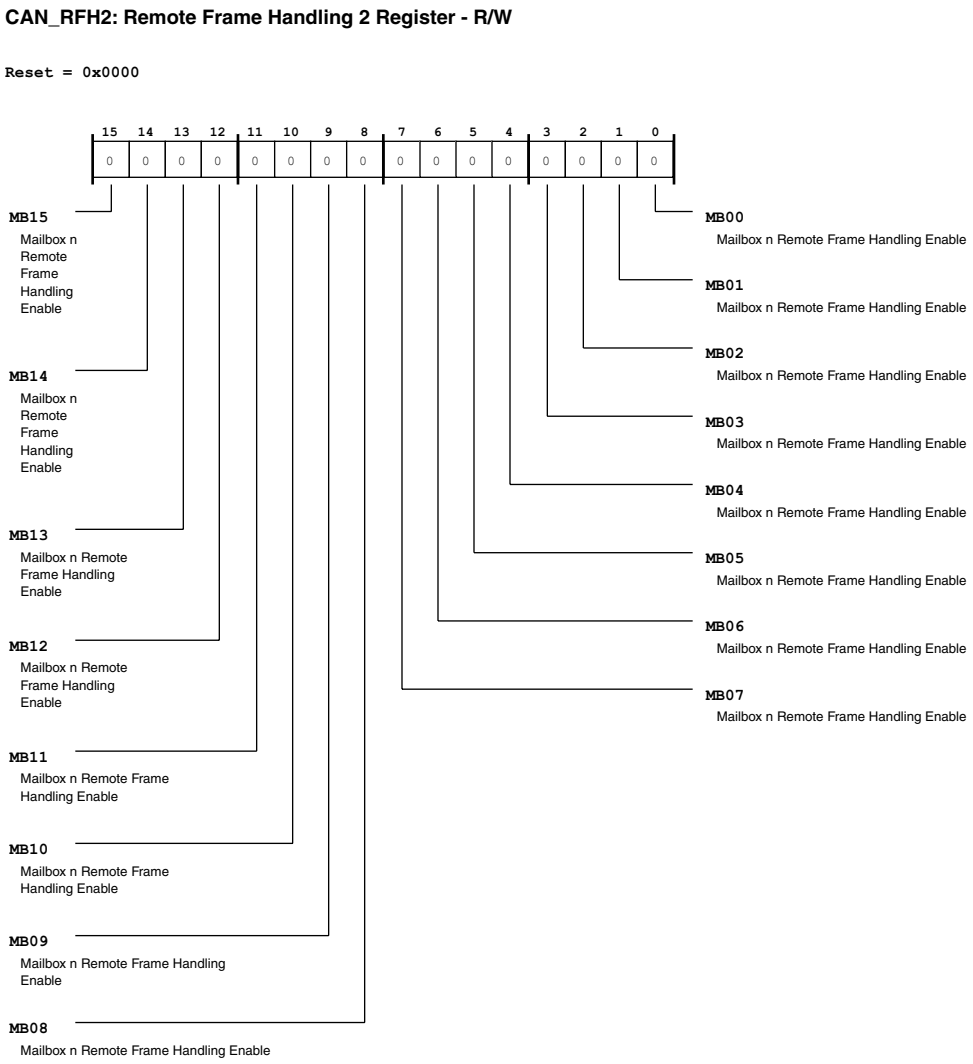


Figure 21-37: CAN_RFH2 Register Diagram

Table 21-32: CAN_RFH2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Remote Frame Handling Enable.

Overwrite Protection/Single Shot Transmission 2 Register

The CAN_OPSS2 register enables overwrite protection for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register enables overwrite protection for the corresponding mailbox when set (=1). Note that enabling this bit affects transmit and receive operations for mailboxes. For more information about remote overwrite protection, see the detailed feature description in the CAN Functional Description section. For more information about how this feature affects transmit and receive operations, see the CAN Operating Modes sections, describing transmit and receive operations.

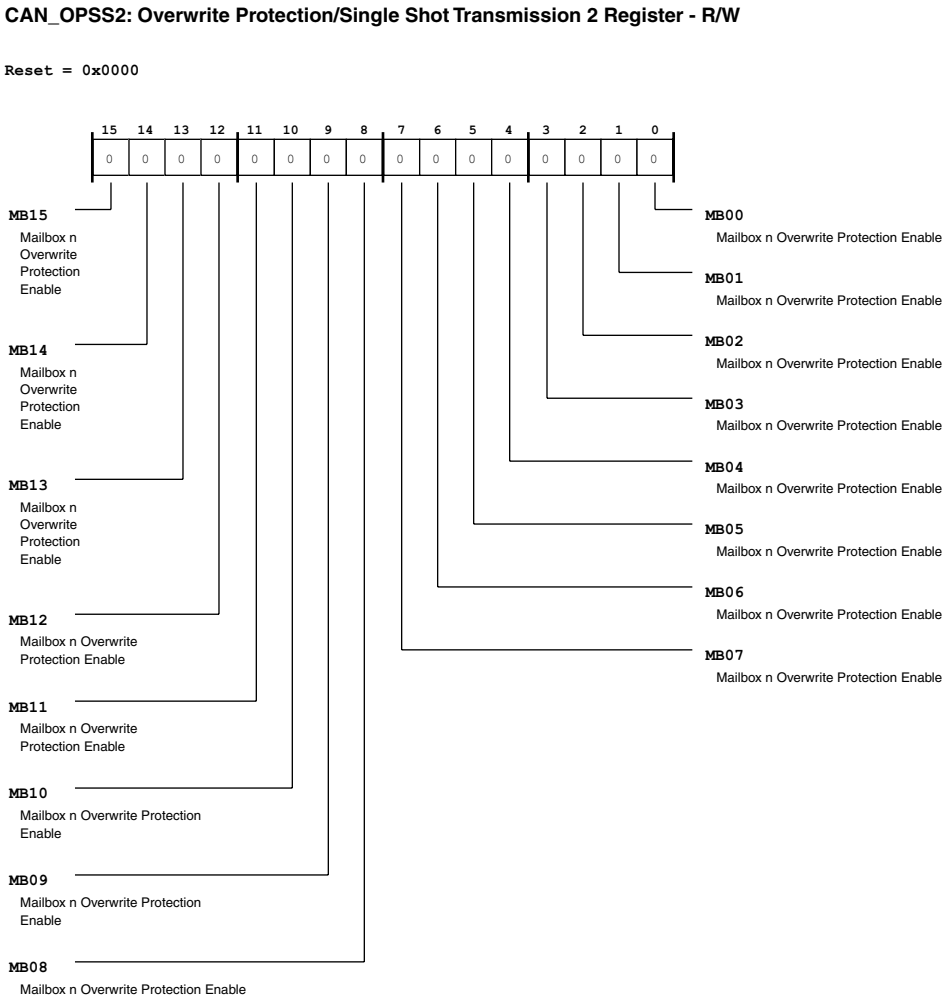


Figure 21-38: CAN_OPSS2 Register Diagram

Table 21-33: CAN_OPSS2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Overwrite Protection Enable.

Clock Register

The CAN_CLK register select the bit rate prescaler for calculating the time quantum (TQ), which is used to derive the CAN clock from the system clock (SCLK). For more information about bit timing and clock operation, see the CAN Operating Modes section.

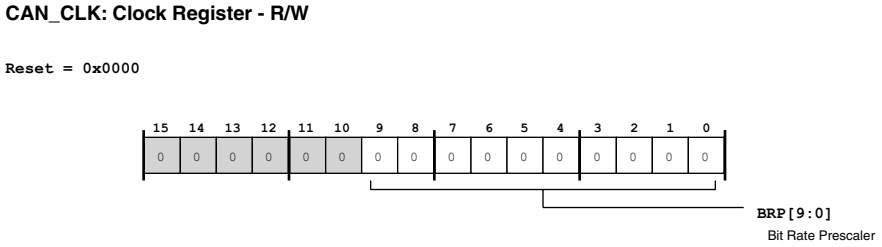


Figure 21-39: CAN_CLK Register Diagram

Table 21-34: CAN_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	BRP	Bit Rate Prescaler. The CAN_CLK.BRP bits select the bit rate prescaler value, which is used to calculate the time quantum for CAN bit timing. The formula using CAN_CLK.BRP to calculate the time quantum is: TQ = (BRP+1) / SCLK Note that it is recommended that the CAN_CLK.BRP value be greater than or equal to 4. For more information about bit timing, see the Operating Modes section.

Timing Register

The CAN_TIMING register select the time segments, sampling, and synchronization for CAN bit timing. For more information about bit timing and clock operation, see the CAN Operating Modes section.

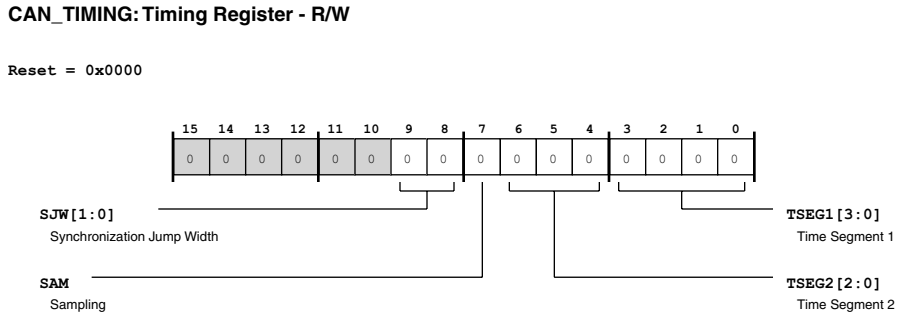


Figure 21-40: CAN_TIMING Register Diagram

Table 21-35: CAN_TIMING Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:8 (R/W)	SJW	Synchronization Jump Width. The CAN_TIMING.SJW bits select the maximum number of time quanta, ranging from 1 to 4(SJW + 1). This selection allows for a re-synchronization attempt when the CAN detects a recessive-to-dominant edge outside the synchronization segment. The re-synchronization automatically moves the sampling point such that the CAN bit is still handled properly. Note that the CAN_TIMING.SJW value should not exceed CAN_TIMING.TSEG2 or CAN_TIMING.TSEG1.
7 (R/W)	SAM	Sampling. The CAN_TIMING.SAM bit selects whether the CAN performs normal sampling (once at the sampling point described by the CAN_TIMING register) or performs over sampling. If CAN_TIMING.SAM is set, the CAN over samples the input signal at three times at the SCLK rate. The resulting value is generated by a majority decision of the three sample values. Note that the CAN_TIMING.SAM bit should always be cleared if the CAN_CLK.BRP value is less than 4.
6:4 (R/W)	TSEG2	Time Segment 2. The CAN_TIMING.TSEG2 bits and CAN_TIMING.TSEG1 bits control how many time quanta of which the CAN bits consist, resulting in the CAN bit rate. For more information about bit timing and clock operation, see the CAN Operating Modes section. Note that the CAN_TIMING.TSEG1 value should always be greater than or equal to the CAN_TIMING.TSEG2 value.
3:0 (R/W)	TSEG1	Time Segment 1. The CAN_TIMING.TSEG1 bits and CAN_TIMING.TSEG2 bits control how many time quanta of which the CAN bits consist, resulting in the CAN bit rate. For more information about bit timing and clock operation, see the CAN Operating Modes section. Note that the CAN_TIMING.TSEG1 value should always be greater than or equal to the CAN_TIMING.TSEG2 value.

Debug Register

The CAN_DBG register controls CAN debug modes, including CAN_TX and CAN_RX pin enable/disable.

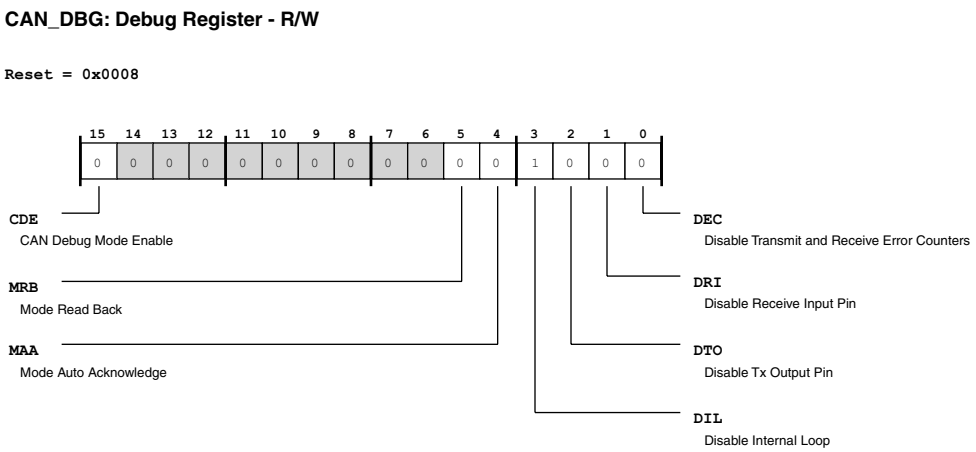


Figure 21-41: CAN_DBG Register Diagram

Table 21-36: CAN_DBG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	CDE	CAN Debug Mode Enable. The CAN_DBG.CDE bit enables debug mode. This bit must be written first before subsequent writes to the CAN_DBG register. When the CAN_DBG.CDE bit is cleared, all CAN debug features are disabled.	
		0	Disable Debug Mode
		1	Enable Debug Mode
5 (R/W)	MRB	Mode Read Back. The CAN_DBG.MRB bit enables read back mode. When enabled, a message transmitted on the CAN bus or through an internal loop back mode is received back directly to the internal receive buffer.	
		0	Disable Read Back Mode
		1	Enable Read Back Mode
4 (R/W)	MAA	Mode Auto Acknowledge. The CAN_DBG.MAA bit enables mode auto acknowledge, allowing the CAN to generate its own acknowledge during the ACK slot of the CAN frame. The CAN_DBG.MAA acknowledge appears on the CAN_TX pin if CAN_DBG.DIL =1 and CAN_DBG.DTO =0. If the acknowledge is only going to be used internally, these test mode bits should be set to CAN_DBG.DIL = 0 and CAN_DBG.DTO =1.	
		0	Disable Auto Acknowledge Mode
		1	Enable Auto Acknowledge Mode

Table 21-36: CAN_DBG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	DIL	Disable Internal Loop. The CAN_DBG.DIL bit disables internal loop mode, which routes the transmit output to the receive input.	
		0	Enable Internal Loop
		1	Disable Internal Loop
2 (R/W)	DTO	Disable Tx Output Pin. The CAN_DBG.DTO bit disables the CAN_TX pin.	
		0	Enable Tx Output Pin
		1	Disable Tx Output Pin, Drive Recessive
1 (R/W)	DRI	Disable Receive Input Pin. The CAN_DBG.DRI bit disables the CAN_RX pin.	
		0	Enable Rx Input Pin
		1	Disable Rx Input Pin, Drive Recessive Internally
0 (R/W)	DEC	Disable Transmit and Receive Error Counters. The CAN_DBG.DEC bit disables the transmit and receive error counters in the CAN_CEC register. When set, the CAN_CEC holds its current contents and is not allowed to increment or decrement the error counters. Note that this mode does not conform to the CAN specification.	
		0	Enable CEC Tx and Rx Error Counters
		1	Disable CEC Tx and Rx Error Counters

Status Register

The CAN_STAT register indicates status for CAN modes and error conditions.

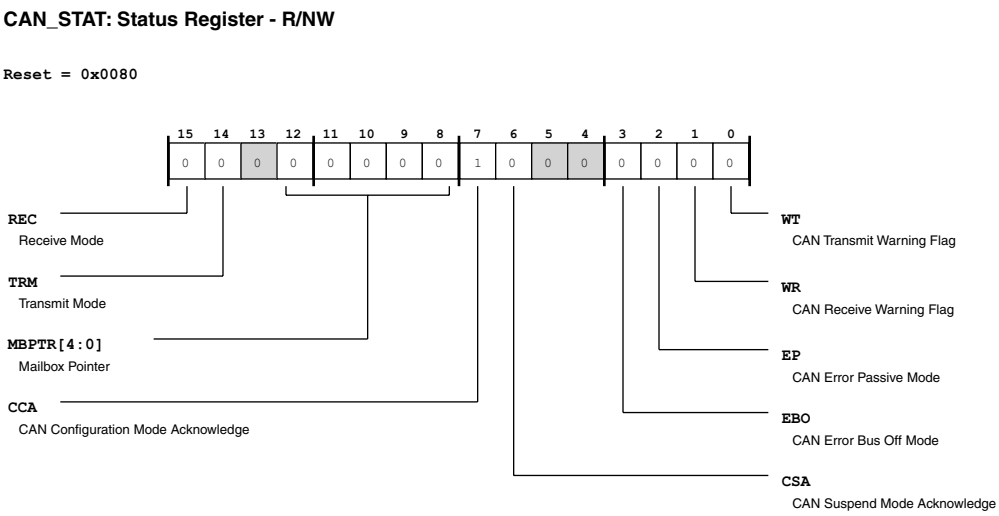


Figure 21-42: CAN_STAT Register Diagram

Table 21-37: CAN_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/NW)	REC	Receive Mode. The CAN_STAT.REC bit indicates whether the CAN is in receive mode.	
		0	Not in Receive Mode
		1	Receive Mode
14 (R/NW)	TRM	Transmit Mode. The CAN_STAT.TRM bit indicates whether the CAN is in transmit mode.	
		0	Not in Transmit Mode
		1	Transmit Mode
12:8 (R/NW)	MBPTR	Mailbox Pointer. The CAN_STAT.MBPTR bits represent the mailbox number of the current transmit message. After a successful transmission, these bits remain unchanged.	
		0	Processing Mailbox 0 Message
		...	...
		31	Processing Mailbox 31 Message

Table 21-37: CAN_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	CCA	CAN Configuration Mode Acknowledge. The CAN_STAT.CCA bit indicates whether the CAN is in configuration mode.	
		0	Not in Configuration Mode
		1	Configuration mode
6 (R/NW)	CSA	CAN Suspend Mode Acknowledge. The CAN_STAT.CSA bit indicates whether the CAN is in suspend mode.	
		0	Not in Suspend Mode
		1	Suspend mode
3 (R/NW)	EBO	CAN Error Bus Off Mode. The CAN_STAT.EBO bit indicates whether the CAN is in error bus off mode.	
		0	TXECNT Below 256
		1	TXECNT Above Bus Off Limit
2 (R/NW)	EP	CAN Error Passive Mode. The CAN_STAT.EP bit indicates whether the CAN is in error passive mode.	
		0	TXECNT and RXECNT Below 128
		1	TXECNT or RXECNT Above EP Level
1 (R/NW)	WR	CAN Receive Warning Flag. The CAN_STAT.WR bit indicates whether the CAN has detected a receive warning flag condition.	
		0	RXECNT Below Limit
		1	RXECNT at Limit
0 (R/NW)	WT	CAN Transmit Warning Flag. The CAN_STAT.WT bit indicates whether the CAN detected a transmit warning flag condition.	
		0	TXECNT Below Limit
		1	TXECNT at Limit

Error Counter Register

The CAN_CEC register , CAN_ESR register, and CAN_EWR register control CAN warnings and errors. For detailed information about error and warning operations, see the Event Control section.

The CAN_CEC register holds an error counter for transmit (CAN_CEC.TXECNT) and an error counter for receive (CAN_CEC.RXECNT). After initialization, both counters are 0. Each time a bus error occurs, one of the counters is incremented by either 1 or 8, depending on the error situation (documented in Version 2. 0 of CAN Specification). Successful transmit and receive operations decrement the respective counter by 1.

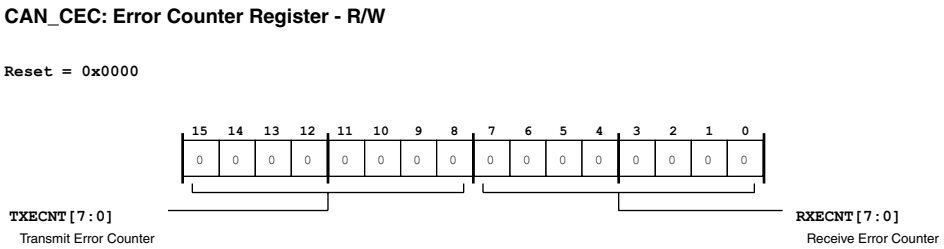


Figure 21-43: CAN_CEC Register Diagram

Table 21-38: CAN_CEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	TXECNT	Transmit Error Counter. The CAN_CEC.TXECNT bits hold the transmit error counter, which is incremented for errors (by either 1 or 8) and is decremented (by 1) for successful transmit operations.
7:0 (R/W)	RXECNT	Receive Error Counter. The CAN_CEC.RXECNT bits hold the receive error counter, which is incremented for errors (by either 1 or 8) and is decremented (by 1) for successful receive operations.

Global CAN Interrupt Status Register

The CAN_GIS register, CAN_GIF register, and CAN_GIM register control CAN interrupts. For detailed information about interrupt operations, see the Event Control section.

The CAN_GIS register holds the interrupt status. All bits in this register are W1C.

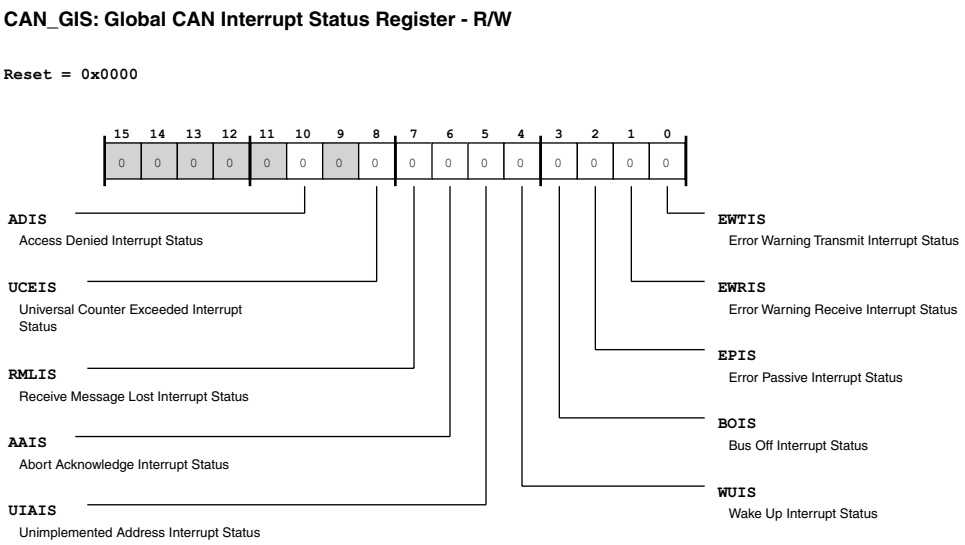


Figure 21-44: CAN_GIS Register Diagram

Table 21-39: CAN_GIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W1C)	ADIS	Access Denied Interrupt Status. The CAN_GIS.ADIS bit indicates when at least one access to the mailbox RAM occurred during a data update by internal logic.	
		0	No Interrupt Pending
		1	Interrupt Pending
8 (R/W1C)	UCEIS	Universal Counter Exceeded Interrupt Status. The CAN_GIS.UCEIS bit indicates when there has been an overflow of the universal counter (in time stamp mode or event counter mode) or the counter has reached the value 0x0000 (in watchdog mode).	
		0	No Interrupt Pending
		1	Interrupt Pending

Table 21-39: CAN_GIS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	RMLIS	Receive Message Lost Interrupt Status. The CAN_GIS.RMLIS bit indicates when a message is received for a mailbox that currently contains unread data. At least one bit in the receive message lost register (CAN_RML1 or CAN_RML2) is set. If the bit in CAN_GIS (and CAN_GIF) is reset and there is at least one bit in CAN_RML1 or CAN_RML2 still set, the bit in CAN_GIF (and CAN_GIF) is not set again. The internal interrupt source signal is only active if a new bit in CAN_RML1 or CAN_RML2 is set.	
		0	No Interrupt Pending
		1	Interrupt Pending
6 (R/W1C)	AAIS	Abort Acknowledge Interrupt Status. The CAN_GIS.AAIS bit indicates when At least one abort acknowledge bit is set in the CAN_AA1 or the CAN_AA2 registers. If the bit in CAN_GIS (and CAN_GIF) is reset and there is at least one bit in CAN_AA1 or CAN_AA2 still set, the bit in CAN_GIS (and CAN_GIF) is not set again. The internal interrupt source signal is only active if a new bit in CAN_AA1 or CAN_AA2 is set. The abort acknowledge bits maintain state even after the corresponding mailbox n is disabled.	
		0	No Interrupt Pending
		1	Interrupt Pending
5 (R/W1C)	UIAIS	Unimplemented Address Interrupt Status. The CAN_GIS.UIAIS bit indicates when there was a processor core access to an address that is not implemented in the CAN.	
		0	No Interrupt Pending
		1	Interrupt Pending
4 (R/W1C)	WUIS	Wake Up Interrupt Status. The CAN_GIS.WUIS bit indicates when the CAN has left the sleep mode because of detected activity on the CAN bus line.	
		0	No Interrupt Pending
		1	Interrupt Pending

Table 21-39: CAN_GIS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W1C)	BOIS	Bus Off Interrupt Status. The CAN_GIS.BOIS bit indicates when the CAN has entered the bus-off state. This interrupt source is active if the status of the CAN changes from normal operation mode to the bus-off mode. If the bit in CAN_GIS (and CAN_GIF) is reset and the bus-off mode is still active, this bit is not set again. If the module leaves the bus-off mode, the bit inCAN_GIS (and CAN_GIF) remains set.	
		0	No Interrupt Pending
		1	Interrupt Pending
2 (R/W1C)	EPIS	Error Passive Interrupt Status. The CAN_GIS.EPIS bit indicates when the CAN has entered the error passive state. This interrupt source is active if the status of the CAN changes from the error active mode to the error passive mode. If the bit in CAN_GIS (and CAN_GIF) is reset and the error passive mode is still active, this bit is not set again. If the CAN leaves the error passive mode, the bit in CAN_GIS (and CAN_GIF) remains set.	
		0	No Interrupt Pending
		1	Interrupt Pending
1 (R/W1C)	EWRIS	Error Warning Receive Interrupt Status. The CAN_GIS.EWRIS bit indicates when the CAN_CEC.RXECNT has reached the warning limit. If the bit in CAN_GIS (and CAN_GIF) is reset and the error warning mode is still active, this bit is not set again. If the CAN leaves the error warning mode, the bit in CAN_GIS (and CAN_GIF) remains set.	
		0	No Interrupt Pending
		1	Interrupt Pending
0 (R/W1C)	EWTIS	Error Warning Transmit Interrupt Status. The CAN_GIS.EWTIS bit indicates when the CAN_CEC.TXECNT has reached the warning limit. If the bit in CAN_GIS (and CAN_GIF) is reset and the error warning mode is still active, this bit is not set again. If the CAN leaves the error warning mode, the bit in CAN_GIS (and CAN_GIF) remains set.	
		0	No Interrupt Pending
		1	Interrupt Pending

Global CAN Interrupt Mask Register

The CAN_GIM register, CAN_GIF register, and CAN_GIF register control CAN interrupts. For detailed information about interrupt operations, see the Event Control section.

The CAN_GIM register holds the interrupt mask. The interrupt mask bits only affect the content of the CAN_GIF register. If the mask bit is not set (enabled/unmasked), the corresponding flag bit is not set when the event occurs.

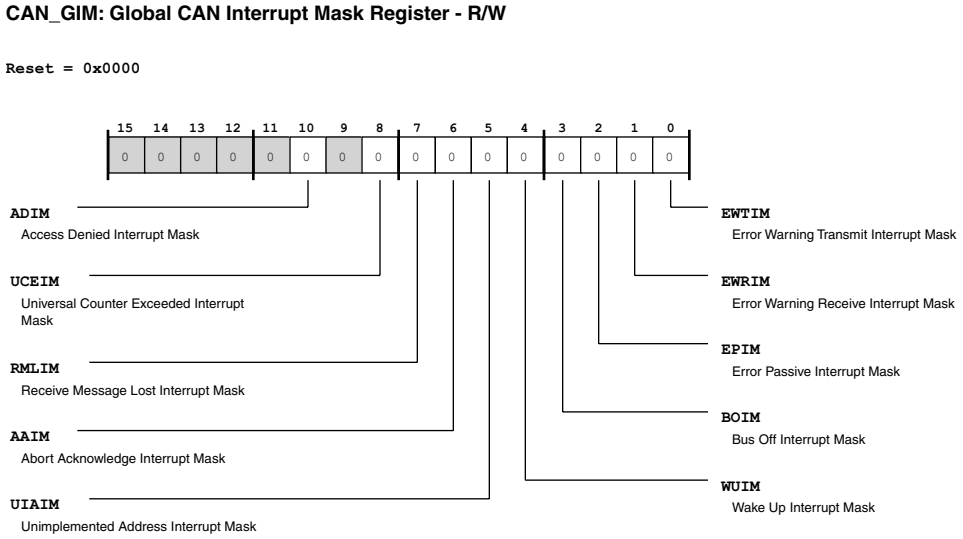


Figure 21-45: CAN_GIM Register Diagram

Table 21-40: CAN_GIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	ADIM	Access Denied Interrupt Mask. The CAN_GIM.ADIM bit enables (unmasks) the access denied interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
8 (R/W)	UCEIM	Universal Counter Exceeded Interrupt Mask. The CAN_GIM.UCEIM bit enables (unmasks) the universal counter exceeded interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)

Table 21-40: CAN_GIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	RMLIM	Receive Message Lost Interrupt Mask. The CAN_GIM.RMLIM bit enables (unmasks) the receive message lost interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
6 (R/W)	AAIM	Abort Acknowledge Interrupt Mask. The CAN_GIM.AAIM bit enables (unmasks) the abort acknowledge interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
5 (R/W)	UIAIM	Unimplemented Address Interrupt Mask. The CAN_GIM.UIAIM bit enables (unmasks) the unimplemented address interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
4 (R/W)	WUIM	Wake Up Interrupt Mask. The CAN_GIM.WUIM bit enables (unmasks) the wake up interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
3 (R/W)	BOIM	Bus Off Interrupt Mask. The CAN_GIM.BOIM bit enables (unmasks) the bus off interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
2 (R/W)	EPIM	Error Passive Interrupt Mask. The CAN_GIM.EPIM bit enables (unmasks) the error passive mode interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
1 (R/W)	EWRIM	Error Warning Receive Interrupt Mask. The CAN_GIM.EWRIM bit enables (unmasks) the error warning receive interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)

Table 21-40: CAN_GIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EWTIM	Error Warning Transmit Interrupt Mask. The CAN_GIM.EWTIM bit enables (unmasks) the error warning transmit interrupt.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)

Global CAN Interrupt Flag Register

The CAN_GIF register, CAN_GIF register, and CAN_GIM register control CAN interrupts. For detailed information about interrupt operations, see the Event Control section.

The CAN_GIF register holds the interrupt flag. The CAN_INT.GIRQ bit is only asserted if a bit in the CAN_GIF is set. The CAN_INT.GIRQ bit remains set as long as at least one bit in the CAN_GIF register is set.All bits in this register are W1C.

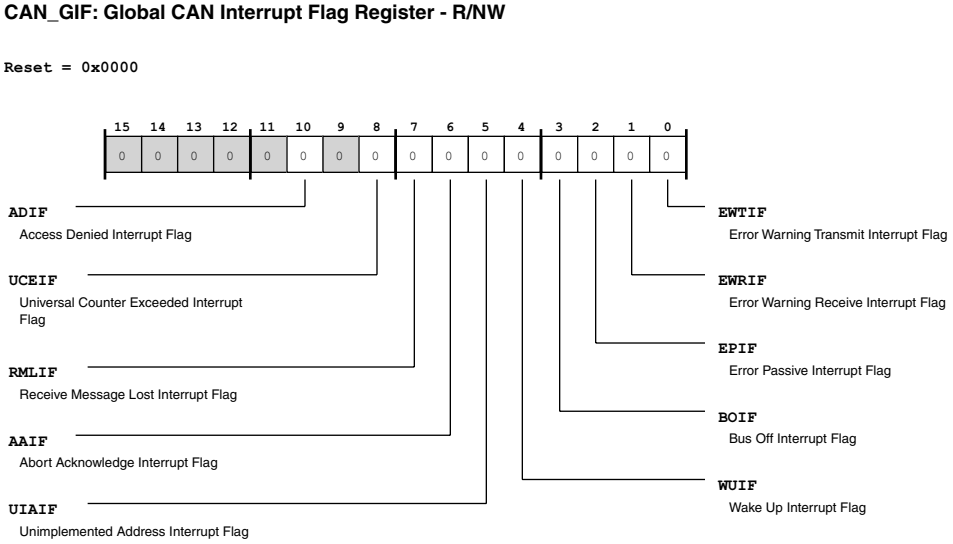


Figure 21-46: CAN_GIF Register Diagram

Table 21-41: CAN_GIF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/NW)	ADIF	Access Denied Interrupt Flag. The CAN_GIF.ADIF bit indicates the access denied interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
8 (R/NW)	UCEIF	Universal Counter Exceeded Interrupt Flag. The CAN_GIF.UCEIF bit indicates the universal counter exceeded interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
7 (R/NW)	RMLIF	Receive Message Lost Interrupt Flag. The CAN_GIF.RMLIF bit indicates the receive message lost interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
6 (R/NW)	AAIF	Abort Acknowledge Interrupt Flag. The CAN_GIF.AAIF bit indicates the abort acknowledge interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
5 (R/NW)	UIAIF	Unimplemented Address Interrupt Flag. The CAN_GIF.UIAIF bit indicates the unimplemented address interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
4 (R/NW)	WUIF	Wake Up Interrupt Flag. The CAN_GIF.WUIF bit indicates the wake up interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)

Table 21-41: CAN_GIF Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/NW)	BOIF	Bus Off Interrupt Flag. The CAN_GIF.BOIF bit indicates the bus off interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
2 (R/NW)	EPIF	Error Passive Interrupt Flag. The CAN_GIF.EPIF bit indicates the error passive mode interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
1 (R/NW)	EWRIF	Error Warning Receive Interrupt Flag. The CAN_GIF.EWRIF bit indicates the error warning receive interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
0 (R/NW)	EWTIF	Error Warning Transmit Interrupt Flag. The CAN_GIF.EWTIF bit indicates the error warning transmit interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)

CAN Master Control Register

The CAN_CTL register controls CAN mode requests, including soft reset.

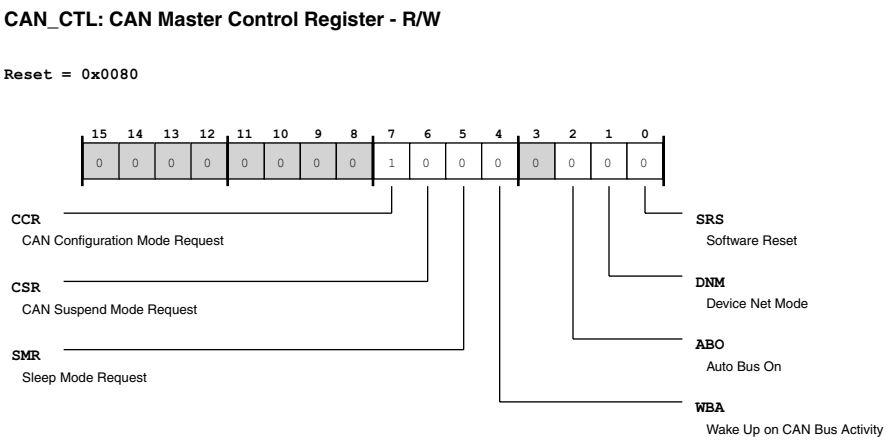


Figure 21-47: CAN_CTL Register Diagram

Table 21-42: CAN_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	CCR	CAN Configuration Mode Request. The CAN_CTL.CCR bit requests that the CAN enter configuration mode. Note that the CAN should always be put in configuration mode before modifying the CAN_CLK or CAN_TIMING registers.	
		0	No Request (Exit Configuration Mode)
		1	Request Configuration Mode
6 (R/W)	CSR	CAN Suspend Mode Request. The CAN_CTL.CSR bit requests that the CAN enter suspend mode. The CAN enters suspend mode after the current operation of the CAN bus is finished.	
		0	No Request (Exit Suspend Mode)
		1	Request Suspend Mode
5 (R/W)	SMR	Sleep Mode Request. The CAN_CTL.SMR bit requests that the CAN enter sleep mode. The CAN enters sleep mode after the current operation of the CAN bus is finished.	
		0	No Request (Exit Sleep Mode)
		1	Request Sleep Mode

Table 21-42: CAN_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	WBA	Wake Up on CAN Bus Activity. The CAN_CTL.WBA bit enables wake on CAN bus activity. When enabled, a dominant bit on the CAN_RX pin ends sleep mode (in addition the default wake up condition of a write to the CAN_INT register).	
		0	Disable Wake on Bus Activity
		1	Enable Wake on Bus Activity
2 (R/W)	ABO	Auto Bus On. The CAN_CTL.ABO bit selects whether (if enabled) the CAN enters active mode after the BusOff recovery sequence or (if disabled) the CAN enters configuration mode after the BusOff recovery sequence.	
		0	Disable Auto Bus On
		1	Enable Auto Bus On
1 (R/W)	DNM	Device Net Mode. The CAN_CTL.DNM bit enables mailbox filtering on a data field. The filtering is done on the standard ID of the message and data fields. For more information, see the CAN_AMnnH.FDF bit description.	
		0	Disable Device Net Mode
		1	Enable Device Net Mode
0 (R/W)	SRS	Software Reset. The CAN_CTL.SRS bit resets the CAN, bringing all control registers to a defined state. Soft reset is entered immediately after software has set the CAN_CTL.SRS bit.	
		0	No Action
		1	Reset CAN

Interrupt Pending Register

The CAN_INT register indicates the status of pending CAN interrupts and indicates the state of the CAN_RX and CAN_TX pins. Though this register is read-only, a write is allowed to exit the built-in sleep mode of the module on processors supporting this feature.

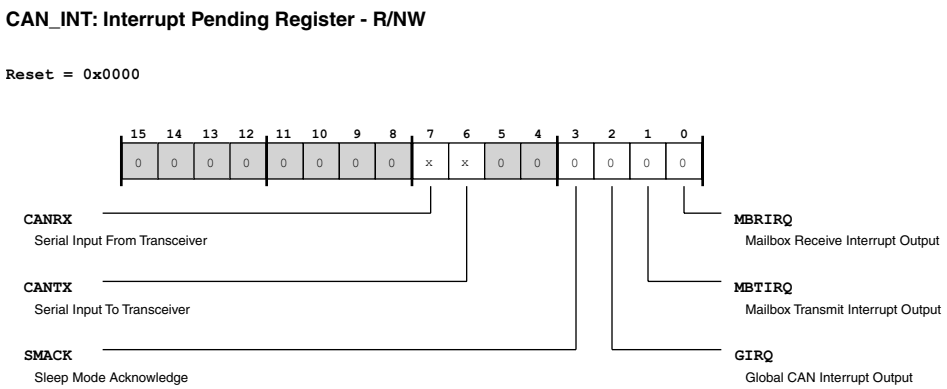


Figure 21-48: CAN_INT Register Diagram

Table 21-43: CAN_INT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	CANRX	Serial Input From Transceiver. The CAN_INT.CANRX bit indicates the logic value that the CAN detects on the CAN_RX pin. Note that the reset/default value for CAN_INT.CANRX is dependent on pin values.	
		0	Dominant Value (Low Active)
		1	Recessive Value (High Active)
6 (R/NW)	CANTX	Serial Input To Transceiver. The CAN_INT.CANTX bit indicates the logic value that the CAN detects on the CAN_TX pin. Note that the reset/default value for CAN_INT.CANTX is dependent on pin values.	
		0	Dominant Value (Low Active)
		1	Recessive Value (High Active)
3 (R/W)	SMACK	Sleep Mode Acknowledge. The CAN_INT.SMACK bit indicates when the CAN has entered sleep mode.	
		0	Not in Sleep Mode
		1	Sleep Mode
2 (R/W)	GIRQ	Global CAN Interrupt Output. The CAN_INT.GIRQ bit indicates when at least one bit is set in the CAN_GIF register, indicating at least one unmasked CAN is flagged (latched). The CAN_INT.GIRQ bit remains set as long as at least one bit is set in the CAN_GIF register.	
		0	No CAN Global Interrupt Flag Set
		1	CAN Global Interrupt Flag (1 or More) Set

Table 21-43: CAN_INT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	MBTIRQ	Mailbox Transmit Interrupt Output. The CAN_INT.MBTIRQ bit indicates when any bits are set in the CAN_MBTIF1 register or CAN_MBTIF2 register, indicating transmit.	
		0	No CAN Transmit Flags Set
		1	CAN Transmit Flags Set (1 or More)
0 (R/W)	MBRIRQ	Mailbox Receive Interrupt Output. The CAN_INT.MBRIRQ bit indicates when any bits are set in the CAN_MBRIF1 register or CAN_MBRIF2 register, indicating receive.	
		0	No CAN Receive Flags Set
		1	CAN Receive Flags Set (1 or More)

Temporary Mailbox Disable Register

The CAN_MBTD register supports temporarily and selectively disabling CAN mailboxes. For more information about this feature, see the Operating Modes section.

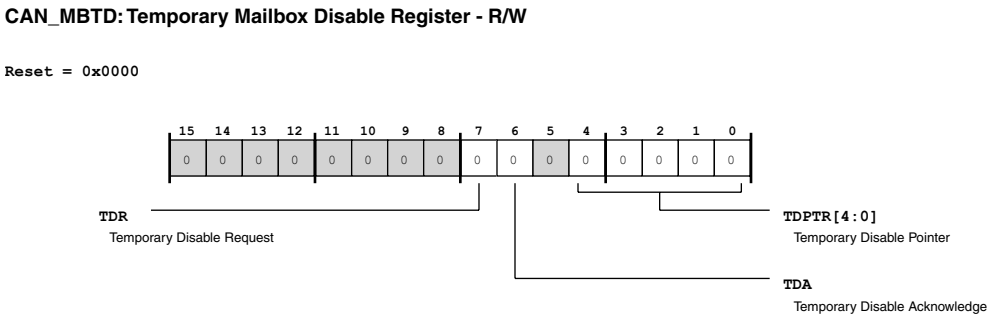


Figure 21-49: CAN_MBTD Register Diagram

Table 21-44: CAN_MBTD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	TDR	Temporary Disable Request. The CAN_MBTD.TDR bit hold the pointer to mailbox, which is disabled when the CAN_MBTD.TDR bit is set.	
		0	No Request
		1	Request Temporary Mailbox Disable

Table 21-44: CAN_MBTD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/NW)	TDA	Temporary Disable Acknowledge. The CAN_MBTD.TDA bit indicates when the mailbox (to which the CAN_MBTD.TDPTR bit point) is disabled. When this bit is set for a mailbox, only the data field of that mailbox may be updated. Accesses that mailboxs control bits and the identifier are denied.	
		0	No Acknowledge
		1	Acknowledge Temporary Mailbox Disable
4:0 (R/W)	TDPTR	Temporary Disable Pointer. The CAN_MBTD.TDPTR bits hold the pointer to mailbox, which is disabled when the CAN_MBTD.TDR bit is set.	

Error Counter Warning Level Register

The CAN_EWR register, CAN_CEC register, and CAN_ESR register control CAN warnings and errors. For detailed information about error and warning operations, see the Operating Modes section.

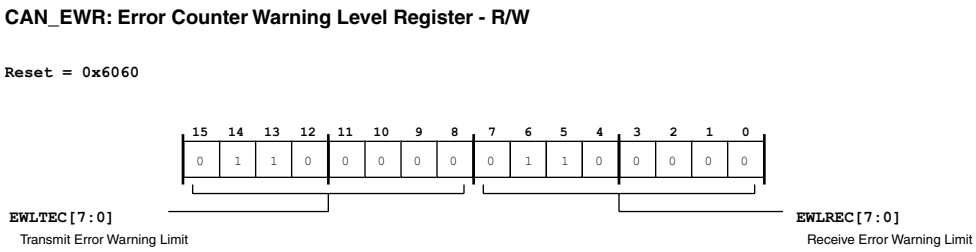


Figure 21-50: CAN_EWR Register Diagram

Table 21-45: CAN_EWR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	EWLTEC	Transmit Error Warning Limit. The CAN_EWR.EWLTEC bits select the transmit error warning limit, which is used as a condition for the CAN_GIS.EWTIS interrupt.
7:0 (R/W)	EWLREC	Receive Error Warning Limit. The CAN_EWR.EWLREC bits select the receive error warning limit, which is used as a condition for the CAN_GIS.EWRIS interrupt.

Error Status Register

The CAN_ESR register, CAN_CEC register, and CAN_EWR register control CAN warnings and errors. All bits in the CAN_ESR are W1C. Note that the CAN updates the CAN_CEC register when error status is detected in the CAN_ESR register. For detailed information about error and warning operations, see the Operating Modes section.

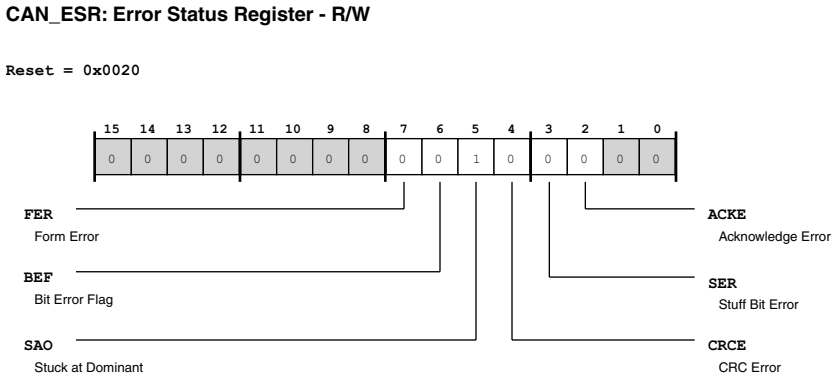


Figure 21-51: CAN_ESR Register Diagram

Table 21-46: CAN_ESR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	FER	Form Error. The CAN_ESR.FER bit indicates when a form error occurs, indicating that a fixed-form bit position in the CAN frame contains one or more illegal bits. This occurs when a dominant bit is detected at a delimiter or end-of-frame bit position.	
		0	No Status
		1	Form Error
6 (R/W1C)	BEF	Bit Error Flag. The CAN_ESR.BEF bit bit indicates (detected by the transmitting node only) the value on the CAN_RX pin does not equal what is being transmitted on the CAN_TX pin. When a node is transmitting, it continuously monitors its receive pin (CAN_RX) and compares the received data with the transmitted data. The node postpones the transmission (during the arbitration phase) if the received and transmitted data do not match. After the arbitration phase (CAN_MBnn_ID1.RTR bit sent successfully), a bit error is signaled when the value on the CAN_RX pin does not equal what is being transmitted on the CAN_TX pin.	
		0	No Status
		1	Bit Error Flag

Table 21-46: CAN_ESR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1C)	SAO	Stuck at Dominant. The CAN_ESR.SAO bit indicates when the CAN_RX pin sticks at dominant level, indicating that shorted wires are likely.	
		0	No Status
		1	Stuck At Dominant
4 (R/W1C)	CRCE	CRC Error. The CAN_ESR.CRCE bit indicates when a CRC error occurs. This error may occur when a receiver calculates the CRC on the data it received and finds the value different than the CRC that was transmitted on the bus.	
		0	No Status
		1	CRC Error
3 (R/W1C)	SER	Stuff Bit Error. The CAN_ESR.SER bit indicates when a stuff bit error (stuffed 6th consecutive bit value is the same as the previous five bits) occurs. The CAN specification requires that the transmitter insert an extra stuff bit of opposite value after 5 bits have been transmitted with the same value. The receiver disregards the value of these stuff bits. The receiver takes advantage of the signal edge to re-synchronize itself. A stuff bit error occurs on receiving nodes when the 6th consecutive bit value is the same as the previous five bits.	
		0	No Status
		1	Stuff Bit Error Receive
2 (R/W1C)	ACKE	Acknowledge Error. The CAN_ESR.ACKE bit indicates when an acknowledge error occurs, indicating that a message is sent and no receivers drive an acknowledge bit.	
		0	No Status
		1	Acknowledge Error

Universal Counter Register

The CAN_UCCNT register holds the current universal count. This register is re-loaded from the CAN_UCRC register when the decrements to zero in auto-transmit mode.

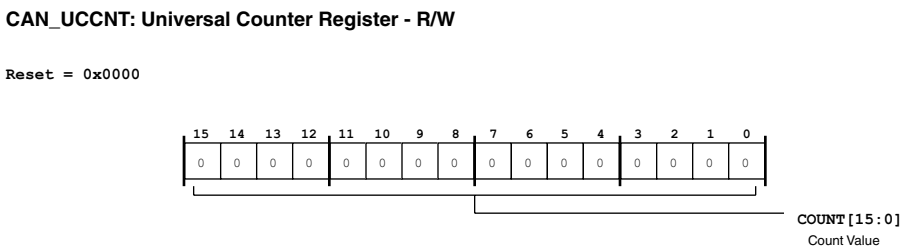


Figure 21-52: CAN_UCCNT Register Diagram

Table 21-47: CAN_UCCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	COUNT	Count Value. The CAN_UCCNT.COUNT bits hold the current universal count value.

Universal Counter Reload/Capture Register

The CAN_UCRC register holds the period value (universal count), which is used in auto-transmit mode as the period for sending the message in mailbox 11 (broadcast heartbeat) to all CAN nodes. Accordingly, messages sent this way usually have high priority.

The period value is written to the CAN_UCRC register. When auto-transmit mode is enabled (CAN_UCCNF.UCCNF = 0x3), the CAN loads the counter with the value in CAN_UCRC. The counter decrements to 0 at the CAN bit clock rate, then is reloaded. Each time the counter decrements to 0, the CAN sets the CAN_TRS1.MB bit for mailbox 11 and sends the corresponding message from mailbox 11.

Note that for auto-transmit mode, mailbox 11 must be configured as a transmit mailbox and must contain valid data (identifier, control bits, and data). This setup must occur before the counter first expires after this mode is enabled.

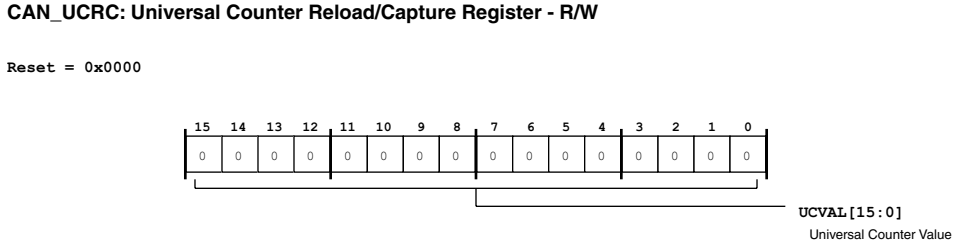


Figure 21-53: CAN_UCRC Register Diagram

Table 21-48: CAN_UCRC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	UCVAL	Universal Counter Value. The CAN_UCRC.UCVAL bits hold the value for the universal count period, which is used in auto-transmit mode.

Universal Counter Configuration Mode Register

The CAN_UCCNF register controls the operation of the universal counter, including counter enable and counter mode selection.

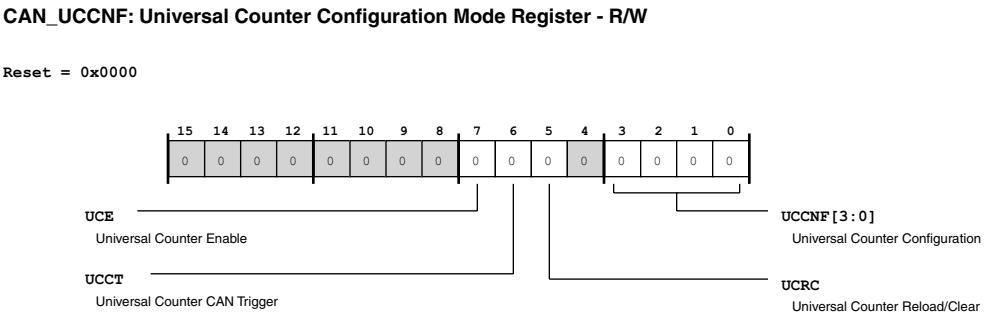


Figure 21-54: CAN_UCCNF Register Diagram

Table 21-49: CAN_UCCNF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	UCE	Universal Counter Enable. The CAN_UCCNF.UCE bit enables universal counter operation in the mode selected by the CAN_UCCNF.UCCNF bits.	
		0	Disable Counter
		1	Enable Counter
6 (R/W)	UCCT	Universal Counter CAN Trigger. The CAN_UCCNF.UCCT bit enables the universal counter trigger, directing the CAN to re-load the counter on mailbox 4 reception in watchdog mode and clear the counter on mailbox 4 reception in time stamp mode. This bit has no effect in all other modes.	
		0	Disable Trigger
		1	Enable Trigger

Table 21-49: CAN_UCCNF Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	UCRC	Universal Counter Reload/Clear. The CAN_UCCNF.UCRC bit re-loads or clears the universal counter, depending on the counter mode. In watchdog mode, setting this bit directs the CAN to re-load the counter. In all other modes, setting this bit directs the CAN to clear the counter.	
		0	No Action
		1	Re-load or Clear the Counter
3:0 (R/W)	UCCNF	Universal Counter Configuration. The CAN_UCCNF.UCCNF bits select the universal counter operating mode. For more information about these modes, see the Operating Modes section.	
		0	Reserved
		1	Time Stamp Mode
		2	Watchdog Mode
		3	Auto-transmit Mode
		4	Reserved
		5	Reserved
		6	Count Error Frames
		7	Count Overload Frames
		8	Count Arbitration Lost
		9	Count Aborted Transmissions
		10	Count Successful Transmissions
		11	Count Rejected Receive Messages
		12	Count Receive Message Lost
		13	Count Successful Receptions
		14	Count Stored Receptions
		15	Count Valid Messages

Acceptance Mask (L) Register

The CAN_AMnnL register and CAN_AMnnH register manage acceptance mask operation. For information about acceptance mask operation, see the Receive Operation section.

CAN_AMnnL: Acceptance Mask (L) Register - R/W

Reset = 0x0000

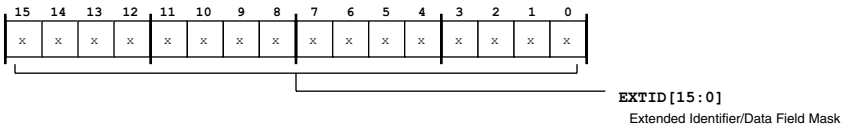


Figure 21-55: CAN_AMnnL Register Diagram

Table 21-50: CAN_AMnnL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	EXTID	Extended Identifier/Data Field Mask. The CAN_AMnnL.EXTID bits hold the extended ID (lower 16 bits) for data field mask in acceptance mask operations.

Acceptance Mask (H) Register

The CAN_AMnnH register and CAN_AMnnL register manage acceptance mask operation. For information about acceptance mask operation, see the Receive Operation section.

CAN_AMnnH: Acceptance Mask (H) Register - R/W

Reset = 0x0000

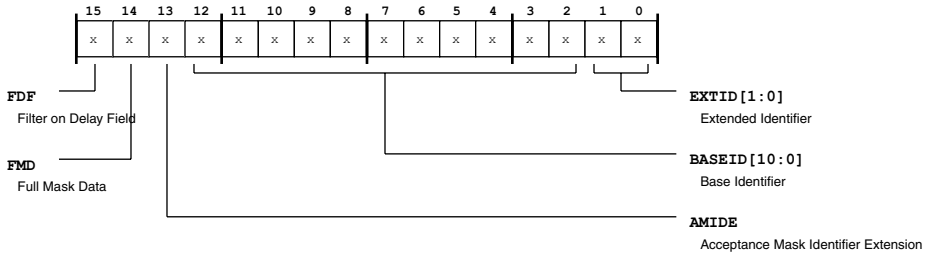


Figure 21-56: CAN_AMnnH Register Diagram

Table 21-51: CAN_AMnnH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	FDF	Filter on Delay Field. The CAN_AMnnH.FDF bit selects the operation of the CAN_AMnnH register and CAN_AMnnL register when the CAN_CTL.DNM bit is enabled. If the CAN_AMnnH.FDF bit is set, the corresponding CAN_AMnnL.EXTID bits hold the data field mask. If the CAN_AMnnH.FDF bit is cleared, the corresponding CAN_AMnnL.EXTID bits hold the high bits of the extended identifier mask.
14 (R/W)	FMD	Full Mask Data. The CAN_AMnnH.FMD bit works with the CAN_AMnnH.FDF bit to determine data field filtering. For information about data field filtering, see the Receive Operation section.
13 (R/W)	AMIDE	Acceptance Mask Identifier Extension. The CAN_AMnnH.AMIDE bit enables the comparison of the received message ID to the value in the CAN_AMnnH.EXTID and CAN_AMnnL.EXTID bits.
12:2 (R/W)	BASEID	Base Identifier. The CAN_AMnnH.BASEID bits hold the base ID for acceptance mask operations.
1:0 (R/W)	EXTID	Extended Identifier. The CAN_AMnnH.EXTID bits hold the extended ID (upper two bits) for acceptance mask operations.

Mailbox Word 0 Register

The CAN_MBnn_DATA0 register holds mailbox data bytes.

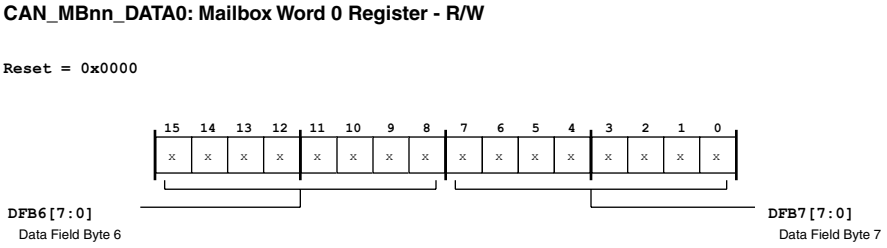


Figure 21-57: CAN_MBnn_DATA0 Register Diagram

Table 21-52: CAN_MBnn_DATA0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	DFB6	Data Field Byte 6. The CAN_MBnn_DATA0.DFB6 bits hold mailbox data.
7:0 (R/W)	DFB7	Data Field Byte 7. The CAN_MBnn_DATA0.DFB7 bits hold mailbox data.

Mailbox Word 1 Register

The CAN_MBnn_DATA1 register holds mailbox data bytes.

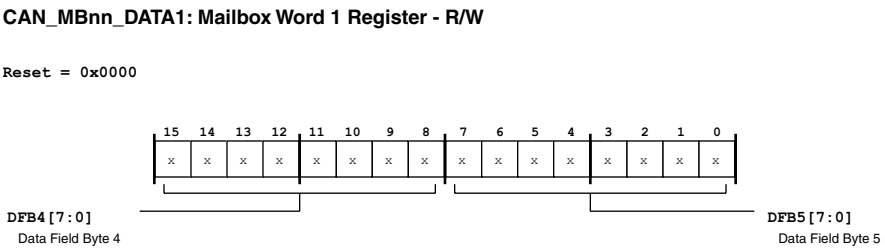


Figure 21-58: CAN_MBnn_DATA1 Register Diagram

Table 21-53: CAN_MBnn_DATA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	DFB4	Data Field Byte 4. The CAN_MBnn_DATA1.DFB4 bits hold mailbox data.
7:0 (R/W)	DFB5	Data Field Byte 5. The CAN_MBnn_DATA1.DFB5 bits hold mailbox data.

Mailbox Word 2 Register

The CAN_MBnn_DATA2 register holds mailbox data bytes.

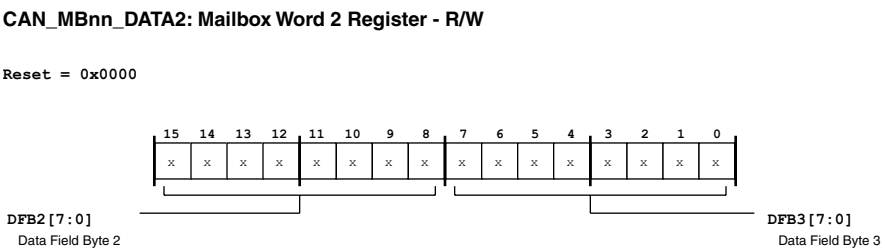


Figure 21-59: CAN_MBnn_DATA2 Register Diagram

Table 21-54: CAN_MBnn_DATA2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	DFB2	Data Field Byte 2. The CAN_MBnn_DATA2.DFB2 bits hold mailbox data.
7:0 (R/W)	DFB3	Data Field Byte 3. The CAN_MBnn_DATA2.DFB3 bits hold mailbox data.

Mailbox Word 3 Register

The CAN_MBnn_DATA3 register holds mailbox data bytes.

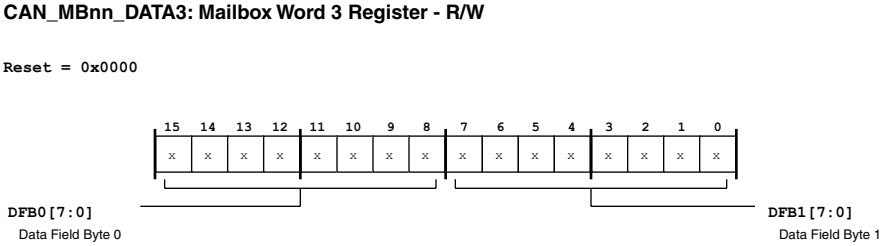


Figure 21-60: CAN_MBnn_DATA3 Register Diagram

Table 21-55: CAN_MBnn_DATA3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	DFB0	Data Field Byte 0. The CAN_MBnn_DATA3.DFB0 bits hold mailbox data.
7:0 (R/W)	DFB1	Data Field Byte 1. The CAN_MBnn_DATA3.DFB1 bits hold mailbox data.

Mailbox Length Register

The CAN_MBnn_LENGTH register holds the data length code for the received remote frame. For more information about remote frames, see the Remote Frame Handling section.

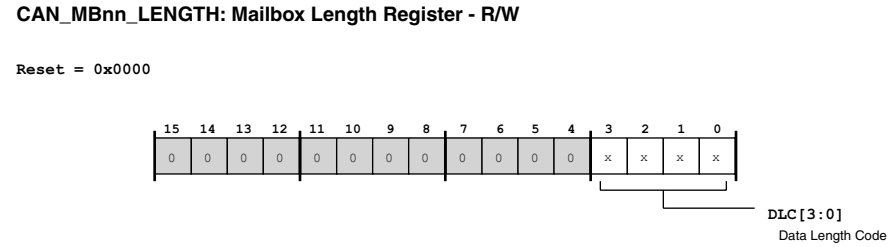


Figure 21-61: CAN_MBnn_LENGTH Register Diagram

Table 21-56: CAN_MBnn_LENGTH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	DLC	Data Length Code. The CAN_MBnn_LENGTH.DLC bits hold the DLC value of the received remote frame. The received value overwrites any previous value.

Mailbox Timestamp Register

The CAN_MBnn_TIMESTAMP register holds an indication of the time of reception or transmission for each message, when the universal counter is in time stamp mode (CAN_UCCNF.UCCNF =0x1). In this mode, the CAN writes the value of the counter (CAN_UCCNT) to the CAN_MBnn_TIMESTAMP register when a received message is stored or a message is transmitted. For more information about timestamps, see the Time Stamps section.

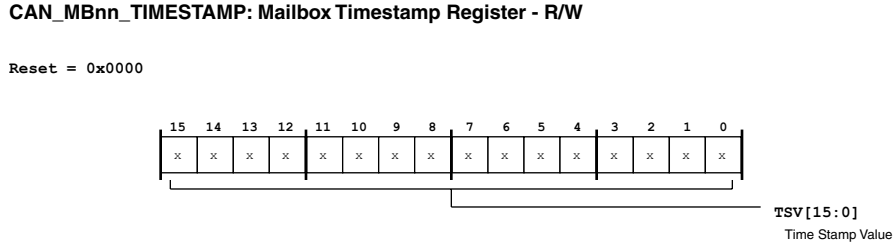


Figure 21-62: CAN_MBnn_TIMESTAMP Register Diagram

Table 21-57: CAN_MBnn_TIMESTAMP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	TSV	Time Stamp Value. The CAN_MBnn_TIMESTAMP.TSV bits hold the message timestamp value.

Mailbox ID 0 Register

The CAN_MBnn_ID0 register contains the lower 16 bits of the 18-bit extended identifier.

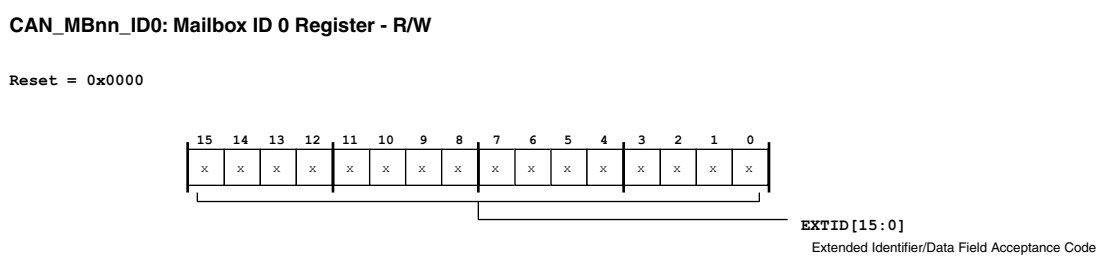


Figure 21-63: CAN_MBnn_ID0 Register Diagram

Table 21-58: CAN_MBnn_ID0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	EXTID	Extended Identifier/Data Field Acceptance Code. The CAN_MBnn_ID0.EXTID bits hold the lower 16 bits of the 18-bit extended ID.

Mailbox ID 1 Register

The CAN_MBnn_ID1 register contains the identifier bits of mailbox. The 11-bit BASE_ID is mapped to The CAN_MBnn_ID1.BASEID field. It also enables the extended identification and contains upper two bits of 18-bit extended identifier. This register also enables the acceptance mask operations. For information about acceptance mask operation, see the Receive Operation section.

CAN_MBnn_ID1: Mailbox ID 1 Register - R/W

Reset = 0x0000

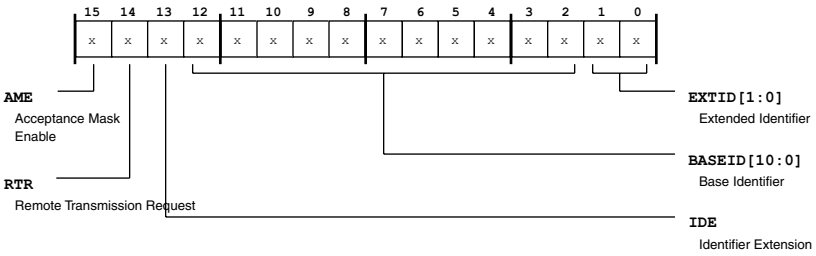


Figure 21-64: CAN_MBnn_ID1 Register Diagram

Table 21-59: CAN_MBnn_ID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	AME	Acceptance Mask Enable. The CAN_MBnn_ID1.AME bit enables acceptance mask operations if the mailbox is configured as receiver. When enabled (=1), only those bits that have the corresponding mask bit cleared are compared to the received message ID. A bit position that is set in the mask register does not need to match. This bit should be set to 0 when the mailbox is configured in transmit mode.
14 (R/W)	RTR	Remote Transmission Request. The CAN_MBnn_ID1.RTR bit selects whether the frame contains data (data frame) or contains a request for data associated with the message identifier in the frame being sent (remote frame).
13 (R/W)	IDE	Identifier Extension. The CAN_MBnn_ID1.IDE bit enables the comparison of the received message ID to the value in the CAN_MBnn_ID1.EXTID and CAN_MBnn_ID0.EXTID bits. When configured as transmitter, it sends the extended identifier in addition to the base identifier.
12:2 (R/W)	BASEID	Base Identifier. The CAN_MBnn_ID1.BASEID bits hold the base identifier for acceptance mask operations.
1:0 (R/W)	EXTID	Extended Identifier. The CAN_MBnn_ID1.EXTID bits hold the upper two bits of 18-bit extended identifier.

22 Universal Serial Bus (USB)

The USB OTG controller provides a low-cost connectivity solution for consumer mobile devices such as cell phones, digital still cameras and MP3 players, allowing these devices to transfer data using a point-to-point USB connection without the need for a personal computer host.

The USB controller can operate in a traditional USB peripheral-only mode as well as the host mode presented in the On-The-Go (OTG) supplement[1] to the USB 2.0 Specification[2]The USB module supports:

- Host mode transfers at high-speed (480 Mbp/sec) rate
- Host mode transfers at full-speed (12 Mbp/sec) rate
- Host mode transfers at low-speed (1.5 Mbp/sec) rates
- Peripheral mode transfers at high-speed (480 Mbp/sec) rate
- Peripheral mode transfers at full-speed (12 Mbp/sec) rate

The USB controller uses a peripheral bus slave interface to access its control and status registers as well as read and write to the endpoint packet buffers. Data is transferred to and from the USB controller through any of the 11 transmit and 11 receive endpoint FIFOs (EP1 – EP11), providing a total of 22 data endpoints.

USB Features

The USB controller provides the following features:

- Low speed, full speed, high speed rates supported
- One bidirectional control endpoint
- Eleven transmit and eleven receive unidirectional endpoints
- 16 KB dynamically configured FIFO RAM
- Eight DMA master channels
- Two top-level maskable general purpose interrupts
- Low power wakeup on activity
- VBUS control interrupts for external analog VBUS control

1.On-The-Go Supplement to the USB 2.0 Specification, Rev 1.0a; June 24, 2003; USB-IF
2.Universal Serial Bus Specification 2.0.

- Session request protocol (SRP) and host negotiation protocol (HNP) capability
- Host transaction scheduling in hardware
- Soft connect/disconnect feature
- Full- and high-speed physical layer UTMI+ level 3 interface for on-chip PHY
- Backwards compatible with existing USB 1.1 hosts
- Support for Battery Charging Specification Revision 1.1

The number of active endpoints at one time is only limited by device requirements or system bandwidth, because each endpoint operates independently from the next. Software determines the type of transfer for each endpoint individually and also the manner in which it is transferred between the USB controller and memory (DMA or interrupt-based). Endpoint zero is used solely for receive and transmit control transfers, which are used for device configuration and information gathering.

USB Functional Description

The following sections describe the function of the USB OTG interface.

USB Architectural Concepts

The USB controller operates in either of two USB operation modes (peripheral or host mode) at a given time.

In peripheral mode, the USB controller encodes, decodes, checks, and directs all USB packets sent and received, responding appropriately to host requests. Data is transferred from the processor core memory into the device's TX FIFOs to be transmitted onto USB as IN packets. In the other direction USB OUT packets are received into the RX FIFOs (having been sent from the host) and transferred to system memory for processing or storage. In peripheral mode, the USB controller acts as a slave device to another USB host; either a personal computer or another OTG host controller.

When operating in host mode, the USB controller uses simple hosting capabilities to master point-to-point connections with another USB peripheral, initiating transfers on the bus for the peripheral to respond. USB IN packets are received into the RX FIFOs to be moved into the processor core memory, and data written into TX FIFOs is transmitted onto the bus as USB OUT packets. In this mode, the USB controller encodes, decodes, and checks USB packets sent and received. The controller automatically schedules isochronous and interrupt transfers from the endpoint buffers such that one transaction is performed every *n* frames, where n represents the polling interval programmed for the endpoint.

Any of the endpoints can be programmed to be written to or read from using the DMA master channels to provide the most efficient means of transferring data between the controller and on-chip memory. USB endpoints 0 through 11 have DMA interrupt lines (`USB_DMA_IRQ`) providing a total of eight DMA request lines. Two top-level maskable interrupts are provided, each of which can be sourced from any or all of

transmit endpoint status, receive endpoint status or global USB status. Details of these can be found in Interrupt Signals .

The USB controller's RAM interface supports a single block of synchronous single-port RAM used to buffer the USB packets. 16K bytes of SRAM are available.

The UTMI+ level 3 PHY interface provides a means of connecting a selection of high- or full-speed PHYs to the controller, from device-only PHYs through full OTG compliant PHYs. The details of the PHY interface can be found in UTMI Interface.

ATTENTION: Check the processor data sheet for requirements regarding minimum system clock frequency needed for proper USB operation.

The USB controller is configured as either a USB OTG A device or B device depending on the type of plug inserted into its USB receptacle. This is determined by the state of the `USB_ID` (connector ID) pin.

The asynchronous wakeup circuit is used to detect when another B device is asserting its D+ pull-up to initiate the SRP (session request protocol) when all other clocks are off. This circuit requires a slow clock (for example, 32 kHz). This slow clock is derived from *SCLK* and enabled using the `USB_PHY_CTL.EN` bit.

Use of the controller for OTG functionality requires the capability to drive VBUS (as a default A device powering the bus), to discharge VBUS (speeding up the time for VBUS to fall below the *SessionEnd* threshold as a B device checking initial conditions), and to charge VBUS to 2.1 V (when initiating SRP as a B device). These controls are driven from the UTMI interface, but the controller also provides a separate interrupt register, `USB_VBUS_CTL`, which represents the drive VBUS, discharge VBUS, and charge VBUS signaling. See the register section for more information on these controls.

Multi-Point Support

The USB controller has the facility, when operating in host mode, to act as the host to a range of USB peripheral devices – high-speed, full-speed or low-speed – where these devices are connected to the USB controller via a USB hub. The key feature of the controller's support for multiple devices is its facility to allow the functions of the target devices to be individually allocated to the different Rx and Tx endpoints implemented. Furthermore, this allocation can be made dynamically, allowing the devices from the targeted peripheral list to be used in different combinations. The combinations of peripheral devices that may be used together are limited by the numbers of Tx and Rx endpoints implemented in the controller. Further devices can only be added where the endpoints they require remain available.

On-Chip Bus Interfaces

The USB controller uses two 32-bit wide independent bus interfaces, a master and a slave, to communicate with a processor-based subsystem. The slave interface allows the processor core to access the control and status registers (including DMA master registers) and the endpoint FIFOs. The master interface is used by the integrated DMA to drive data into or out of the endpoint FIFOs with minimal processor core interaction. For more information, see *USB Block Diagram*.

FIFO Configuration

Each bidirectional endpoint (provided as two unidirectional endpoints) has its own endpoint number (0 for control, 1 on up for data transfer). Although two endpoints might use the same number, the endpoints may support different transfer types. Each of these bidirectional endpoints has a fixed region of the SRAM in the USB controller to which it has access, and this feature dictates to some extent the types of transfers that may be used for that particular endpoint. This restriction follows from the maximum size of USB packets, which varies with each transfer type. The following table lists the endpoint FIFO configuration, with an indication of the transfer types possible for that particular buffer size.

Table 22-1: FIFO Sizes and Transfer Types

Bidirectional Endpoint (RX and TX)	FIFO Size (each direction)	USB Transfer Types
0	64 bytes	Size fixed for control transfers
1–4	Dynamically configured in powers of 2 from 8 to 8192 bytes	Bulk, Interrupt, Isochronous
1–11	Dynamically configured in powers of 2 from 8 to 8192 bytes	Bulk, Interrupt, Isochronous

Each endpoint FIFO can buffer one or two packets (in double-buffered mode). Double-buffering is recommended for most applications to improve efficiency by reducing the frequency with which each endpoint needs to be serviced.

Double-buffering bulk transactions means that data transfers over the USB are not slowed if packets can be loaded/unloaded from the FIFO in the time it takes to transfer a packet over the bus. Double-buffering isochronous transactions also allows more time to load/unload the FIFO, but in addition, it also allows the SOF interrupt to be used to service the endpoint rather than the endpoint interrupt. This has the following advantages:

- Easy detection of lost packets
- Regular interrupt timing (making it easier to source/sink the data)
- If more than one isochronous endpoint is used, they can all be serviced with one interrupt.

Clocking

The USB controller uses the system clock *SCLK* to generate an internal clock (CLK) used to clock the USB registers. For proper operation, the system clock, SCLK, must be greater than 30MHz. The transceiver clock (XCLK) is a 60 MHz clock sourced from the UTMI PHY and is used by the PHY interface logic and USB engine.

The USB clock (USB_CLKIN) is provided through a dedicated external crystal or crystal oscillator. Using the integrated USB phase locked loop, USB PLL, with programmable multipliers, the USB on-the-go dual-role device controller generates the necessary internal clocking frequency for USB.

NOTE: For best performance, (for best signal integrity), follow the guidelines in the data sheet for selecting an input clock frequency.

When the controller is in the SUSPEND state and when no session is active, the clock to much of the USB controller is stopped to reduce power consumption. The clock becomes operational again when RESUME signaling is detected on the USB lines.

UTMI Interface

The interface to the on-chip PHY uses the industry-standard UTMI+ (universal transceiver macro interface) level 3.

This provides full- and high-speed device and OTG functionality and supports communication to a hub.

The PHY is a mixed-signal block and includes the following:

- Full-speed and high-speed drivers and receivers (single-ended and differential)
- Full-speed and high-speed CDR
- Full-speed/high-speed shift registers, NRZI encode/decode and bit-stuff encode/decode
- Data line pull-up and pull-down resistors
- VBUS and `USB_ID` level detection
- Host disconnect detection

Although the UTMI specification indicates that VBUS charging, driving and discharging be done inside the PHY, for process-restricting and power reasons, these functions need to be implemented off-chip in a separate USB charge-pump chip.

ADSP-BF60x USB Register List

The universal serial bus (USB) controller is a multipoint high-speed dual role USB 2,0 compliant controller. The USB controller can operate in a traditional USB peripheral-only mode as well as the host mode presented in the on-the-go (OTG) supplement to the USB 2.0 Specification, Rev 1.0a; June 24, 2003; USB-IF. A set of registers govern USB controller operations. For more information on USB controller functionality, see the USB controller register descriptions.

Table 22-2: ADSP-BF60x USB Register List

Name	Description
`USB_FADDR`	Function Address Register
`USB_POWER`	Power and Device Control Register

Table 22-2: ADSP-BF60x USB Register List (Continued)

Name	Description
USB_INTRTX	Transmit Interrupt Register
USB_INTRRX	Receive Interrupt Register
USB_INTRTXE	Transmit Interrupt Enable Register
USB_INTRRXE	Receive Interrupt Enable Register
USB_IRQ	Common Interrupts Register
USB_IEN	Common Interrupts Enable Register
USB_FRAME	Frame Number Register
USB_INDEX	Index Register
USB_TESTMODE	Testmode Register
USB_FIFOBn	FIFO Byte (8-Bit) Register
USB_FIFOHn	FIFO Half-Word (16-Bit) Register
USB_FIFOn	FIFO Word (32-Bit) Register
USB_DEV_CTL	Device Control Register
USB_TXFIFOSZ	Transmit FIFO Size Register
USB_RXFIFOSZ	Receive FIFO Size Register
USB_TXFIFOADDR	Transmit FIFO Address Register
USB_RXFIFOADDR	Receive FIFO Address Register
USB_EPINFO	Endpoint Information Register
USB_RAMINFO	RAM Information Register
USB_LINKINFO	Link Information Register
USB_VPLEN	VBUS Pulse Length Register
USB_HS_EOF1	High-Speed EOF 1 Register
USB_FS_EOF1	Full-Speed EOF 1 Register
USB_LS_EOF1	Low-Speed EOF 1 Register

Table 22-2: ADSP-BF60x USB Register List (Continued)

Name	Description
USB_SOFT_RST	Software Reset Register
USB_MPn_TXFUNCADDR	MPn Transmit Function Address Register
USB_MPn_TXHUBADDR	MPn Transmit Hub Address Register
USB_MPn_TXHUBPORT	MPn Transmit Hub Port Register
USB_MPn_RXFUNCADDR	MPn Receive Function Address Register
USB_MPn_RXHUBADDR	MPn Receive Hub Address Register
USB_MPn_RXHUBPORT	MPn Receive Hub Port Register
USB_EPn_TXMAXP	EPn Transmit Maximum Packet Length Register
USB_EP0_CSRn_H	EP0 Configuration and Status (Host) Register
USB_EP0_CSRn_P	EP0 Configuration and Status (Peripheral) Register
USB_EPn_TXCSR_H	EPn Transmit Configuration and Status (Host) Register
USB_EPn_TXCSR_P	EPn Transmit Configuration and Status (Peripheral) Register
USB_EPn_RXMAXP	EPn Receive Maximum Packet Length Register
USB_EPn_RXCSR_H	EPn Receive Configuration and Status (Host) Register
USB_EPn_RXCSR_P	EPn Receive Configuration and Status (Peripheral) Register
USB_EP0_CNTn	EP0 Number of Received Bytes Register
USB_EPn_RXCNT	EPn Number of Bytes Received Register
USB_EPn_TXTYPE	EPn Transmit Type Register
USB_EP0_TYPEn	EP0 Connection Type Register
USB_EP0_NAKLIMITn	EP0 NAK Limit Register
USB_EPn_TXINTERVAL	EPn Transmit Polling Interval Register
USB_EPn_RXTYPE	EPn Receive Type Register
USB_EPn_RXINTERVAL	EPn Receive Polling Interval Register
USB_EP0_CFGDATAn	EP0 Configuration Information Register

Table 22-2: ADSP-BF60x USB Register List (Continued)

Name	Description
USB_DMA_IRQ	DMA Interrupt Register
USB_DMAn_CTL	DMA Channel n Control Register
USB_DMAn_ADDR	DMA Channel n Address Register
USB_DMAn_CNT	DMA Channel n Count Register
USB_RQPKTCNTn	EPn Request Packet Count Register
USB_CT_UCH	Chirp Timeout Register
USB_CT_HHSRTN	Host High Speed Return to Normal Register
USB_CT_HSBT	High Speed Timeout Register
USB_LPM_ATTR	LPM Attribute Register
USB_LPM_CTL	LPM Control Register
USB_LPM_IEN	LPM Interrupt Enable Register
USB_LPM_IRQ	LPM Interrupt Status Register
USB_LPM_FADDR	LPM Function Address Register
USB_VBUS_CTL	VBUS Control Register
USB_BAT_CHG	Battery Charging Control Register
USB_PHY_CTL	PHY Control Register
USB_PLL_OSC	PLL and Oscillator Control Register

ADSP-BF60x USB Interrupt List

Table 22-3: ADSP-BF60x USB Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
USB0 Status/FIFO Data Ready	122		LEVEL
USB0 DMA Status/Transfer Complete	123		LEVEL

ADSP-BF60x USB Trigger List

Table 22-4: ADSP-BF60x USB Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
USB0 DMA Status/Transfer Complete	69	LEVEL

Table 22-5: ADSP-BF60x USB Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
None		

USB Block Diagram

The USB block diagram shows the functional blocks within the USB. For more information about the blocks, see the *USB Functional Description*.

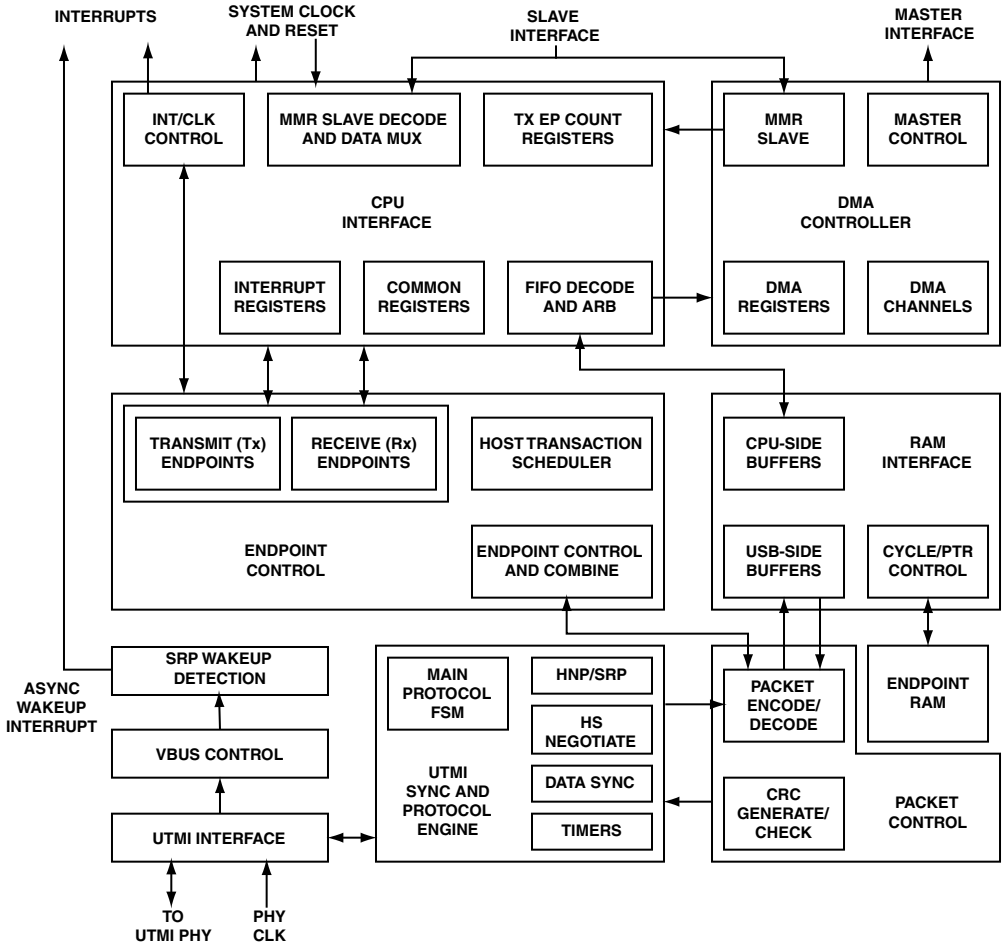


Figure 22-1: USB OTG Controller Block Diagram

USB Definitions

A list of common USB terms and their definitions as used in this specification and with respect to the USB controller follows:

'A' Device

The USB device with a mini-A plug inserted into its receptacle. The 'A' device always supplies power to VBUS.

'B' Device

The USB device with a standard-B or mini-B plug inserted into its receptacle. The B device starts a session as the peripheral.

Bi-directional endpoint

An endpoint that can concurrently support receive and transfer packets.

Control endpoint

An endpoint that is solely used for transfer of USB control packets for setup and configuration. In all USB devices, the control endpoint refers to the bi-directional endpoint 0.

Dual role device

A USB device that can operate either as the USB host in an OTG session or as a traditional USB peripheral.

Endpoint

A single physical communication channel for USB, implemented as a FIFO and control logic for that endpoint. Each endpoint has an associated USB transfer type, maximum packet size, bandwidth requirement, endpoint number, and (often) a fixed transfer direction.

Frame

A regular, fixed 1ms time slot that can contain several transactions. The transfer type determines what transactions are permitted for a given endpoint.

HNP

Host negotiation protocol. Part of the USB OTG Supplement that allows the host function to be transferred between two connected dual role devices.

Packet

The lowest level of data exchange on USB. The size is determined by the transfer type and buffer size of the USB peripheral.

PHY

The PHY is a transceiver circuit that implements the physical layer of USB. For full speed USB OTG this includes line drivers and receivers, pull-up/pull-down resistors as well as device ID and VBUS level detection.

Session

A period during which USB transfers take place within an OTG connection. This can be initiated by the

'A' device (by driving VBUS) or 'B' device (by initiating SRP). VBUS is powered during a session.

SRP

Session request protocol. Part of the USB OTG Supplement that allows a 'B' device to turn on VBUS and initiate a USB session.

Transaction

Collection of one or more packets in sequence

Transfer

Collection of one or more transfers in sequence

Unidirectional endpoint

Endpoint with its direction fixed in a single direction (for example, it can only receive packets from the USB) in both host and peripheral modes.

USB References

The following references provide further information regarding the USB.

- *On-The-Go Supplement to the USB 2.0 Specification*, Rev 1.0a, June 24, 2003, USB-IF
- *Universal Serial Bus Specification 2.0*

USB Operating Modes

The USB OTG interface may operate in peripheral mode or host mode.

When the USB controller is operating in peripheral mode, the controller may be attached to a conventional host (such as a personal computer) or another OTG device operating in host mode. The second device can be high-speed or full-speed. When linked to another peripheral device, the USB controller can also act as the host, and if the other device is also a dual role controller, the two devices can switch roles as required.

The role taken by the USB controller depends on the way the devices are cabled together. Each USB cable has an A and a B device end. If the A end of the cable is plugged into the device containing the USB controller, the USB controller takes the role of the host device and goes into host mode (in this case the `USB_DEV_CTL.HOSTMODE` bit is set to 1). If the B of the cable is plugged in, the USB controller goes instead into peripheral mode (and the `USB_DEV_CTL.HOSTMODE` bit remains at 0).

When both devices contain dual role controllers, signaling may be used to switch the roles of the two devices, without switching the cable connecting the two devices. The conditions under which the USB controller may switch between peripheral and host mode are detailed in *Host Negotiation Protocol*.

NOTE: The USB controller's multi-point capability is associated with a range of registers recording the allocation of device functions to individual USB controller's endpoints and device function characteristics such as endpoint number, operating speed and transaction type on an endpoint-by-endpoint basis. Although principally associated with the use of the USB controller as the host to a number of devices, these registers also need to be set when the core is used as the host for a single target device.

To enable the USB:

1. Configure the USB PLL multiplier settings in the USB PLL control register. Check the processor data sheet for the input clock frequency requirements.
2. Enable the USB PHY by setting the USB_PHY_CTL.EN bit.
3. Poll the bit in the USB PLL control register to ensure that the USB PLL has locked to the new frequency.

Peripheral Mode

USB OTG interface operations for the peripheral mode differ from host mode in a number of ways. The following sections describe peripheral mode operations.

Endpoint Setup

In peripheral mode, there are a few endpoint-specific configuration bits that are used when setting up an endpoint for transfer for all types of peripheral transfer. They determine how the processor core interacts with the endpoint FIFO.

One key parameter required before transfer can occur through an endpoint is the maximum USB packet size that the endpoint can support. This value is set by the software and depends on a variety of system constraints. These include the size of hardware FIFO available and system latencies as well as the USB transfer type and class being used. The USB_EPn_TXMAXP or USB_EPn_RXMAXP registers define the maximum amount of data that can be transferred to the selected endpoint in a single frame, and the value must match the programmed maximum individual packet size (*MaxPktSize*) of the standard endpoint descriptor for the endpoint.

For transmit endpoints, the maximum packet size is programmed using the USB_EPn_TXMAXP. For receive endpoints, the USB_EPn_RXMAXP register is used. The maximum packet size must not exceed the actual hardware endpoint FIFO size. The sizes of the transmit/receive FIFOs, as well as, single or double buffered mode for Endpoints 1 to 11 are determined by the settings in the corresponding USB_RXFIFOSZ or USB_TXFIFOSZ register. Because the USB controller uses a 32-bit interface, the value chosen for *MaxPktSize* should be an even number, as this selection simplifies transferring data between FIFOs and the processor core.

Additional setup parameters are configured using the USB_EPn_TXCSR_H or USB_EPn_RXCSR_H register (depending on whether the endpoint in question is receive or transmit). The USB_EPn_RXCSR_H.DMAREQEN bit in this register is used to enable the assertion of the appropriate DMA request whenever the endpoint is able to receive or transmit another packet. The USB_EPn_RXCSR_H.AUTOCLR and USB_EPn_

RXCSR_H.AUTOREQ bits can be used to automatically set the FIFO ready triggers (USB_EPn_RXCSR_H.RXPKTRDY and USB_EPn_TXCSR_H.TXPKTRDY) whenever a packet is transferred to streamline DMA operation for transfers that span multiple packets. Note, however, that USB_EPn_RXCSR_H.AUTOCLR and USB_EPn_RXCSR_H.AUTOREQ cannot be used with high-bandwidth endpoints. Refer to the "Register Descriptions" section for more details on the endpoint control and status registers.

IN Transactions as a Peripheral

When the USB controller is operating in peripheral mode, data for IN transactions is handled through the transmit FIFOs. The maximum size of data packet that may be placed in a transmit endpoint's FIFO for transmission is programmable and (where applicable) is determined by the value written to the USB_EPn_TXMAXP register for that endpoint (maximum payload multiplied by the number of transactions per micro-frame).

Note that the maximum packet size set for any endpoint must not exceed the FIFO size. (See *FIFO Configuration*.)

ATTENTION: Do not write to the USB_EPn_TXMAXP register while there is data in the FIFO, as unexpected results may occur.

The two types packet buffering sued for IN transactions are described below.

Single packet buffering. Set the USB_EPn_TXCSR_P.TXPKTRDY bit as each packet to be sent is loaded into the transmit FIFO. If the USB_EPn_TXCSR_P.AUTOSET bit is set, the USB_EPn_TXCSR_P.TXPKTRDY bit is automatically set when a maximum-sized packet is loaded into the FIFO. For packet sizes less than the maximum, and where auto set may not be used (high-bandwidth isochronous/interrupt transactions) always set the USB_EPn_TXCSR_P.TXPKTRDY bit manually (for example by the processor core).

When the USB_EPn_TXCSR_P.TXPKTRDY bit is set, either manually or automatically, the USB_EPn_TXCSR_P.NEFIFO bit is also set and the packet is ready to be sent. When the packet is successfully sent, both the USB_EPn_TXCSR_P.TXPKTRDY and USB_EPn_TXCSR_P.NEFIFO bits are cleared and the appropriate transmit endpoint interrupt is generated (if enabled). The next packet can then be loaded into the FIFO.

Double packet buffering. Set the USB_EPn_TXCSR_P.TXPKTRDY bit as each packet to be sent is loaded into the transmit FIFO. If the USB_EPn_TXCSR_P.AUTOSET bit is set, the USB_EPn_TXCSR_P.TXPKTRDY bit is automatically set when a maximum-sized packet is loaded into the FIFO. For packet sizes less than the maximum, the USB_EPn_TXCSR_P.TXPKTRDY bit always has to be set manually (for example, set by the processor core).

When the USB_EPn_TXCSR_P.TXPKTRDY bit is set, either manually or automatically, the USB_EPn_TXCSR_P.NEFIFO bit also is set. The USB_EPn_TXCSR_P.TXPKTRDY bit is then immediately cleared (and an interrupt generated, if enabled). A second packet can now be loaded into the transmit FIFO and the USB_EPn_TXCSR_P.TXPKTRDY bit is set again (either manually or automatically if the packet is the maximum size). Both packets are now ready to be sent.

When the first packet is successfully sent, the USB_EPn_TXCSR_P.TXPKTRDY bit is cleared and the appropriate transmit endpoint interrupt is generated (if enabled) to signal that another packet can now be loaded into the transmit FIFO. The state of the USB_EPn_TXCSR_P.NEFIFO bit at this point indicates how many

packets may be loaded. If the USB_EPn_TXCSR_P.NEFIFO bit is set then there is another packet in the FIFO and only one more packet can be loaded. If the USB_EPn_TXCSR_P.NEFIFO bit is cleared then there are no packets in the FIFO and two more packets can be loaded.

OUT Transactions as a Peripheral

When the USB controller is operating in peripheral mode, data for OUT transactions are handled through the USB controller's receive FIFOs.

The maximum amount of data received by a receive endpoint in any frame or micro-frame (in high-speed mode) is programmable and is determined by the value written to the USB_EPn_RXMAXP register for that endpoint. This is the maximum payload multiplied by the number of transactions per micro-frame (where applicable). The maximum packet size must not exceed the FIFO size.

If the size of the receive endpoint FIFO is less than twice the maximum packet size for this endpoint (as set in the USB_EPn_RXMAXP register), only one data packet can be buffered in the FIFO and single buffering is selected. When a packet is received and placed in the receive FIFO, the USB_EPn_RXCSR_P.RXPKTRDY bit and the USB_EPn_RXCSR_P.FIFOFULL bit are set and the appropriate receive endpoint interrupt is generated (if enabled) to signal that a packet can now be unloaded from the FIFO. After the packet is unloaded, clear the USB_EPn_RXCSR_P.RXPKTRDY bit to allow further packets to be received. If the USB_EPn_RXCSR_P.AUTOCLR bit i is set and a maximum-sized packet is unloaded from the FIFO, the USB_EPn_RXCSR_P.RXPKTRDY bit is cleared automatically. The USB_EPn_RXCSR_P.FIFOFULL bit is also cleared. For packet sizes less than the maximum, clear the USB_EPn_RXCSR_P.RXPKTRDY bit manually (for example by the processor core).

If double packet buffering is enabled, then two data packets can be buffered. When the first packet to be received is loaded into the receive FIFO, the USB_EPn_RXCSR_P.RXPKTRDY bit is set and the appropriate receive endpoint interrupt is generated (if enabled) to signal that a packet can now be unloaded from the FIFO. Note that the USB_EPn_RXCSR_P.FIFOFULL bit is not set at this point. This bit is only set if a second packet is received and loaded into the receive FIFO.

After the first packet is unloaded, clear the USB_EPn_RXCSR_P.RXPKTRDY bit to allow further packets to be received. If the USB_EPn_RXCSR_P.AUTOCLR bit is set and a maximum-sized packet is unloaded from the FIFO, the USB_EPn_RXCSR_P.RXPKTRDY bit is cleared automatically. For packet sizes less than the maximum, clear the USB_EPn_RXCSR_P.RXPKTRDY bit manually (for example by the processor core).

If the USB_EPn_RXCSR_P.FIFOFULL bit was set to 1 when USB_EPn_RXCSR_P.RXPKTRDY is cleared, the USB controller first clears the USB_EPn_RXCSR_P.FIFOFULL bit. The controller then sets the USB_EPn_RXCSR_P.RXPKTRDY bit again, indicating that there is another packet waiting in the FIFO to be unloaded.

High-Bandwidth Isochronous/Interrupt Transactions

High-bandwidth isochronous/interrupt transactions use much the same protocol as other isochronous/interrupt transactions. There are, however, some special features to conducting high-bandwidth transactions.

- When setting the maximum packet size handled by the endpoint in the USB_EPn_TXMAXP/USB_EPn_RXMAXP registers, the maximum number of transactions per micro-frame also needs to be set via the USB_EPn_TXMAXP.MULTM1 and USB_EPn_RXMAXP.MULTM1 bits of these registers.

  This maximum number of transactions (2 or 3) also represents the maximum number of sections in which any single high-bandwidth packet can be transferred, which in turn sets the maximum size of the packet to 2 or 3 times the maximum payload specified for the endpoint in the same register.

NOTE:

The maximum payload that can be sent in any transaction is 1K byte.

NOTE:

- When sending packets, set the USB_EPn_TXCSR_P.TXPKTRDY bit using the application software. Similarly, when unloading packets from the receive endpoint FIFO, clear the USB_EPn_RXCSR_P.RXPKTRDY bit using the application software.

CAUTION:

The AutoSet and AutoClear functions cannot be used to set and clear these bits in high-bandwidth transactions.

CAUTION:

- The transmission of packets as a number of sections introduces a further type of error – the transmission of incomplete packets.

  For transmit endpoints, transmitting incomplete packets principally applies when the interface is in peripheral mode and occurs when the transmission fails to receive enough IN tokens from the host to send all the parts of the data packet. It can also apply to high-bandwidth interrupt transactions in host mode where the core does not receive any response from the device to which the packet is being sent. In both cases, the USB_EPn_TXCSR_P.INCOMPTX bit is set.

  For receive endpoints, the incomplete packet issue occurs when the PIDs of the received parts of the data packet show that one or more parts of the data packet have not been received. When this happens, the USB_EPn_RXCSR_P.INCOMPRX bit is set. Usually this bit is set in peripheral mode. However it can also be set in host mode (using the USB_EPn_RXCSR_H.INCOMPRX bit) only if the device that the USB is communicating with fails to respond in accordance with the USB protocol.

High Bandwidth Isochronous/Interrupt IN Endpoints

In high-speed mode, transmit endpoints set up for high-bandwidth isochronous/interrupt transactions can transmit up to three USB packets in any micro-frame, with a payload of up to 1024 bytes in each packet, corresponding to a data transfer rate of up to 3072 bytes per micro-frame.

The **High Bandwidth IN Endpoints** figure provides an overview of high-bandwidth IN endpoints in USB.

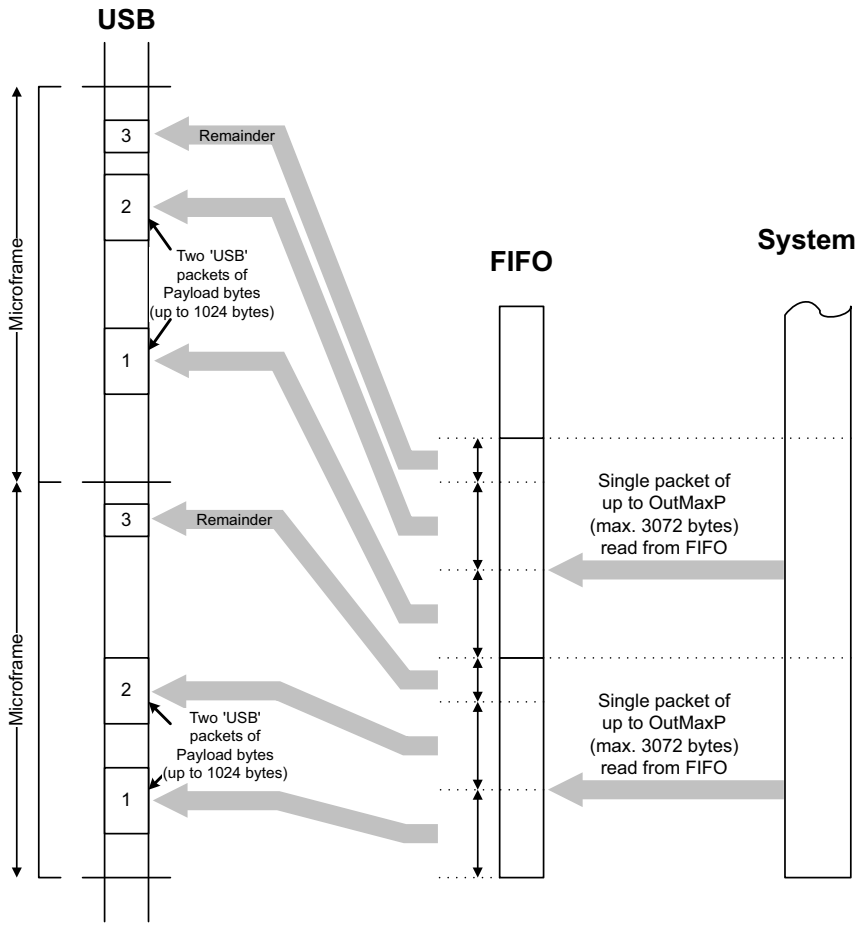


Figure 22-2: High Bandwidth IN Endpoints

The USB controller supports these transfers by permitting the loading of data packets with up to three times the normal packet size into the associated FIFO in a single transaction. From the point of view of the software in the processor core, the operation is then exactly as described above for single or double packet buffering (as appropriate) except that the `USB_EPn_TXCSR_P.TXPKTRDY` bit always needs to be set manually (for example by the processor core) as the auto set feature does not operate with high-bandwidth isochronous transfers.

Any data packet loaded into the FIFO that is larger than the maximum payload is automatically split into USB packets of the maximum payload, or smaller, for transmission over the USB. The number of USB packets transmitted per micro-frame and the maximum payload in each packet is defined using the `USB_EPn_TXMAXP.MAXPAY`bits to set the maximum payload in any USB packet and the `USB_EPn_`

TXMAXP.MULTM1 bits to set the maximum number of such packets that can be sent in one micro-frame (2 or 3). Together, these set the maximum size of packet that can be loaded into the FIFO.

At least one USB packet always is sent. The number of further USB packets sent in the same micro-frame depends on the amount of data loaded into the FIFO. The USB_EPn_TXCSR_P.TXPKTRDY bit is cleared and an interrupt is generated only when all the packets have been sent. Each USB packet is sent in response to an IN token. If, at the end of a micro-frame, the USB controller has not received enough IN tokens to send all the USB packets (for example, because one of the IN tokens received was corrupted), the remaining data is flushed from the FIFO. The USB_EPn_TXCSR_P.TXPKTRDY bit is cleared and the USB_EPn_TXCSR_P.INCOMPTX bit is set to indicate that not all of the data loaded into the FIFO was sent.

High Bandwidth Isochronous/Interrupt OUT Endpoints

In high-speed mode, isochronous receive endpoints can receive up to three USB packets in any micro-frame, with a payload of up to 1024 bytes in each packet, corresponding to a data transfer rate of up to 3072 bytes per micro-frame. High-bandwidth interrupt transactions can similarly be received in host mode, but there is no support for high-bandwidth interrupt transactions in peripheral mode.

The **High Bandwidth OUT Endpoints** figure shows an overview of high-bandwidth OUT endpoints.

The USB controller supports this rate by automatically combining all the USB packets received during a micro-frame into a single packet of up to 3 normal packets in size within the receive FIFO. From the point of view of the software in the processor core, the operation is then exactly as described as for single or double packet buffering (as appropriate), except the USB_EPn_RXCSR_P.RXPKTRDY bit always needs to be cleared manually (for example by the processor core) because the auto-clear function does not operate with high-bandwidth isochronous transfers.

The maximum number of USB packets that may be received in any micro-frame and the maximum payload of these packets are configured using the USB_EPn_RXMAXP.MAXPAY bits to set the maximum payload in any USB packet and the USB_EPn_RXMAXP.MULTM1 bits to set the maximum number of these packets that may be received in a micro-frame (2 or 3).

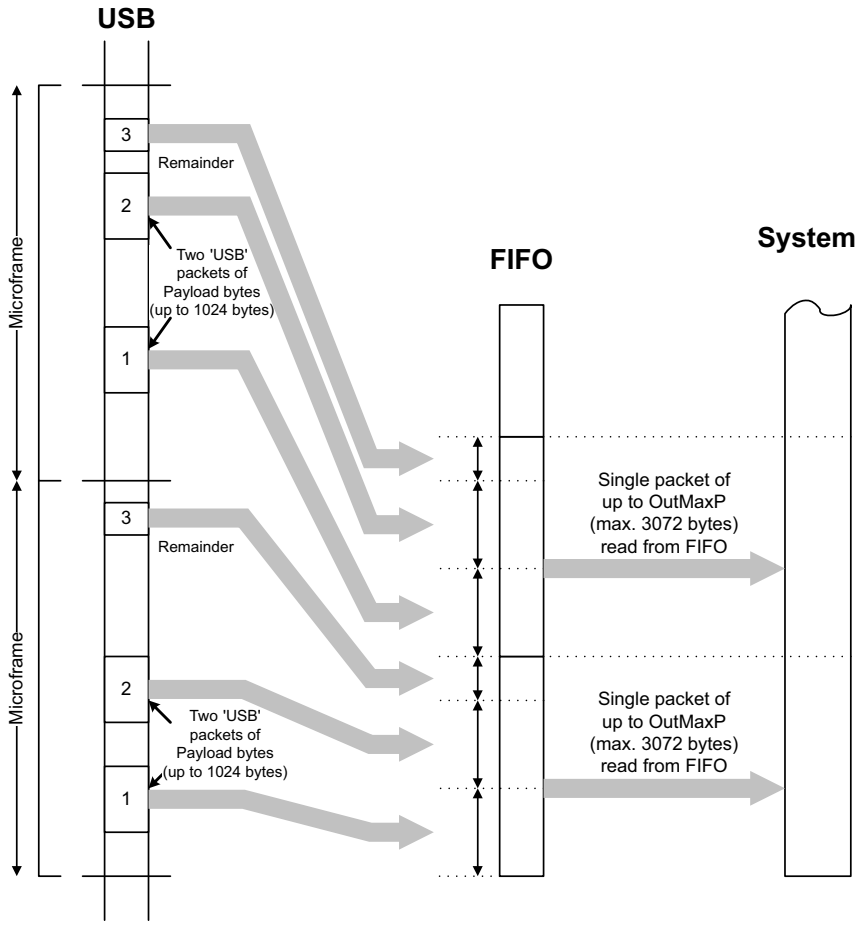


Figure 22-3: High Bandwidth OUT Endpoints

The number of USB packets sent in any micro-frame depends on the amount of data to be transferred, and is indicated through the PIDs used for the individual packets. If the indicated number of packets have not been received by the end of a micro-frame, the `USB_EPn_RXCSR_P.INCOMPRX` bit is set to indicate that the data in the FIFO is incomplete. An interrupt is still generated to allow the data that has been received to be read from the FIFO.

Peripheral Transfer Work Flows

The USB transfer types (control, bulk, isochronous and interrupt transfers) each have significantly different system requirements as well as individual USB transfer-specific features. This dictates that they are each dealt with slightly differently in software. For these reasons, there is no uniform way of doing transfers across all transfer types using the USB controller.

The following sections provide some guidelines for peripheral mode transfer flows for each of the transfer types, in both IN (transmit) and OUT (receive) directions. In the case of bulk endpoints, the optimal transfer flow differs depending on whether the final size of the transfer is known or unknown. Whether the transfer size is known or not depends on the USB driver class being used. Some define the complete transfer size, and others operate on a packet-by-packet basis using a short packet (a packet of less than the

value configured in the USB_EPn_TXMAXP register or less than the value configured in the USB_EPn_RXMAXP register) to denote the end of a transfer.

Each of the work flows use the following common steps.

1. Configure the endpoint control and status registers and the USB_EPn_TXMAXP or USB_EPn_RXMAXP value.
2. Configure the appropriate data transfer mechanism (DMA or interrupt setup).
3. Data transfer occurs.

The work flows do not describe the USB controller's actions immediately preceding the endpoint setup (for example, the reception of an IN/OUT token from the host, token validity checking, or NAK generation, among others). Note also that there is currently no error-handling contained in the work flows (for example, checking the USB_EPn_RXCSR_P.FIFOFULL bit before writing data).

The terms packets, frames and transfers are used in the proceeding sections with their strict USB definitions (see *USB Definitions*).

Control Transactions as a Peripheral

Endpoint 0 is the main control endpoint of the USB controller. As such, the routines required to service Endpoint 0 are more complicated than those required to service other endpoints.

The software is required to handle all the standard device requests that may be sent or received through Endpoint 0. These are described in *Universal Serial Bus Specification*, Revision 2.0, Chapter 9. The protocol for these device requests involves different numbers and types of transactions per transfer. To accommodate this, the processor needs to take a state machine approach to command decoding and handling.

The standard device requests received by a USB peripheral can be divided into three categories: zero data requests (in which all the information is included in the command), write requests (in which the command will be followed by additional data), and read requests (in which the device is required to send data back to the host).

This following sections describe the sequence of actions that the software must perform to process these different types of device request.

Write Requests

Write requests involve an additional packet (or packets) of data being sent from the host after the 8-byte command. An example of a write standard device request is SET_DESCRIPTOR.

As with all requests, the sequence of events begins when the software receives an endpoint 0 interrupt. The USB_EPn_RXCSR_P.RXPKTRDY bit is also set. The 8-byte command should then be read from the endpoint 0 FIFO and decoded.

As with a zero data request, the USB_EP0_CSRn_P register should then be written to set the USB_EP0_CSRn_P.SPKTRDY bit (indicating that the command is read from the FIFO) but in this case the USB_EP0_CSRn_P.DATAEND bit should not be set (indicating that more data is expected).

When a second endpoint 0 interrupt is received, the USB_EP0_CSRn_P register is read to check the endpoint status. The USB_EP0_CSRn_P.RXPKTRDY bit is set to indicate that a data packet is received. The USB_EP0_CNTn register should then be read to determine the size of this data packet. The data packet can then be read from the endpoint 0 FIFO.

If the length of the data associated with the request (indicated by the WLENGTH field in the command) is greater than the maximum packet size for endpoint 0, further data packets are sent. In this case, the USB_EP0_CSRn_P.SPKTRDY bit is set, but the USB_EP0_CSRn_P.DATAEND bit should not be set.

When all the expected data packets have been received, the USB_EP0_CSRn_P register is written to set the USB_EP0_CSRn_P.SPKTRDY bit and to set the USB_EP0_CSRn_P.DATAEND bit (indicating that no more data is expected).

When the host moves to the status stage of the request, another endpoint 0 interrupt is generated to indicate that the request has completed. No further action is required from the software—the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it has been decoded, the USB_EP0_CSRn_P register should be written to set the USB_EP0_CSRn_P.SPKTRDY bit and to set the USB_EP0_CSRn_P.SENDSTALL bit. When the host sends more data, the USB controller will send a stall to tell the host that the request was not executed. An endpoint 0 interrupt is generated and the USB_EP0_CSRn_P.SENTSTALL bit is set.

If the host sends more data after the USB_EP0_CSRn_P.DATAEND has been set, then the USB controller sends a stall. An endpoint 0 interrupt is generated and the USB_EP0_CSRn_P.SENTSTALL bit is set.

Read Requests

Read requests have a packet (or packets) of data sent from the function to the host after the 8-byte command. Examples of standard device requests for read are: GET_CONFIGURATION, GET_INTERFACE, GET_DESCRIPTOR, GET_STATUS, and SYNCH_FRAME.

As with all requests, the sequence of events will begin when the software receives an endpoint 0 interrupt. The USB_EPn_RXCSR_P.RXPKTRDY bit is also set. The 8-byte command should then be read from the endpoint 0 FIFO and decoded. Write the USB_EP0_CSRn_P.SPKTRDY bit (indicating that the command has read from the FIFO).

The data to be sent to the host should then be written to the endpoint 0 FIFO. If the data to be sent is greater than the maximum packet size for endpoint 0, only the maximum packet size should be written to the FIFO. The USB_EP0_CSRn_P.TXPKTRDY bit should then be set (indicating that there is a packet in the FIFO to be sent). When the packet has been sent to the host, another endpoint 0 interrupt is generated and the next data packet can be written to the FIFO.

When the last data packet has been written to the FIFO, the USB_EP0_CSRn_P register should be written to set the USB_EP0_CSRn_P.TXPKTRDY bit and to set the USB_EP0_CSRn_P.DATAEND bit (indicating that there is no more data after this packet).

When the host moves to the status stage of the request, another endpoint 0 interrupt will be generated to indicate that the request has completed. No further action is required from the software-the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it has been decoded, the USB_EP0_CSRn_P register should be written to set the USB_EP0_CSRn_P.SPKTRDY bit and to set the USB_EP0_CSRn_P.SENDSTALL bit. When the host requests data, the USB controller will send a stall to tell the host that the request was not executed. An endpoint 0 interrupt will be generated and the USB_EP0_CSRn_P.SENTSTALL bit is set.

If the host requests more data after USB_EP0_CSRn_P.DATAEND has been set, then the USB controller sends a stall. An endpoint 0 interrupt is generated and the USB_EP0_CSRn_P.SENTSTALL bit is set.

Zero Data Requests

Zero data requests have all their information included in the 8-byte command and require no additional data to be transferred.

Examples of zero data standard device requests are: SET_FEATURE, CLEAR_FEATURE, SET_ADDRESS, SET_CONFIGURATION, and SET_INTERFACE.

As with all requests, the sequence of events begins when the software receives an endpoint 0 interrupt. The USB_EP0_CSRn_P.RXPKTRDY bit will also have been set. The 8-byte command should then be read from the endpoint 0 FIFO, decoded and the appropriate action taken. For example if the command is SET_ADDRESS, the 7-bit address value contained in the command is written to the USB_FADDR register.

The USB_EP0_CSRn_P.SPKTRDY bit should be set (indicating that the command is read from the FIFO) and the USB_EP0_CSRn_P.DATAEND bit should be set (indicating that no further data is expected for this request).

When the host moves to the status stage of the request, a second endpoint 0 interrupt is generated, indicating that the request has completed. No further action is required from the software—the second interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when it is decoded, the USB_EP0_CSRn_P.SPKTRDY bit is set which sets the USB_EP0_CSRn_P.SENDSTALL bit. When the host moves to the status stage of the request, the USB controller sends a stall to tell the host that the request was not executed. A second endpoint 0 interrupt is generated and the USB_EP0_CSRn_P.SENTSTALL bit is set.

If the host sends more data after the USB_EP0_CSRn_P.DATAEND bit is set, then the USB controller sends a stall. An endpoint 0 interrupt is generated and the USB_EP0_CSRn_P.SENTSTALL bit is set.

ENDPOINT 0 States

When the USB is operating as a peripheral, the Endpoint 0 control needs three modes (IDLE, TX and RX) corresponding to the different phases of the control transfer and the states Endpoint 0 enters for the different phases of the transfer. (See *Endpoint 0 Service Routine as Peripheral*.)

The default mode on power-up or reset should be IDLE. The RxPktRdy bit becoming set when Endpoint 0 is in IDLE state indicates a new device request. Once the device request is unloaded from the FIFO, the USB decodes the descriptor to find whether there is a data phase and, if so, the direction of the data phase of the control transfer (in order to set the FIFO direction).

Depending on the direction of the data phase, Endpoint 0 goes into either TX state or RX state. If there is no data phase, Endpoint 0 remains in IDLE state to accept the next device request.

The processor needs to take different actions at the different phases of the possible transfers (for example, "Loading the FIFO", "Setting TxPktRdy") are indicated in the *Endpoint 0 Control States* figure. Note that the USB changes the FIFO direction depending on the direction of the data phase, independently of the processor.

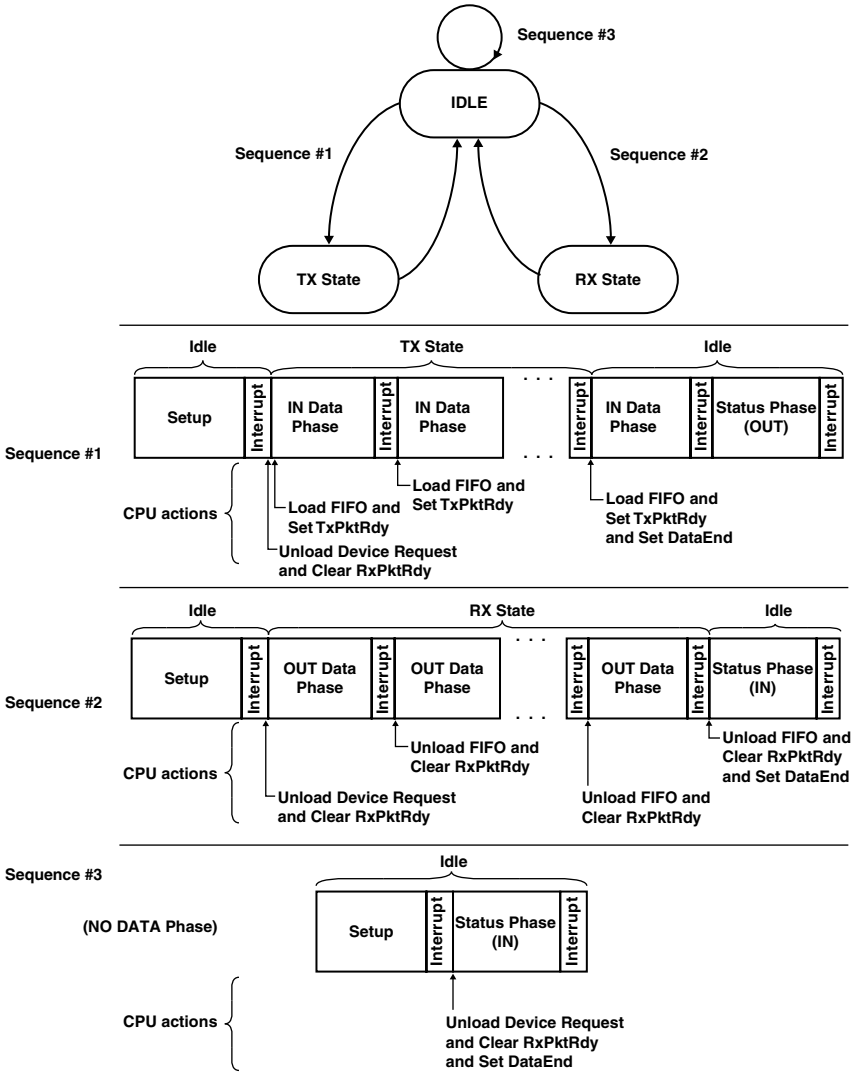


Figure 22-4: Endpoint 0 Control States

Endpoint 0 Service Routine as Peripheral

An endpoint 0 interrupt is generated:

- When the USB controller sets the USB_EP0_CSRn_P.RXPKTRDY bit after a valid token has been received and data has been written to the FIFO.
- When the USB controller clears the USB_EP0_CSRn_P.TXPKTRDY bit after the data packet in the FIFO has been successfully transmitted to the host.
- When the USB controller sets the USB_EP0_CSRn_P.SENTSTALL bit after a control transaction is ended due to a protocol violation.
- When the USB controller sets the USB_EP0_CSRn_P.SETUPEND bit because a control transfer has ended before USB_EP0_CSRn_P.DATAEND is set.

Whenever the endpoint 0 service routine is entered, the firmware must first check whether the current control transfer has been ended due to either a stall condition or a premature end-of-control transfer. If the control transfer ends due to a stall condition, the USB_EP0_CSRn_P.SENTSTALL bit is set. If the control transfer ends due to a premature end-of-control transfer, the USB_EP0_CSRn_P.SETUPEND is be set. In either case, the firmware should abort processing the current control transfer and set the state to IDLE.

Once the firmware has determined that the interrupt was not generated by an illegal bus state, the next action depends on the endpoint state.

If endpoint 0 is in IDLE state, the only valid reason an interrupt can be generated is as a result of the core receiving data from the USB bus. The service routine must check for this by testing the USB_EP0_CSRn_P.RXPKTRDY bit. If this bit is set, then the core has received a SETUP packet. This must be unloaded from the FIFO and decoded to determine the action the core must take. Depending on the command contained within the SETUP packet, endpoint 0 enters one of the following three states.

- If the command is a single packet transaction (SET_ADDRESS, SET_INTERFACE and the others) without a data phase, the endpoint remains in the IDLE state.
- If the command has an OUT data phase (SET_DESCRIPTOR and others), the endpoint enter the RX state.
- If the command has an IN data phase (GET_DESCRIPTOR and others), the endpoint enters the TX state.

If the endpoint is in TX state, the interrupt indicates that the core has received an IN token and data from the FIFO has been sent. The firmware must respond to this either by placing more data in the FIFO if the host is still expecting more data[1] or by setting the USB_EP0_CSRn_P.DATAEND bit to indicate that the data phase is complete. Once the data phase of the transaction has been completed, endpoint 0 should be returned to the IDLE state to await the next control transaction.

If the endpoint is in the RX state, the interrupt indicates that a data packet has been received. The firmware must respond by unloading the received data from the FIFO. The firmware must then determine whether

1.Command transactions all include a field that indicates the amount of data the host expects to receive or is going to send.

it has received all of the expected data. If it has, the firmware should set the USB_EP0_CSRn_P.DATAEND bit and return endpoint 0 to IDLE state. If more data is expected, the firmware should set the USB_EP0_CSRn_P.SPKTRDY bit to indicate that it has read the data in the FIFO and leave the endpoint in the RX state.

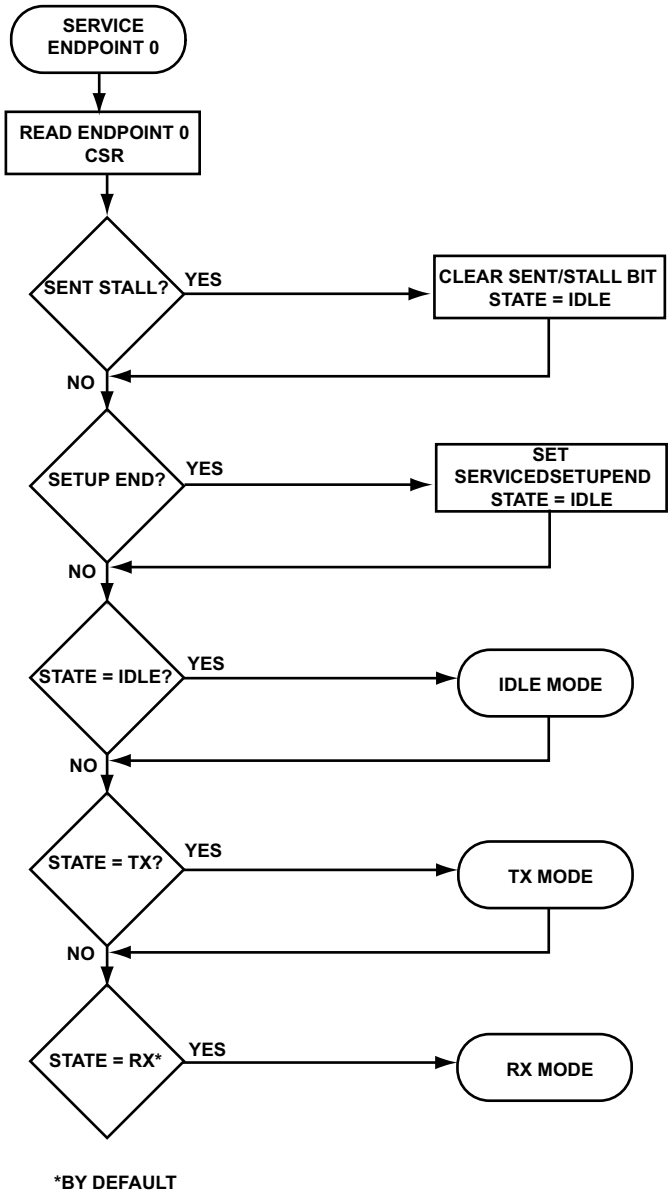


Figure 22-5: Endpoint 0 Service Routine

Idle Mode

The endpoint 0 control must select the IDLE mode at power-on or reset. The endpoint 0 control should return to this mode when the RX and TX modes are terminated.

And, as shown in the **Endpoint 0 Idle Mode (Setup Phase)** figure, this is also the mode in which the SETUP phase of control transfer is handled.

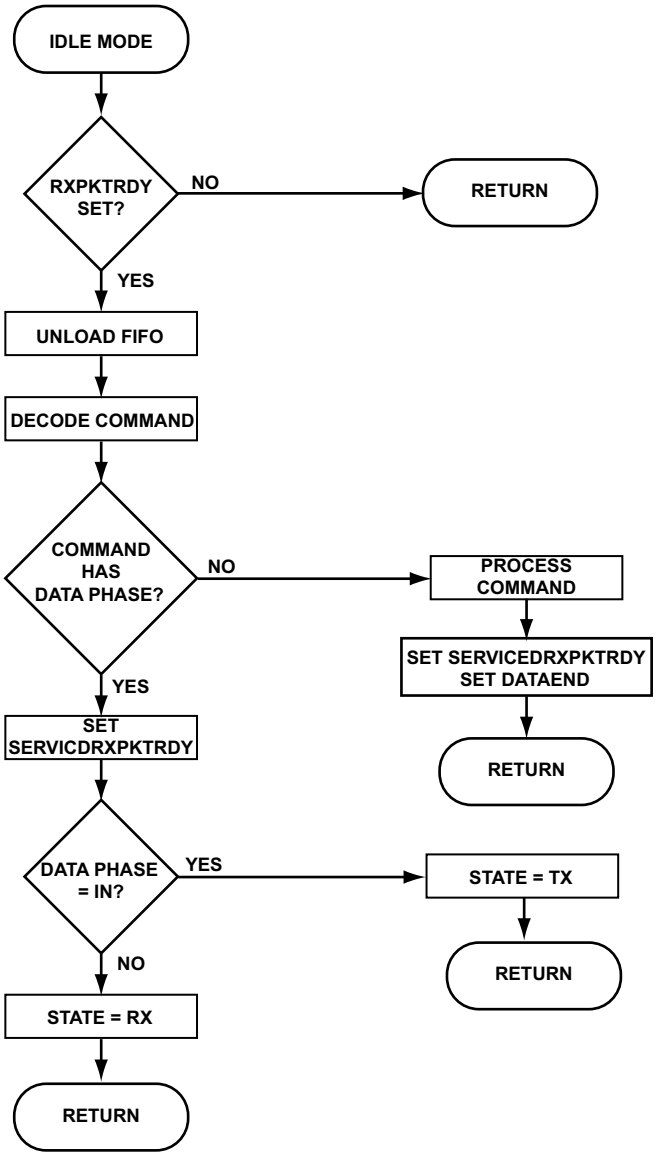


Figure 22-6: Endpoint 0 Idle Mode (Setup Phase)

TX Mode

As shown in the **Endpoint 0 TX Mode** figure when the endpoint is in TX state, all arriving IN tokens need to be treated as part of a data phase until the required amount of data has been sent to the host. If either a SETUP or an OUT token is received while the endpoint is in the TX state, a `USB_EP0_CSRn_P.SETUPEND` condition occurs since the core expects only IN tokens.

Three events can cause the TX mode to terminate before the expected amount of data has been sent:

- The host sends an invalid token which sets the `USB_EP0_CSRn_P.SETUPEND` bit.
- The firmware sends a packet containing less than the maximum packet size for endpoint 0.

- The firmware sends an empty data packet.

Until the transaction is terminated, when the firmware receives an interrupt which indicates that a packet has been sent from the FIFO, it simply loads the FIFO. An interrupt is generated when USB_EP0_CSRn_P.TXPKTRDY is cleared.

When the firmware forces the termination of a transfer (by sending a short or empty data packet), it should set the USB_EP0_CSRn_P.DATAEND bit to indicate to the core that the data phase is complete and that the core should receive an acknowledge packet next.

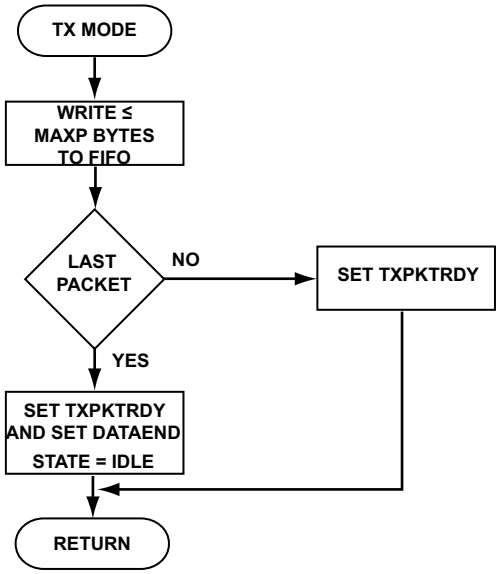


Figure 22-7: Endpoint 0 TX Mode

RX Mode

As shown in the **Endpoint 0 RX Mode** figure, In RX mode, all arriving data should be treated as part of a data phase until the expected amount of data has been received. If either a SETUP or an IN token is received while the endpoint is in RX state, a USB_EP0_CSRn_P.SETUPEND condition occurs since the core expects only OUT tokens.

Three events can cause the RX mode to terminate before the expected amount of data has been received:

- The host sends an invalid token causing a USB_EP0_CSRn_P.SETUPEND bit set.
- The host sends a packet which contains less than the maximum packet size for Endpoint 0.
- The host sends an empty data packet.

Until the transaction is terminated, when the firmware receives an interrupt which indicates that new data has arrived (USB_EP0_CSRn_P.RXPKTRDY bit set), it simply needs to unload the FIFO and clear USB_EP0_CSRn_P.RXPKTRDY by setting the USB_EP0_CSRn_P.SPKTRDY bit.

When the firmware detects the termination of a transfer (by receiving either the expected amount of data or an empty data packet), it should set the USB_EP0_CSRn_P.DATAEND bit to indicate to the core that the data phase is complete and that the core should receive an acknowledge packet next.

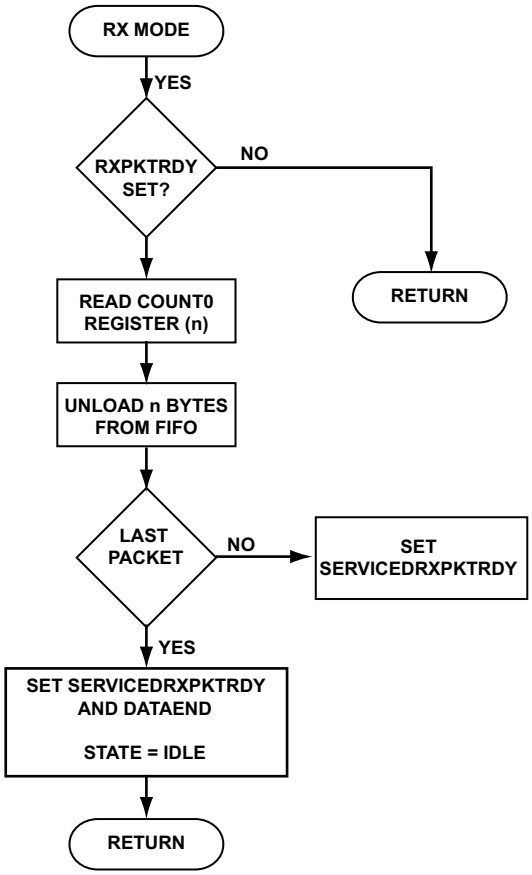


Figure 22-8: Endpoint 0 RX Mode

Peripheral Mode, Bulk IN, Transfer Size Known

PREREQUISITE:

For this process, the maximum individual packet size (*MaxPktSize*) in bytes and the complete transfer size (*TxferSize*) in bytes, must be known.

1. Load *MaxPktSize* into USB_TXMAXPn.
2. Set DMA_ENA = 1, AUTOSET_T = 1, ISO_T = 0, FRCDATATOG = 0 in USB_TXCSR.
3. Load *TxferSize* into USB_DMA_CNTn.
4. Configure the DMA controller to write the data into the corresponding TX FIFO address.
5. On each USB_DMAxINT transition, the DMA controller writes a new packet into the FIFO. TXPKTRDY is automatically set when each new packet is written.

6. Step 5 is repeated for each full packet of the transfer.
7. Even if the final packet is a short packet, the packet automatically is detected by the USB controller because TXPKTRDY is set.

Peripheral Mode, Bulk IN, Transfer Size Unknown

PREREQUISITE:

For this process, the maximum individual packet size (*MaxPktSize*) in bytes is assumed to be an even number of bytes.

1. Load *MaxPktSize* into USB_TXMAXPn.
2. Set DMA_ENA = 1, AUTOSET_T = 1, ISO_T = 0, FRCDATATOG = 0 in USB_TXCSR.
3. Configure the DMA controller to write *MaxPktSize*/2 half words into the corresponding TX FIFO address on each USB_DMAxINT.
4. Set up an ISR, sensitive to the DMA work-block-complete interrupt, that writes a remaining short packet into the TX FIFO using processor core DMA. Then set TXPKTRDY or simply send a zero-length packet by toggling TXPKTRDY.
5. On each USB_DMAxINT transition, the DMA controller writes a new packet into the FIFO. TXPKTRDY automatically is set when each new packet is written.
6. Step 5 is repeated for each full packet of the transfer.
7. The final short/zero-length packet is managed by the ISR from step 4.

Peripheral Mode, ISO IN, Small MaxPktSize

PREREQUISITE:

For this process, the maximum individual packet size (*MaxPktSize*) in bytes is less than 128 bytes and is an even number of bytes. Double buffering is assumed to be enabled, and the auto set feature unused (because packets are often less than *MaxPktSize*).

1. Load *MaxPktSize* into USB_TXMAXPn.
2. Set ISO_T = 1 in USB_TXCSR.
3. Preload the first two packets into the endpoint TX FIFO and set TXPKTRDY.
4. Set up an ISR, sensitive to the SOF_B interrupt, which writes a new packet into the TX FIFO and sets TXPKTRDY.
5. Set SOF_B = 1 in USB_INTRUSBE to generate an interrupt on each start-of-frame.
6. Step 5 is repeated for each ISO packet.

Peripheral Mode, ISO IN, Large MaxPktSize

PREREQUISITE:

For this process, the maximum individual packet size (*MaxPktSize*) in bytes is greater than 128 bytes and is an even number of bytes. Double buffering is assumed to be enabled, and the auto set feature unused (because packets are often less than *MaxPktSize*).

1. Load *MaxPktSize* into USB_TXMAXPn.
2. Set ISO_T = 1 in USB_TXCSR.
3. Set ISO_UPDATE = 1 in USB_POWER to prevent initial packet loaded into the FIFO from being transmitted on USB until the next 1ms frame.
4. Load the total number of bytes for the first two packets into USB_DMA_CNTn.
5. Configure the DMA controller to pre-load the two packets into the corresponding TX FIFO address. Set TXPKTRDY.
6. Set up an ISR, sensitive to the SOF_B interrupt, which writes a new packet into the TX FIFO by configuring the DMA controller to load the packet.
7. Set SOF_B = 1 in USB_INTRUSBE to generate an interrupt on each start-of-frame.
8. Step 7 is repeated for each ISO packet.

Peripheral Mode, Bulk OUT, Transfer Size Known

PREREQUISITE:

For this process, the maximum individual packet size (*MaxPktSize*) in bytes and the complete transfer size (*TxferSize*) in bytes must be known.

1. Load *MaxPktSize* into USB_RXMAXPn.
2. Set DMA_ENA = 1, AUTOCLEAR_R = 1, ISO_R = 0, FRCDATATOG = 0, DMAREQMODE_R = 0 in USB_RXCSR.
3. Configure the DMA controller to read the full *TxferSize*/2 half words from the corresponding RX FIFO address.
4. On each USB_DMAxINT transition, the DMA controller reads another packet from the FIFO. RXPKTRDY is automatically cleared by the USB controller when each new packet is read.
5. Step 5 is repeated for each full packet of the transfer.
6. If *TxferSize* is not an exact multiple of *MaxPktSize*, the final USB_DMAxINT transition causes the DMA controller to read out only the short packet that remains.

Peripheral Mode, Bulk OUT, Transfer Size Unknown

PREREQUISITE:

For this process, the maximum individual packet size (*MaxPktSize*) in bytes must be known.

1. Load *MaxPktSize* into USB_RXMAXPn.
2. Set DMA_ENA = 1, AUTOCLEAR_R = 1, ISO_R = 0, FRCDATATOG = 0, DMAREQMODE_R = 1 in USB_RXCSR.
3. Set the appropriate EPx_RX_E bit in USB_INTRRXE.
4. Configure the DMA controller to read *MaxPktSize*/2 half words from the corresponding RX FIFO address on each USB_DMAxINT transition.
5. Set up an ISR, sensitive to the RX interrupt, which reads USB_RXCOUNT and then transfers USB_RXCOUNT bytes (in half words) from the RX FIFO to the processor core. Depending on the number of bytes in the FIFO, this can be performed by configuring the DMA to read the data, or by reading it with the processor core.
6. On each USB_DMAxINT transition, the DMA controller reads a packet from the FIFO. RXPKTRDY is automatically cleared by the USB controller when each new packet is read.
7. Step 5 is repeated for each full packet of the transfer.
8. If a packet is received that is less than *MaxPktSize*, the RX interrupt goes high, and the ISR from step 5 reads out the remaining short packet.

Peripheral Mode, ISO OUT, Small MaxPktSize

PREREQUISITE:

For this process, the maximum individual packet size (*MaxPktSize*) in bytes is less than 128 bytes, and double buffering is assumed to be enabled.

1. Load *MaxPktSize* into USB_RXMAXPn.
2. Set ISO_R = 1 in USB_RXCSR.
3. Set up an ISR, sensitive to the SOF_B interrupt, that reads the FIFO_FULL bit, reads the USB_RXCOUNT status register, and finally removes one or two packets (equal to the USB_RXCOUNT number of bytes) from the FIFO then clears RXPKTRDY.
4. Set SOF_B = 1 in USB_INTRUSBE to generate an interrupt on each start-of-frame.
5. Step 4 is repeated for each ISO packet.

Peripheral Mode, ISO OUT, Large MaxPktSize

PREREQUISITE:

For this process, the maximum individual packet size (*MaxPktSize*) in bytes is greater than 128 bytes, and double buffering is assumed to be enabled.

1. Load *MaxPktSize* into the USB_RXMAXPn register.
2. Set the ISO_R = 1 in the USB_RXCSR register.
3. Set up an ISR, (sensitive to the SOF_B interrupt), that reads the FIFO_FULL bit, reads the USB_RXCOUNT status register, and finally configures the DMA controller to remove one or two packets (equal to the USB_RXCOUNT number of bytes) from the FIFO.
4. Set up an ISR, sensitive to the DMA work-block-complete interrupt to clear RXPKTRDY.
5. Set SOF_B = 1 in the USB_INTRUSBE register to generate an interrupt on each start-of-frame.
6. Step 5 is repeated for each ISO packet.

Peripheral Mode Suspend

When no activity has occurred on the USB for 3 ms, the USB controller enters suspend mode. If the suspend interrupt (USB_IRQ.SUSPEND) is enabled, an interrupt is generated at this time.

When resume signaling is detected, the USB controller exits suspend mode. If the USB_IRQ.RESUME interrupt is enabled, an interrupt is generated. The processor core can also force the USB controller to exit suspend mode by setting the USB_POWER.RESUME bit. This initiates a remote wakeup. When this bit is set, the USB controller exits suspend mode and drives resume signaling onto the bus. The processor core should clear this bit after 10 ms (a maximum of 15 ms) to end resume signaling.

NOTE: The USB_IRQ.RESUME interrupt is not generated when suspend mode is exited by the processor core, nor is this interrupt generated when the software initiates remote wakeup.

Start-of-frame (SOF) Packets

When the USB controller is operating in peripheral mode, it should receive a start-of-frame packet from the host every millisecond when in full-speed mode, or every 125 microseconds when in high-speed mode.

When the SOF packet is received, the 11-bit frame number contained in the packet is written into the USB_FRAME register and an output pulse, lasting one USB clock bit period, is generated. A start-of-frame interrupt is also generated (if enabled by the USB_IRQ.SOF bit).

After the USB controller has started to receive SOF packets, the controller expects one every millisecond (or 125 ms when in high-speed mode). If no SOF packet is received after 1.00358 ms (or 125.125 ms), it is assumed that the packet is lost. A start-of-frame pulse (together with a USB_IRQ.SOF interrupt) is still generated even though the USB_FRAME register is not updated. The USB controller continues to generate a

SOF pulse every millisecond (or 125 ms) and re-synchronizes these pulses to the received SOF packets when these packets are successfully received again.

Soft Connect/Soft Disconnect

In peripheral mode, the USB controller can be programmed to switch between normal mode and non-driving mode by setting or clearing the USB_POWER.SOFTCONN bit. When USB_POWER.SOFTCONN=1, the USB controller is placed in its normal mode and the D+/D– lines of the USB bus are enabled. When the USB_POWER.SOFTCONN=0, the PHY is put into non-driving mode and D+ and D– are three-stated. The USB controller appears to have been disconnected from the USB bus.

After system reset, USB_POWER.SOFTCONN=0. From that point, the USB controller appears disconnected until the software has set USB_POWER.SOFTCONN =1. The application software can then choose when to set the PHY to its normal mode. Systems with a lengthy initialization procedure may use this to ensure that initialization is complete and the system is ready to perform enumeration before connecting to the USB. Once the USB_POWER.SOFTCONN bit has been set to 1, the software can also simulate a disconnect by clearing this bit to 0.

Error Handling As a Peripheral

A control transfer may be aborted due to a protocol error on the USB, the host prematurely ending the transfer, or if the function controller software wishes to abort the transfer (for example, because it cannot process the command).

The USB controller automatically detects protocol errors and sends a stall packet to the host under the following conditions.

1. The host sends more data during the OUT data phase of a write request than was specified in the command. This condition is detected when the host sends an OUT token after the USB_EP0_CSRn_P.DATAEND bit is set.
2. The host requests more data during the IN data phase of a read request than was specified in the command. This condition is detected when the host sends an IN token after the USB_EP0_CSRn_P.DATAEND bit is set.
3. The host sends more than *MaxPktSize* data bytes in an OUT data packet.
4. The host sends a non-zero length DATA1 packet during the status phase of a read request.

When the USB controller has sent the stall packet, it sets the USB_EP0_CSRn_P.SENTSTALL bit and generates an interrupt. When the software receives an Endpoint 0 interrupt with the USB_EP0_CSRn_P.SENTSTALL bit set, it should abort the current transfer, clear the USB_EP0_CSRn_P.SENTSTALL bit, and return to the IDLE state.

If the host prematurely ends a transfer by entering the status phase before all the data for the request is transferred, or by sending a new SETUP packet before completing the current transfer, then the USB_EP0_CSRn_P.SETUPEND bit is set and an Endpoint 0 interrupt generated. When the software receives an Endpoint 0 interrupt with the USB_EP0_CSRn_P.SETUPEND bit set, it should abort the current transfer, set

the USB_EP0_CSRn_P.SSETUPEND bit, and return to the IDLE state. If the USB_EP0_CSRn_P.RXPKTRDY bit is set, this indicates that the host has sent another SETUP packet and the software should then process this command.

If the software wants to abort the current transfer, because it cannot process the command or has some other internal error, then it should set the USB_EP0_CSRn_P.SENTSTALL bit. The USB controller then sends a stall packet to the host, set the USB_EP0_CSRn_P.SENTSTALL bit and generate an Endpoint 0 interrupt.

Stalls Issued to Control Transfers

In peripheral mode, the USB controller automatically issues a stall handshake to a control transfer under the following conditions:

1. The host sends more data during an OUT data phase of a control transfer than was specified in the device request during the SETUP phase. This condition is detected by the USB controller when the host sends an OUT token (instead of an IN token) after the processor core has unloaded the last OUT packet and set the USB_EP0_CSRn_P.DATAEND bit.
2. The host requests more data during an IN data phase of a control transfer than was specified in the device request during the SETUP phase. This condition is detected by the USB controller when the host sends an IN token (instead of an OUT token) after the processor core has cleared USB_EPn_TXCSR_P.TXPKTRDY and set USB_EP0_CSRn_P.DATAEND in response to the ACK issued by the host to what should have been the last packet.
3. The host sends more than *MaxPktSize* data with an OUT data token.
4. The host sends the wrong PID (packet identifier) for the OUT status phase of a control transfer.
5. The host sends more than a zero length data packet for the OUT status phase.

Zero Length OUT Data Packets in Control Transfers

A zero-length OUT data packet is used to indicate the end of a control transfer. In normal operation, such packets should only be received after the entire length of the device request is transferred (for example, after the processor core has set the USB_EP0_CSRn_P.DATAEND bit). If the host sends a zero-length OUT data packet before the entire length of device request is transferred, this packet signals the premature end of the transfer. In this case, the USB controller automatically flushes any IN token loaded by processor core ready for the data phase from the FIFO and sets the USB_EP0_CSRn_P.SETUPEND bit.

Host Mode

USB OTG interface operations in host mode differ from peripheral mode in a number of ways. The following sections describe host mode operations.

Transaction Scheduling

When operating as a host, the USB controller maintains a frame counter. If the target function is a full-speed device, the USB controller automatically sends an SOF packet at the start of each frame or micro-frame. If the target function is a low-speed device, a K state is transmitted on the bus to act as a *keep-alive* to stop the low-speed device from going into suspend mode.

After the SOF packet is transmitted, the USB controller cycles through all the endpoints looking for active transactions. An active transaction is defined as an RX endpoint for which the `USB_EPn_RXCSR_H.REQPKT` bit is set or a TX endpoint for which the `USB_EPn_TXCSR_H.TXPKTRDY` bit is set.

An active isochronous or interrupt transaction will only start if it is found on the first transaction scheduler cycle of a frame and if the interval counter for that endpoint has counted down to zero. This ensures that only one interrupt or isochronous transaction occurs per endpoint per *n* frames/micro frames (or up to three if high-bandwidth support is selected) where *n* is the interval set in the `USB_EPn_TXINTERVAL` or `USB_EPn_RXINTERVAL` register for that endpoint.

An active bulk transaction is started immediately, provided there is sufficient time left in the frame to complete the transaction before the next SOF packet is due. If the transaction needs to be retried (for example, because a NAK was received or the target function did not respond) then the transaction is not retried until the transaction scheduler has checked all the other endpoints for active transactions first. This check ensures that an endpoint that is sending a lot of NAKs does not block other transactions on the bus. The USB controller lets you specify a limit (`USB_EPn_TXINTERVAL` or `USB_EPn_RXINTERVAL` registers) to the length of time in which NAKs may be received from a particular target before the endpoint is timed out.

Endpoint Setup and Data Transfer

When the `HOST_MODE` bit is set to 1, the USB controller operates as a host for point-to-point communications with another USB device or, when attached to a hub, for communication with a whole range of devices in a multi-point set-up. High-speed, full-speed and low-speed USB functions are supported, both for point-to-point communication and for operation through a hub. (Where necessary, the core automatically carries out the necessary transaction translation needed to allow a low-speed or full-speed device to be used with a USB 2.0 hub.)

Control, Bulk, Isochronous or Interrupt transactions are supported.

Transfers between the subsystem and endpoint FIFOs in host mode are similar to peripheral mode. With this in mind, see many of the descriptions of processor core to FIFO data transfer in *Peripheral Mode*.

Control Transaction as a Host

Host control transactions are conducted through Endpoint 0. The software is required to handle all the Standard Device Requests that may be sent or received through Endpoint 0 (as described in *Universal Serial Bus Specification*, Revision 2.0, Chapter 9).

For a USB peripheral, there are three categories of standard device requests:

- **Zero data requests.** Comprise a SETUP command followed by an IN status phase. All the information is included in the command.
- **Write requests.** Comprised of a SETUP command, followed by an OUT data phase followed by an IN status phase. The command is followed by additional data.
- **Read requests** comprise a SETUP command, followed by an IN data phase followed by an OUT status phase. The device is required to send data back to the host.

A timeout may be set to limit the length of time during which the USB controller w retries a transaction that is continually NAKed by the target. This limit can be between 2 and 2^{15} frames/micro frames and is set through the USB_EP0_NAKLIMITn register.

The following sections look at the steps in different phases of a control transaction to describe the actions of the core in issuing standard device requests.

Setup Phase as a Host

The processor core driving the host device performs the following actions for the SETUP phase of a control transaction.

1. Load the eight bytes of the required device request command into the Endpoint 0 FIFO.
2. Set the USB_EP0_CSRn_H.SETUPPKT bit and USB_EP0_CSRn_H.TXPKTRDY bit. These bits must be set together.

   The USB controller then sends a SETUP token followed by the 8-byte command to Endpoint 0 of the addressed device, retrying as necessary.
3. At the end of the attempt to send the data, the USB controller generates an Endpoint 0 interrupt (for example, set USB_INTRTXE.EP0). The processor core should then read the USB_EP0_CSRn_H register to establish whether the USB_EP0_CSRn_H.RXSTALL, USB_EP0_CSRn_H.TOERR or the USB_EP0_CSRn_H.NAKTO bits are set.

   If USB_EP0_CSRn_H.RXSTALL=1, the target did not accept the command (for example, because it is not supported by the target device) and so has issued a stall response.

   If USB_EP0_CSRn_H.TOERR=1, the USB controller has tried to send the SETUP packet and the following data packet three times without getting a response.

   If USB_EP0_CSRn_H.NAKTO=1, the USB controller has received a NAK response to each attempt to send the SETUP packet, for longer than the time set in the USB_EP0_NAKLIMITn register. The USB controller can then be directed either to continue trying this transaction (until it times out again) by clearing the USB_EP0_CSRn_H.NAKTO bit or to abort the transaction by flushing the FIFO before clearing the USB_EP0_CSRn_H.NAKTO bit.

4. If none of USB_EP0_CSRn_H.RXSTALL, USB_EP0_CSRn_H.TOERR or USB_EP0_CSRn_H.NAKTO bits are set, the SETUP phase is correctly acknowledged and the processor core should proceed to the following IN data phase, OUT data phase or IN status phase specified for the particular standard device request.

IN Data Phase as a Host

The processor core driving the host device performs the following actions for the IN data phase of a control transaction.

1. Set the USB_EP0_CSRn_H.REQPKT bit.
2. Wait while the USB controller sends the IN token and then receives the required data back.
3. When the USB controller generates the Endpoint 0 interrupt (for example, by setting the USB_INTRTXE.EP0 bit), read the USB_EP0_CSRn_H register to establish whether the USB_EP0_CSRn_H.RXSTALL bit, the USB_EP0_CSRn_H.TOERR bit, the USB_EP0_CSRn_H.NAKTO bit or the USB_EP0_CSRn_H.RXPKTRDY bit is set.

   If USB_EP0_CSRn_H.RXSTALL=1, the target has issued a stall response.

   If USB_EP0_CSRn_H.TOERR=1, the USB controller has tried to send the required IN token three times without getting a response.

   If USB_EP0_CSRn_H.NAKTO=1, the USB controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the USB_EP0_NAKLIMITn register. The USB controller can then be directed either to continue trying this transaction (until it times out again) by clearing the USB_EP0_CSRn_H.NAKTO bit; or to abort the transaction by clearing USB_EP0_CSRn_H.REQPKT before clearing the USB_EP0_CSRn_H.NAKTO bit.
4. If the USB_EP0_CSRn_H.RXPKTRDY bit is set, the processor core should read the data from the Endpoint 0 FIFO, then clear USB_EP0_CSRn_H.RXPKTRDY.
5. If further data is expected, the processor core should repeat the previous steps.

   When all the data is successfully received, the processor core should proceed to the OUT status phase of the control transaction.

OUT Data as a Host (Control)

The processor core driving the host device performs the following actions for the OUT data phase of a control transaction.

1. Load the data to be sent into the Endpoint 0 FIFO
2. Set the USB_EP0_CSRn_H.TXPKTRDY bit.

   The USB controller sends an OUT token followed by the data from the FIFO to Endpoint 0 of the addressed device, retrying as necessary.

3. At the end of the attempt to send the data, the USB controller generates an Endpoint 0 interrupt (for example by setting the USB_INTRTX.EP0 bit). The processor core should then read the USB_EP0_CSRn_H to establish whether the USB_EP0_CSRn_H.RXSTALL bit, the USB_EP0_CSRn_H.TOERR bit, or the USB_EP0_CSRn_H.NAKTO bit is set.

   If USB_EP0_CSRn_H.RXSTALL=1, the target has issued a stall response.

   If USB_EP0_CSRn_H.TOERR=1 the USB controller has tried to send the OUT token and the following data packet three times without getting a response.

   If USB_EP0_CSRn_H.NAKTO=1, the USB controller has received a NAK response to each attempt to send the OUT token, for longer than the time set in the USB_EP0_NAKLIMITn register. The USB controller can then be directed either to continue trying this transaction (until it times out again) by clearing the USB_EP0_CSRn_H.NAKTO bit; or to abort the transaction by flushing the FIFO before clearing the USB_EP0_CSRn_H.NAKTO bit.

   If none of the USB_EP0_CSRn_H.RXSTALL, USB_EP0_CSRn_H.TOERR, or USB_EP0_CSRn_H.NAKTO bits are set, the OUT data is correctly acknowledged.

4. If further data needs to be sent, the processor core should repeat the previous steps.

   When all the data is successfully sent, the processor core should proceed to the IN status phase of the control transaction.

IN Status Phase as a Host (Following SETUP Phase or OUT Data Phase)

The processor core driving the host device performs the following actions for the IN status phase of a control transaction.

1. Set the USB_EP0_CSRn_H.STATUSPKT and USB_EP0_CSRn_H.REQPKT bits. These bits must be set together.

2. Wait while the USB controller both sends an IN token and receives a response from the USB peripheral.

3. When the USB controller generates the Endpoint 0 interrupt (for example, sets the USB_INTRTX.EP0 bit), read the USB_EP0_CSRn_H register to establish whether the USB_EP0_CSRn_H.RXSTALL, USB_EP0_CSRn_H.TOERR, USB_EP0_CSRn_H.NAKTO, or the USB_EP0_CSRn_H.RXPKTRDY bits are set.

   If USB_EP0_CSRn_H.RXSTALL=1 the target could not complete the command and so has issued a stall response.

   If USB_EP0_CSRn_H.TOERR=1the USB controller has tried to send the required IN token three times without getting a response.

   If USB_EP0_CSRn_H.NAKTO=1 the USB controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the USB_EP0_NAKLIMITn register. The USB controller can then be directed either to continue trying this transaction (until it times out again) by clearing the USB_EP0_CSRn_H.NAKTO bit or to abort the transaction by clearing USB_EP0_CSRn_H.REQPKT and USB_EP0_CSRn_H.STATUSPKT before clearing the USB_EP0_CSRn_H.NAKTO bit.

4. If the USB_EP0_CSRn_H.RXPKTRDY bit is set, the processor core should clear it.

OUT Status Phase as a Host (Following IN Data Phase)

The processor core driving the host device performs the following actions for the OUT status phase of a control transaction.

1. Set USB_EP0_CSRn_H.STATUSPKT and USB_EP0_CSRn_H.TXPKTRDY bits. These bits must be set together.
2. Wait while the USB controller both sends the OUT token and a zero-length DATA1 packet.
3. At the end of the attempt to send the data, the USB controller generates an Endpoint 0 interrupt. The processor core should then read the USB_EP0_CSRn_H register to discover if the USB_EP0_CSRn_H.RXSTALL, USB_EP0_CSRn_H.TOERR, or USB_EP0_CSRn_H.NAKTO bits are set.

   If USB_EP0_CSRn_H.RXSTALL=1 the target could not complete the command and so has issued a stall response.

   If USB_EP0_CSRn_H.TOERR=1 the USB controller has tried to send the STATUS packet and the following data packet three times without getting a response.

   If USB_EP0_CSRn_H.NAKTO=1 the USB controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the USB_EP0_NAKLIMITn register. The USB controller can then be directed either to continue trying this transaction (until it times out again) by clearing the USB_EP0_CSRn_H.NAKTO bit or to abort the transaction by flushing the FIFO before clearing the USB_EP0_CSRn_H.NAKTO bit.
4. If none of the USB_EP0_CSRn_H.RXSTALL, USB_EP0_CSRn_H.TOERR, or USB_EP0_CSRn_H.NAKTO bits are set, the status phase is correctly acknowledged.

Host IN Transactions

When the USB controller operates as a host, IN transactions are handled like OUT transactions are handled when the USB controller is operating as a peripheral. But the transaction must first be initiated by setting the USB_EPn_RXCSR_H.REQPKT bit. This bit indicates to the transaction scheduler that there is an active transaction on this endpoint. The transaction scheduler then sends an IN token to the target function.

When the packet is received and placed in the RX FIFO, the USB_EPn_RXCSR_H.RXPKTRDY bit is set, and the appropriate RX endpoint interrupt is generated (if enabled) to signal that a packet can now be unloaded from the FIFO. When the packet is unloaded, USB_EPn_RXCSR_H.RXPKTRDY is cleared. The USB_EPn_RXCSR_H.AUTOCLR bit can be used to automatically clear the USB_EPn_RXCSR_H.RXPKTRDY bit when a maximum sized packet is unloaded from the FIFO. There is also an USB_EPn_RXCSR_H.AUTOREQ bit that automatically sets the USB_EPn_RXCSR_H.REQPKT bit when the USB_EPn_RXCSR_H.RXPKTRDY bit is cleared. The USB_EPn_RXCSR_H.AUTOCLR and USB_EPn_RXCSR_H.AUTOREQ bits can be used with an external DMA controller to perform complete bulk transfers without processor core intervention.

If the target function responds to a bulk or interrupt IN token with a NAK, the USB controller keeps retrying the transaction until the NAK limit set in the USB_EP0_NAKLIMITn register) is reached. If the target function responds with a stall, the USB controller does not retry the transaction, but interrupts the processor core by setting the USB_EPn_RXCSR_H.RXSTALL bit. If the target function does not respond to the IN token within the required time (or there was a CRC or bit-stuff error in the packet), the USB controller retries the transaction. If after three attempts the target function still has not responded, the USB controller clears the USB_EPn_RXCSR_H.REQPKT bit and interrupts the processor core with the DATAERROR_R bit in USB_RXCSR set.

Host OUT Transactions

When the USB controller operates as a host, OUT transactions are handled in a similar manner to the way IN transactions are handled when the USB controller operates as a peripheral.

The USB_EPn_TXCSR_H.TXPKTRDY bit needs to be set as each packet is loaded into the TX FIFO and the USB_EPn_TXCSR_H.AUTOSET bit can be used to cause the USB_EPn_TXCSR_H.TXPKTRDY bit to be automatically set when a maximum sized packet is loaded into the FIFO. The USB_EPn_TXCSR_H.AUTOSET bit can be used with an external DMA controller to perform complete bulk transfers without processor core intervention.

If the target function responds to the OUT token with a NAK, the USB controller keeps retrying the transaction until the NAK limit set in the USB_EP0_NAKLIMITn register is reached. If the target function responds with a stall, the USB controller does not retry the transaction, but interrupts the processor core by setting the USB_EPn_TXCSR_H.RXSTALL bit. If the target function does not respond to the OUT token within the required time (or there was a CRC or bit-stuff error in the packet), the USB controller retries the transaction. If after three attempts the target function still has not responded, the USB controller flushes the FIFO and interrupts the processor core by setting the USB_EPn_TXCSR_H.TXTOERR bit.

Multi-Point Support

The following sections describe the controller's multi-point support.

- *Allocating Devices to Endpoints*
- *Multi-Point Operation*
- *Multi-Point Bandwidth Considerations*

Allocating Devices to Endpoints

The separate functions of the connected devices are allocated to the endpoints within the USB controller through a group of three registers, which are associated with each implemented Rx or Tx endpoint (including Endpoint 0).

The registers are USB_MPn_TXFUNCADDR/ USB_MPn_RXFUNCADDR, USB_MPn_TXHUBADDR/ USB_MPn_RXHUBADDR and USB_MPn_TXHUBPORT/ USB_MPn_RXHUBPORT. Note that the location of these registers depends on which of the endpoints is being addressed.

The information that needs to be recorded in the transmit and receive function address registers is the address of the target function that is to be accessed through the selected endpoint. This information needs to be recorded separately for each Tx and Rx endpoint that is used. In particular, both USB_MPn_TXFUNCADDR and USB_MPn_RXFUNCADDR need to be set for Endpoint 0.

The transmit and receive hub address and hub port registers are used when a full- or low-speed device is connected to the USB controller via a high-speed USB 2.0 hub, which carries out the required transaction translation between high-speed transmission and low-/full-speed transmission. In this situation, the USB_MPn_TXHUBADDR/ USB_MPn_RXHUBADDR and USB_MPn_TXHUBPORT/ USB_MPn_RXHUBPORT registers need to record the address of the hub that carries out the transaction translation and the port of that hub through which the associated Tx/Rx endpoint needs to access the device.

Note that if Endpoint 0 is connected to a hub, then both the Tx and the Rx versions of these registers need to be set for this endpoint. The hub address registers are also used to record whether the hub offers multiple transaction translators or just a single transaction translator. This has a significant effect on the overall bandwidth that can be achieved.

In addition to recording the address of the target function through these three registers, the endpoint number and operating speed of the target device and the type of transaction that is executed need to be recorded. For a Tx endpoint, this information needs to be set in the USB_EPn_TXTYPE register when the index register is set to select the required endpoint. For an Rx endpoint, this information needs to be set in the USB_EPn_RXTYPE register when the index register is set to select the required endpoint. In both cases, the endpoint number is recorded in bits 3–0, the transaction type is selected through bits 5–4, and the operating speed is selected through bits 7–6.

Only the speed needs to be set for Endpoint 0 because endpoint 0 only has the facilities to handle control transactions and therefore is always associated with a device Endpoint 0. This speed setting is made through bits 7–6 of the Type 0 register, which is located at address 0x1A when the index register is set to 0.

Multi-Point Operation

Once the allocation of functions to endpoints has been made and the operating speed of the target device recorded, most operations in a multi-point set-up are no different from those for the equivalent actions where the core is attached to a single other device.

However, additional steps are required

- When the option of dynamically switching the allocation of functions to endpoints is taken (for example to allow a wider range of devices to be supported)
- When the control packets normally associated with Endpoint 0 are handled through a different endpoint.

If dynamic allocation is used, it is essential for the program to keep track of the current data toggle state associated with the endpoint and with each of the devices that are allocated to that endpoint. Knowledge of this state is necessary to allow the program to select the correct data toggle state when the switch is made between one device and the other. (This action is the programs responsibility because the core cannot determine what data toggle state is expected when a function is being switched in and out of use.)

The data toggle state can be switched from its current state by writing to the appropriate USB_EPn_TXCSR_H or USB_EPn_RXCSR_H register to set the data toggle write enable and data toggle bits that are included in these registers when the core is in host mode.

Data toggle write enable and data toggle bits are also included in the USB_EP0_CSRn_H register. However, control operations carried out through the core's Endpoint 0 should normally always leave the data toggle in the expected state.

Where control packets are handled through an endpoint other than Endpoint 0, programs need to prompt for each setup token to be sent. This involves setting the USB_EPn_TXCSR_H.SETUPPKT bit when the core is operating in host mode, alongside the USB_EPn_TXCSR_H.TXPKTRDY bit. If the USB_EPn_TXCSR_H.SETUPPKT bit is not set, an OUT token is sent.

Overall, the recommendation is to use the controller's Endpoint 0 to handle control packets for all of the devices attached to the controller, and to switch the allocation of this endpoint as appropriate. Sending the correct token is ensured, as is ensuring that the data toggle is correctly set for this endpoint.

Using a different endpoint for this function is possible, as described above, but there are further points to note:

- The control function must be allocated to an Rx/Tx endpoint pair (with the same endpoint number).
- The chosen endpoints must each be associated with FIFOs that can accommodate the packet size associated with EP0 transactions at the chosen operating speed which can be a minimum of 8 bytes for low- or full-speed transactions but 64 bytes for high-speed transactions.

Multi-Point Bandwidth Considerations

The ability of a multi-point system to cope with isochronous transactions, in particular, is determined by the available bandwidth.

Once an endpoint has been set up, all scheduling is handled in hardware. However, as with PC-based EHCI/OHCI/UHCI hosts, before opening a periodic pipe (for use by isochronous or interrupt traffic), software must determine that there is sufficient bandwidth available.

Further, if the periodic pipe is opened to a full-speed device through a high-speed hub, software must confirm that sufficient bandwidth is available both on the local high-speed bus and the full-speed bus generated by the transaction translator in the hub. The bandwidth required for different transactions can be determined using similar algorithms to those used in connection with PC-based hosts (detailed in Section 5.11.3 of the USB 2.0 Specification).

Note that the available bandwidth is greater where the hub used supports multiple transaction translators.

Babble Interrupt

If the bus is still active at the end of a frame, the USB controller assumes that the function it is connected to has malfunctioned, suspends all transactions, and generates a babble interrupt (USB_IRQ.RSTBABBLE). The USB controller does not start a transaction until the bus is inactive for at least the minimum

inter-packet delay. The controller also does not start a transaction unless it can be finished before the end of the frame.

To recover from a babble error condition, the processor must take the following actions inside the interrupt service routine.

1. Turn off VBUS. Wait until the VBUS level indicator reads b#01.
2. Turn on VBUS. Wait until the VBUS level indicator reads b#11.
3. Set the `USB_IRQ.SESSREQ` bit

The VBUS level indicator is the `USB_DEV_CTL.VBUS` bit field

NOTE:

Because VBUS is sourced external to the processor, make sure that the hardware design connects a GPIO or the dedicated `USB_VBUS` signal to the external source so that you can use software to turn VBUS on and off.

NOTE:

VBUS Events

The USB On-The-Go specification defines a series of thresholds to which the devices involved in point-to-point communications are required to respond.

- VBUS Valid (between 4.4 V and 4.75 V)
- Session Valid for A device (between 0.8 V and 2.1 V)
- Session End (between 0.2 V and 0.8 V)

Which thresholds are critical and the processor response depends on whether the device is an A device or a B device and the circumstances of the event. These actions are described below.

Actions as an "A" Device

VBUS >VBUS Valid with session initiated by USB controller. VBUS level indicator = b#11 and session bit is set. When VBUS is greater than VBUS valid, the USB controller selects host mode and waits for a device to be connected. It then generates a connect interrupt. The processor resets and enumerates the connected B device.

VBUS > Session Valid with session initiated by B device. VBUS level indicator = b#10 and session bit is clear. When VBUS is greater than session valid, the USB controller generates a session request interrupt. The processor sets the session bit and the USB controller either stays in Host mode or changes to Peripheral mode, depending upon the state of the pull-up resistor on the B device. For more information, refer to the host negotiation protocol of the OTG specification. The selected mode is indicated by the state of the Host Mode bit.

VBUS below VBUS Valid while the Session bit remains set. VBUS level indicator b#11 and session bit is set. This indicates a problem with the VBUS power level. For example, the battery power may have dropped too low to sustain VBUS valid. Or, the B device may be drawing more current than the A device can provide. In either case, the USB controller will automatically terminate the session and generate a VBUS error interrupt.

To recover from this VBUS error condition, the processor must take the following actions inside the VBUS error interrupt handler.

- Turn off VBUS wait until the `USB_DEV_CTL.VBUS` reads b#01.
- Turn on VBUS wait until the `USB_DEV_CTL.VBUS` reads b#11.
- Set the `USB_DEV_CTL.SESSION` bit

The VBUS level indicator the `USB_DEV_CTL.VBUS` bit field.

NOTE: Because VBUS is sourced external to the processor, make sure that the hardware design connects a GPIO or the dedicated `DrvVBUS` signal to the external source so that software can be used to turn VBUS on and off.

Actions as a "B" Device

VBUS > Session Valid. VBUS level indicator = b#10 and session bit is clear. This indicates activity from the A device. The USB controller sets the session bit and disconnects the pull down resistor on the D+ line.

VBUS < Session Valid. while the session bit remains set VBUS level indicator = b#01 and session bit is set. This indicates that the A device has lost power (or become disconnected). The USB controller clears the session bit and generates a disconnect interrupt. The processor ends the session.

VBUS < Session End. VBUS level indicator = b#00. This is the condition under which a B device can initiate a session request. If the session bit is set, then after 2 ms of SE0 on the bus, the USB controller starts SRP by first pulsing the data line, then pulsing the `USB_VBUS` signal.

Host Mode Reset

If the `USB_POWER.RESET` is set while the USB controller is in host mode, the USB controller generates reset signaling on the bus. The processor core should keep this bit set for 20 ms to ensure correct resetting of the target device. After the processor core has cleared the bit, the USB controller starts its frame counter and transaction scheduler.

Host Mode Suspend

The controller has a suspend mode that allows power savings for the processor. The mode operates as described below.

Entry into Suspend mode. When operating as a host, the USB controller can be prompted to go into Suspend mode by setting the `USB_POWER.SUSPEND` bit. When this bit is set, the USB controller completes the current transaction then stops the transaction scheduler and frame counter. No further transactions

are started and no SOF packets are generated. If the USB_POWER.SUSPEND bit is set, the UTMI+ PHY goes into low-power mode when the USB controller goes into suspend mode and stops the clock.

Sending Resume Signaling. When the application requires the USB controller to leave suspend mode, it needs to clear then set the USB_POWER.RESUME bit, and leave it set for 20 ms. While the USB_POWER.RESUME bit is high, the USB controller generates resume signaling on the bus. After 20 ms, the processor core should clear the USB_POWER.RESUME bit, at which point the frame counter and transaction scheduler are started.

Responding to Remote Wake-up. If resume signaling is detected from the target while the USB controller is in suspend mode, the UTMI+ PHY is brought out of low-power mode and the clock restarts. The USB controller then exits suspend mode and automatically sets the USB_POWER.RESUME bit to take over generating the resume signaling from the target. If the USB_IRQ.RESUMEbit=1, an interrupt is generated.

Suspending and Resuming the Controller

With the introduction of link power management, there are two basic methods for the USB controller to be suspended and resumed. These two methods are demonstrated in the basic LPM transaction diagram shown below.

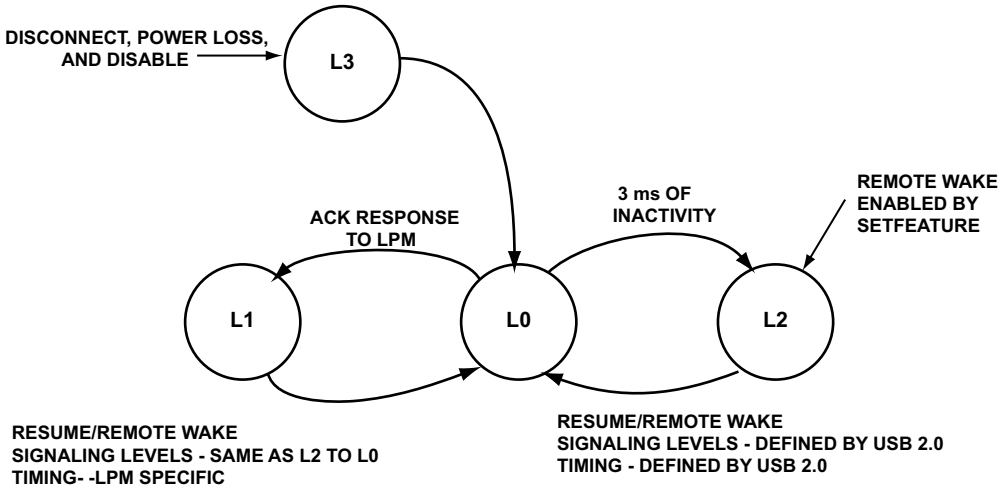


Figure 22-9: Basic LPM Transaction

The procedure in which the USB controller is suspended and resumed depends on whether the core is operating as a device or a host and the method of suspend desired. These options are described in the following sections.

Suspend/Resume by Inactivity on the USB Bus (L0 to L2 State) in Peripheral Mode

The following steps occur in this mode.

1. Entry into suspend mode. When operating as a peripheral, the USB controller monitors activity on the USB and when no activity has occurred for 3 ms, the controller goes into suspend mode. If the USB_

IRQ.SUSPEND interrupt has been enabled, an interrupt is generated at this time. The USB_IRQ.SUSPEND output also goes low (if enabled).

At this point, the *POWERDWN* signal is also asserted to indicate that the application may save power by stopping USB_CLKIn. POWERDWN then remains asserted until either power is removed from the bus (indicating that the device has been disconnected) or resume signaling or reset signaling is detected on the bus.

2. When resume signaling occurs on the bus, the USB_CLKIn must be restarted if necessary. The USB controller then automatically exits suspend mode. If the USB_IRQ.RESUME interrupt is enabled, an interrupt is generated.

3. Initiating a remote wakeup. To initiate a remote wakeup while the controller is in suspend mode, set the USB_POWER.RESUME bit=1. (Note: If USB_CLKIn has been stopped, it will need to be restarted before this write can occur.) The software should leave then this bit set for approximately 10 ms (minimum of 2 ms, a maximum of 15 ms) before resetting it to 0. By this time the hub should have taken over driving Resume signaling on the USB.

NOTE:

The USB_IRQ.RESUME interrupt is not generated when the software initiates a remote wakeup.

NOTE:

Suspend/Resume By Inactivity On The USB Bus (L0 To L2 State) In Host Mode

The following steps occur in this mode.

1. Entry into suspend mode. When operating as a host, the USB controller can be prompted to go into suspend mode by setting the USB_POWER.SUSPEND bit. When this bit is set, the USB controller completes the current transaction then stops the transaction scheduler and frame counter. No further transactions are started and no SOF packets are generated. If the USB_POWER.SUSEN bit is set, the UTMI+ PHY goes into low-power mode when the controller goes into suspend mode and stop USB_CLKIn.

2. Sending resume signaling. When the application requires the controller to leave suspend mode, it clears the USB_POWER.SUSPEND bit, sets the USB_POWER.RESUME bit and leaves it set for 20 ms. While the USB_POWER.RESUME bit is high, the controller generates Resume signaling on the bus. After 20 ms, the processor core should clear the USB_POWER.RESUME bit, at which point the frame counter and transaction scheduler are started.

3. Responding to remote wake-up. If resume signaling is detected from the target while the USB controller is in suspend mode, the UTMI+ PHY is brought out of low-power mode and restart USB_CLKIn. The controller then exits suspend mode and automatically sets the USB_POWER.RESUME bit to 1 to take over generating the resume signaling from the target. If the USB_IRQ.RESUME interrupt is enabled, an interrupt is generated.

Suspend/Resume By an LPM Transaction (L0 To L1 State) In Peripheral Mode

The following steps occur in this mode.

1. Entry into suspend mode. When operating as a peripheral, the controller never initiates an LPM suspend (transition from the L0 state to the L1 state). Rather, the controller only suspends at the request of the host. However, for this to occur, the LPM feature must be enabled by setting up the USB_LPM_CTL register appropriately. The register field USB_LPM_CTL.EN bit is used to enable and support extended and LPM transactions. The USB_LPM_CTL.TX field is used instruct the hardware that it is ready to suspend and to respond to the next LPM transaction with an ACK. In this case, the controller responds to the next LPM transaction with an ACK if all other conditions are met. The response to an LPM transaction by the controller is summarized in the table below.

Table 22-6: Response to LPM Transaction

LPMXMT	LPMCNTRL	Data Pending (Resides in Tx FIFOs)	Response to Next LPM Transaction
1'b0, 1'b0 1'b1 1'b1	2'b00, 2'b10 2'b00 2'b10	Don't Care	Timeout
1'b0, 1'b1	2'b01	Don't Care	STALL
1'b0	2'b11	Don't Care	NYET
1'b1	2'b11	Yes	NYET
1'b1	2'b11	No	ACK

For all cases shown above in which the controller responds (no timeout occurs), an LPM interrupt is generated in the USB_LPM_IRQ register. Note that the controller responds with an ACK only if there is no data pending in any of the TX Endpoint FIFOs. If there is data pending, the USB controller responds with a NYET.

Once an LPM transaction is successfully received three events occur:

a. The USB_LPM_ATTR register is updated with values received in the LPM transaction just received. See the "Register Descriptions" section of this chapter for complete information on this register.

b. The controller suspend 9 µs after transmitting the ACK. Resume signaling can be driven by the host or the controller 50 µs after this event. During this 9 µs interval, the host may continue to transmit the LPM transaction. The controller responds with an ACK in this case regardless of the USB_LPM_CTL.TX bit value.

c. An interrupt is generated informing software of the response (an ACK in this case). An ACK response is the indication to software that the controller has suspended.

   Since the primary purpose of LPM is to save power, the software reads the USB_LPM_ATTR register to determine the attributes of the suspend. Software must make a determination based

on these attributes whether additional power savings in the system can be found. In making this determination note that if the host initiates the resume signaling, the controller is required to respond to packet transmissions within the time specified by USB_LPM_ATTR.HIRD + 10 µs.

2. When resume signaling occurs on the bus. When the host resumes the bus, it drives resume signaling for a minimum time specified by the host initiated resume duration bit field (USB_LPM_ATTR.HIRD). The controller must be able to respond to traffic within the time HIRD + 10 µs. The controller transitions to a normal operating state automatically and a resume interrupt is generated in the USB_LPM_IRQ register.

   However for this to occur, the inputs CLK and XCLK must be available. To facilitate the resume timing requirement, a negative ACK (NAK) is provided using the USB_LPM_CTL.NAK bit. If this bit is set to 1'b1, all endpoints respond to any transaction (other then an LPM) with a NAK. This bit only takes effect after the controller has suspended LPM. Typically, this bit is asserted when the USB_LPM_CTL.TX field is also asserted. Using this feature may simplify the resume timing requirement because only XCLK is needed for the controller to respond (with a NAK) to traffic. Software can continue to restore the system to normal operation while the controller responds to all transactions with a NAK. After the system has been completely restored, software can then clear the USB_LPM_CTL.NAK bit.

3. Initiating remote wakeup. To initiate a remote wakeup while the controller is in suspend mode, it write a 1'b1 to theUSB_LPM_CTL.RESUME bit. This bit is self clearing. Writing a 1'b1 drives resume signaling on the bus for 50 µs. The host responds by driving resume for 60 µs to 990 µs. 10 µs after the host stops driving resume, the controller transitions to its normal operational state and is ready for packet transmission. A resume interrupt is generated in the USB_LPM_IRQ register.

Suspend/Resume by an LPM Transaction (L0 to L1 State) in Host Mode

The following steps occur in this mode.

1. Entry into suspend mode. When operating as a host, the controller initiates an LPM suspend (transition from the L0 state to the L1 state) by initiating an LPM transaction as follows.

   a. Software sets up the desired attributes of the suspend in the USB_LPM_ATTR register. Enabling remote wakeup and a large HIRD gives the peripheral more opportunity to conserve power.

   b. All LPM interrupts should be enabled in the USB_LPM_IEN register.

   c. Software should initiate the transaction by writing a 0x01 to the USB_LPM_CTL register.

   d. An interrupt is generated to inform software of the response to the LPM transaction. If an ACK was received, then the controller suspends automatically within 8 µs. This is the indication that the controller has suspended.

   If the response from the device has a bit stuff error or a PID error, then an USB_LPM_IRQ.LPMERR interrupt is generated. The hardware immediately attempts the LPM transaction two more times. The device does not suspend for 8 µs after the initial LPM so it can respond to either of these subsequent LPM transactions. If a LPM timeout has occurred three times, the USB_LPM_IRQ.LPMNC and the USB_LPM_IRQ.LPMERR interrupts are set. At this time, software is unaware of the device state and must deduce it by other means.

2. Sending resume signaling. Resume signaling should be generated by software as follows.
    a. All LPM interrupts should be enabled in the USB_LPM_IEN register.
    b. Software should write the USB_LPM_CTL.RESUME bit which is self-clearing. This causes resume signaling to be generated on the bus for the time that is currently specified in the USB_LPM_ATTR.HIRD bit field. It is assumed by hardware that this value was used in the last LPM transaction that caused the suspend.
    c. After HIRD + 10 µs, the controller transitions to its normal operational state and is ready for packet transmission and a USB_LPM_IRQ.LPMRES interrupt is generated.

NOTE: Prior to resuming, software must ensure that the system is completely restored from a low power state and that the inputs CLK and XCLK are available.

3. Responding to remote wake-up. If the remote wakeup feature is enabled in the LPM transaction that caused the suspend, then the device may drive resume signaling on the bus. When this occurs, the device drives resume signaling BUS for 50 µs. The controller will immediately begin driving resume signaling on the BUS and will do so for 60 µs. 10 µs after completion of the resume signaling, the controller transitions to its normal operating state and is ready for packet transmission. At this time, the USB_LPM_IRQ.LPMRES interrupt is generated.

USB Event Control

The following sections provide information on the use of interrupts, reset and the reporting of errors and interface status.

Interrupt Signals

The two interrupts generated from the USB controller are shown in *ADSP-BF60x USB Interrupt List*.

Interrupts can be generated from control endpoint zero under the following conditions

- When a control transaction ends before the end of the data is transferred.
- When a data packet is sent or received from the endpoint 0 FIFOs.

Interrupts can be generated from transmit endpoints (USB_INTRTX) under the following conditions:

- packet sent from the TX FIFO (host and peripheral mode)
- after three attempts at transmitting a packet with no valid handshake packet received (host mode)

Interrupts can be generated from receive endpoints (USB_INTRRX) under the following conditions:

- packet received into the RX FIFO (host and peripheral mode)
- when a stall handshake is received (host mode)

- After three attempts at receiving a packet and no data packet is received (host mode).

Interrupts can be generated from the USB status (`USB_IRQ`) under the following conditions:

- When VBUS drops below the VBUS valid threshold during a session (A device only).
- When SRP signaling is detected (A device only).
- When device disconnect is detected (host mode).
- When a session ends (peripheral mode).
- Device connection detected (host mode).
- Start-of-frame (SOF)
- Reset signaling detected on USB (peripheral mode).
- Babble detected (host mode).
- In suspend mode when resume signaling detected on USB.
- When suspend signaling is detected (peripheral mode).

Interrupts are generated for the following VBUS control requests by the USB controller:

- drive VBUS greater than 4.4 V (default A device)
- stop driving VBUS
- start charging VBUS (peripheral mode)
- stop charging VBUS
- start discharging VBUS (peripheral mode)
- stop discharging VBUS

Interrupt Handling

When the processor core is interrupted with a USB interrupt, it needs to read the interrupt status register to determine which endpoint(s) have caused the interrupt and jump to the appropriate routine. If multiple endpoints have caused the interrupt, Endpoint 0 should be serviced first, followed by the other endpoints. A flowchart for the USB interrupt service routine is shown in the **USB Interrupt Service Routine** figure.

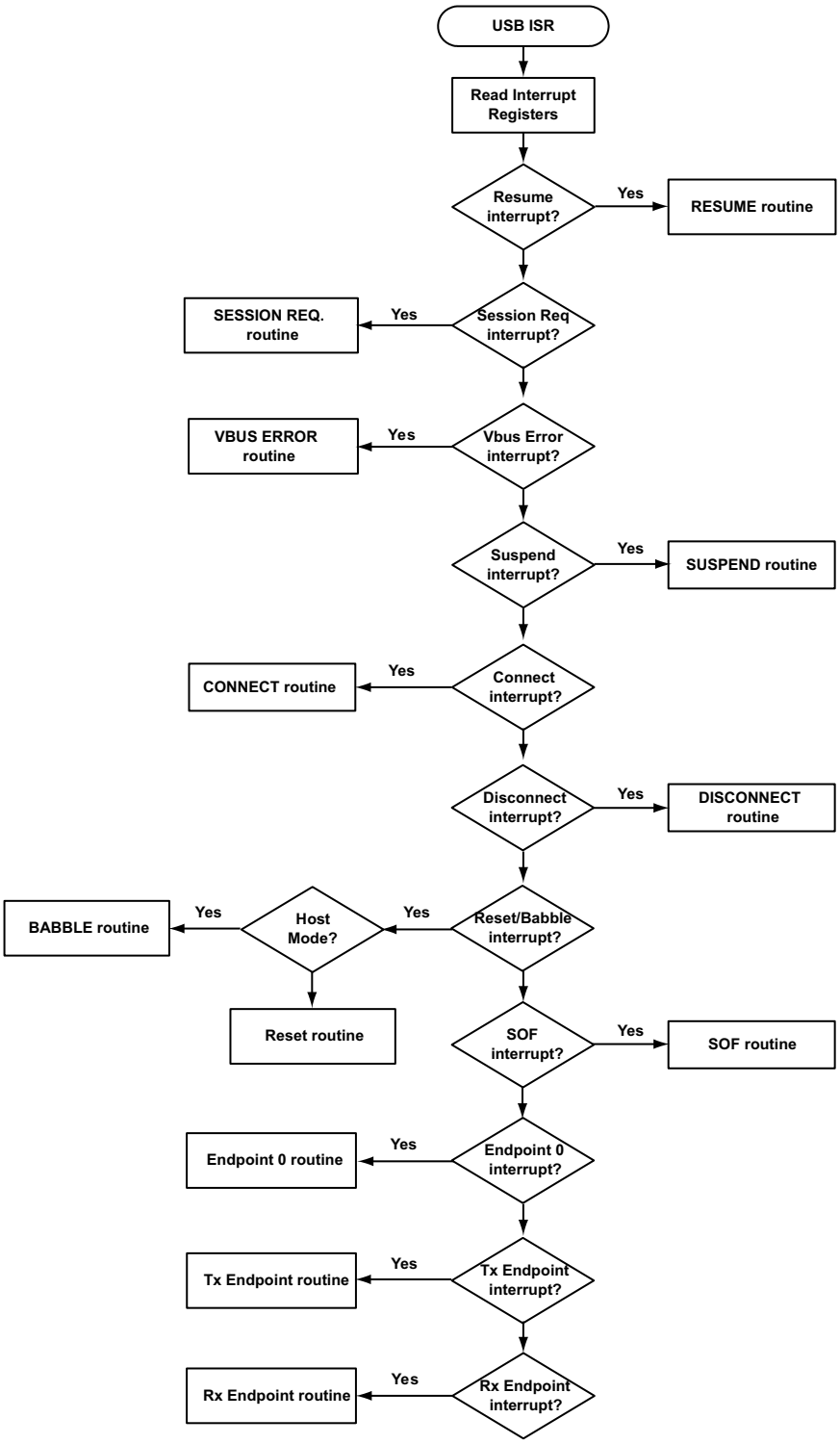


Figure 22-10: USB Interrupt Service Routine

Reset Signals

The USB controller includes an active-high synchronous hardware reset sourced from the processor core. Another source of peripheral reset is through the USB, when USB reset signaling is detected on the I/O lines. As dictated by the USB 2.0 Specification, this state is entered when both the D+ and D- inputs are driven low for a period of 2.5 ms or more (though the reset itself is held for typically greater than 10 ms by the USB host).

Reset in Peripheral Mode

When a USB reset is detected, the USB controller performs the following actions:

- USB_FADDR register set to zero
- USB_INDEX register set to zero
- all endpoint FIFOs flushed
- all control and status registers cleared
- all interrupts enabled
- reset interrupt generated

The USB_IRQ and USB_VBUS_CTL, registers are not affected by the USB controller reset. These registers are only reset (along with those listed above) during a system reset.

If the USB_POWER.HSEN bit was set, the USB controller also tries to negotiate for high-speed operation. Whether high-speed operation is selected is indicated by the USB_POWER.HSMODE bit.

When the application software receives a reset interrupt, it should close any open pipes and wait for bus enumeration to begin.

USB Reset in Host Mode

If the USB_POWER.RESET bit=1 while the USB controller is in host mode, the controller generates reset signaling on the bus. If the USB_POWER.HSEN bit =1, the controller also tries to negotiate for high-speed operation.

The processor core should keep the USB_POWER.RESET bit set for at least 20 ms to ensure correct resetting of the target device. After the processor core has cleared the bit, the USB controller starts its frame counter and transaction scheduler.

High-speed operation is selected by the USB_POWER.HSMODE bit

USB Programming Model

The following sections describe the USB OTG programming model.

Peripheral Mode Flow Charts

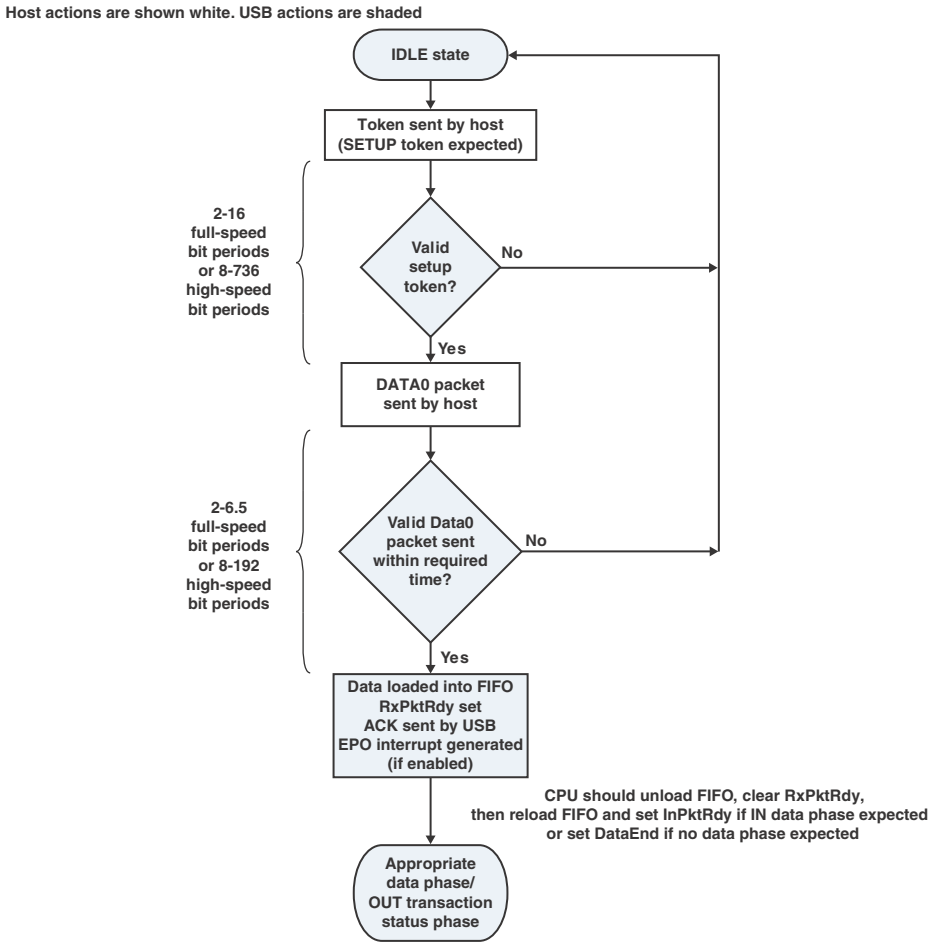


Figure 22-11: USB Control Setup Phase

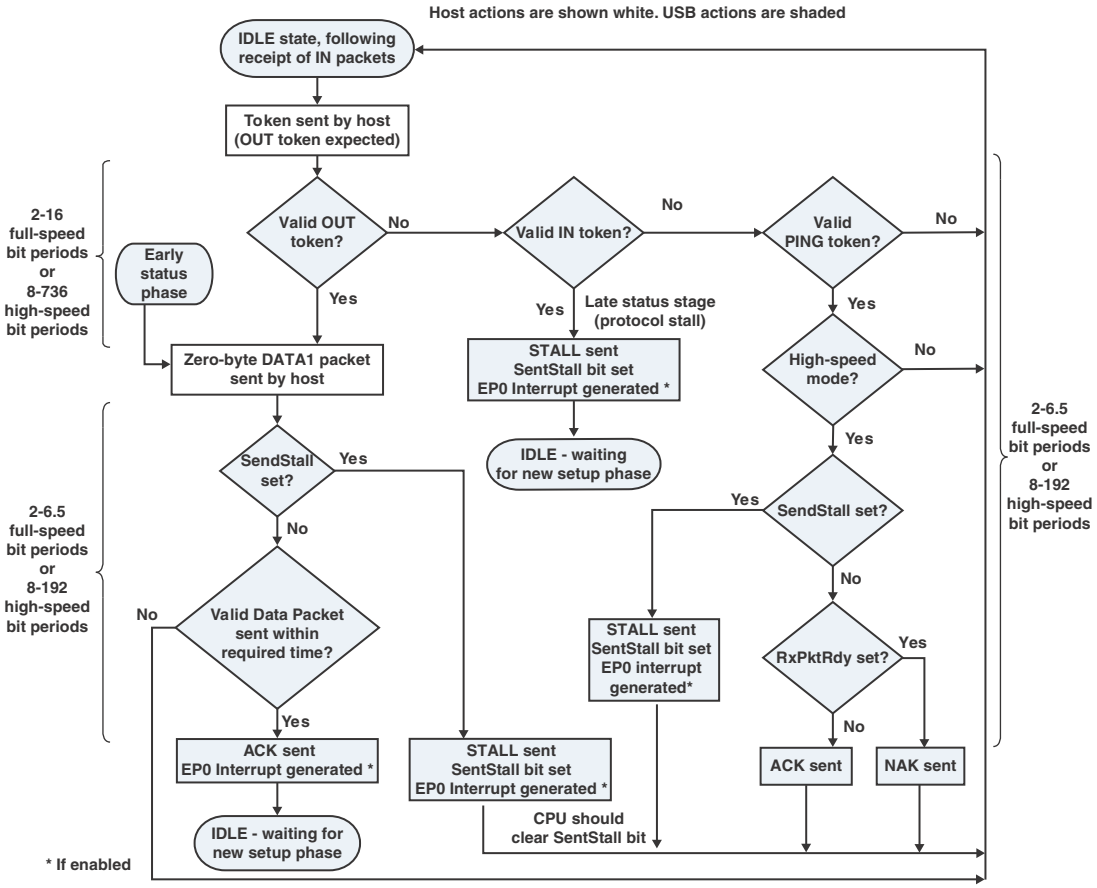


Figure 22-12: Control In Data Phase

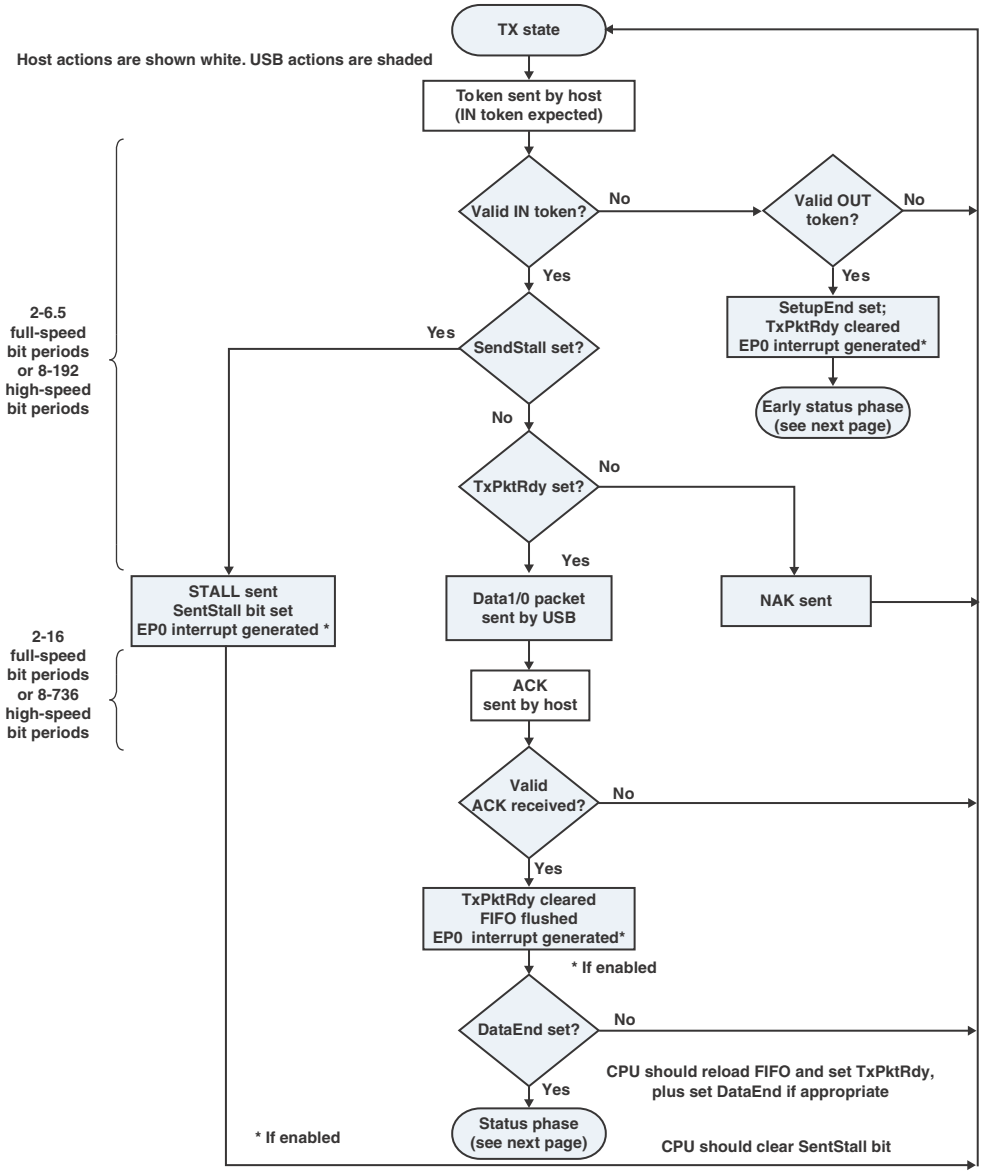


Figure 22-13: Control In Data Status Phase

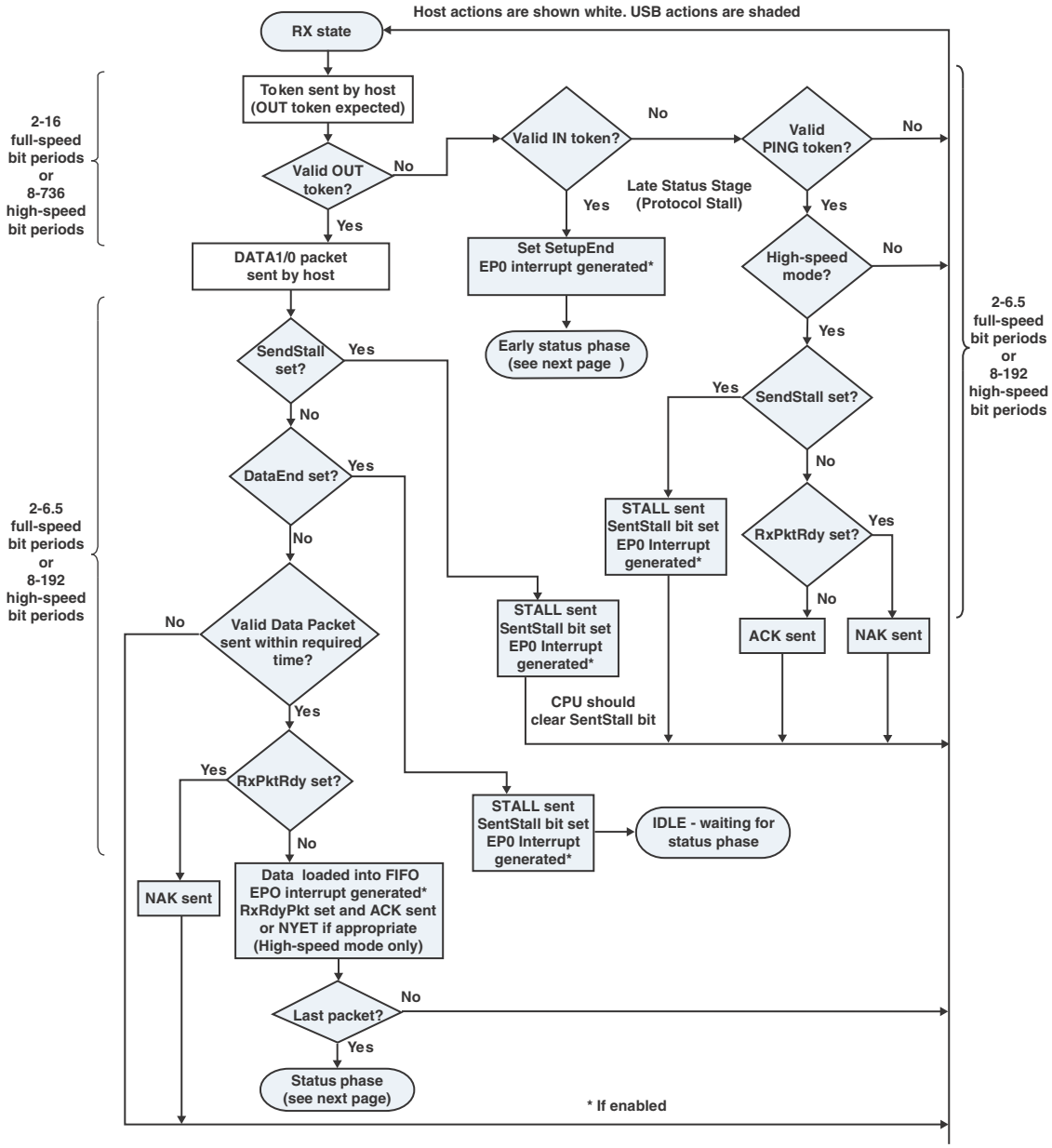


Figure 22-14: Control Out Data Phase

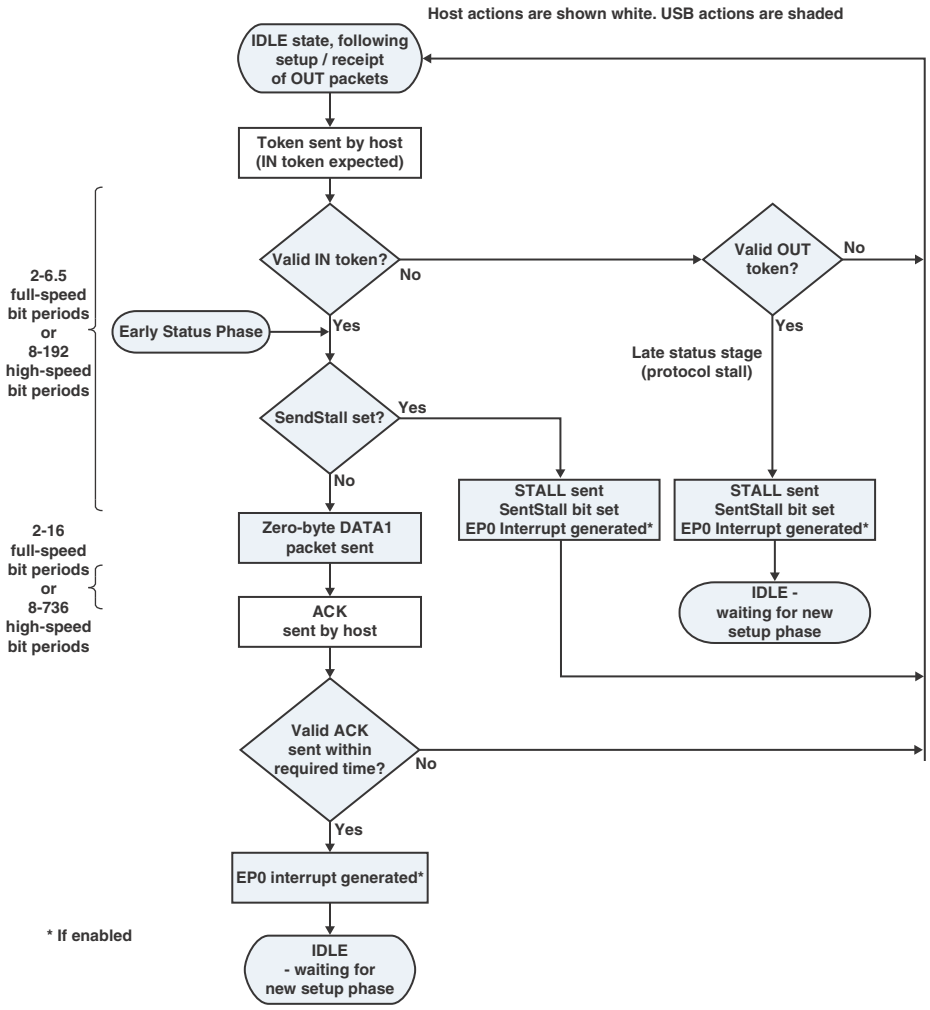


Figure 22-15: Control Out Data Status Phase

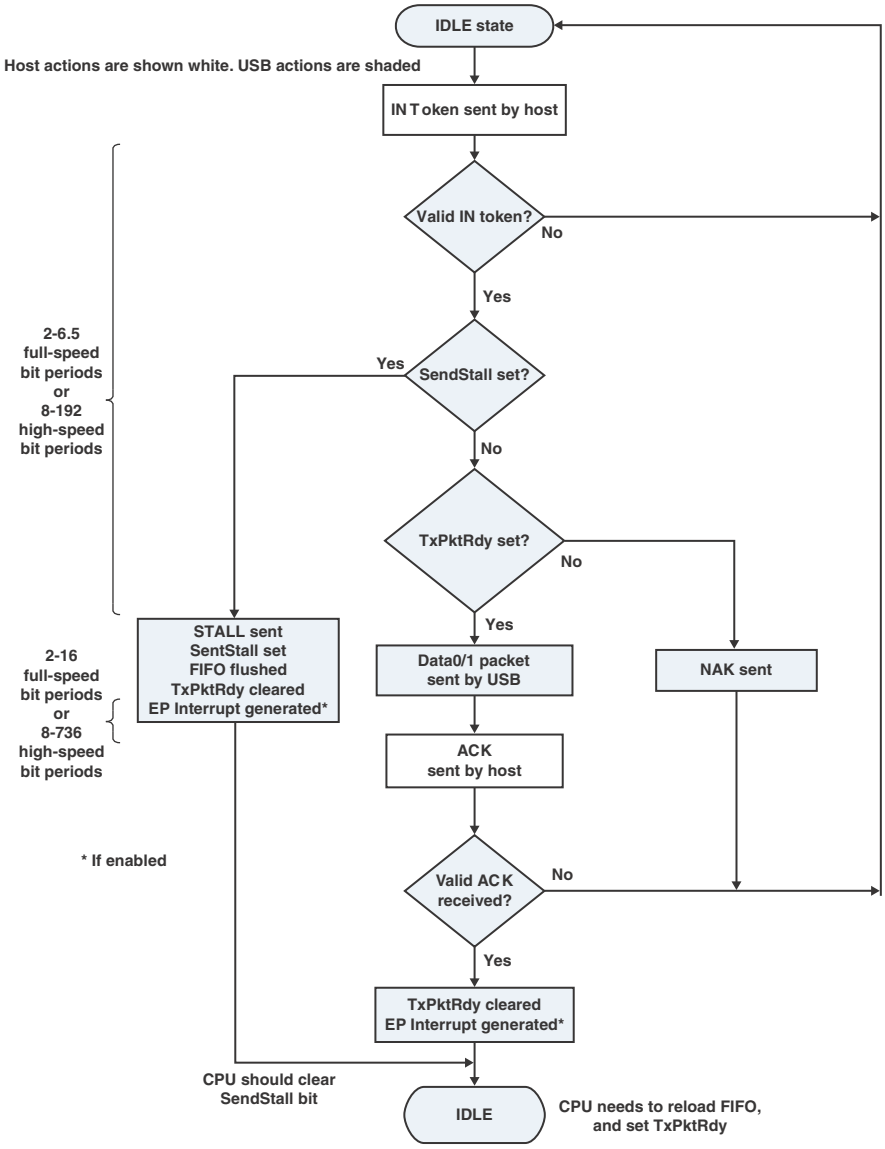


Figure 22-16: Bulk/Low Bandwidth Interrupt In Transaction

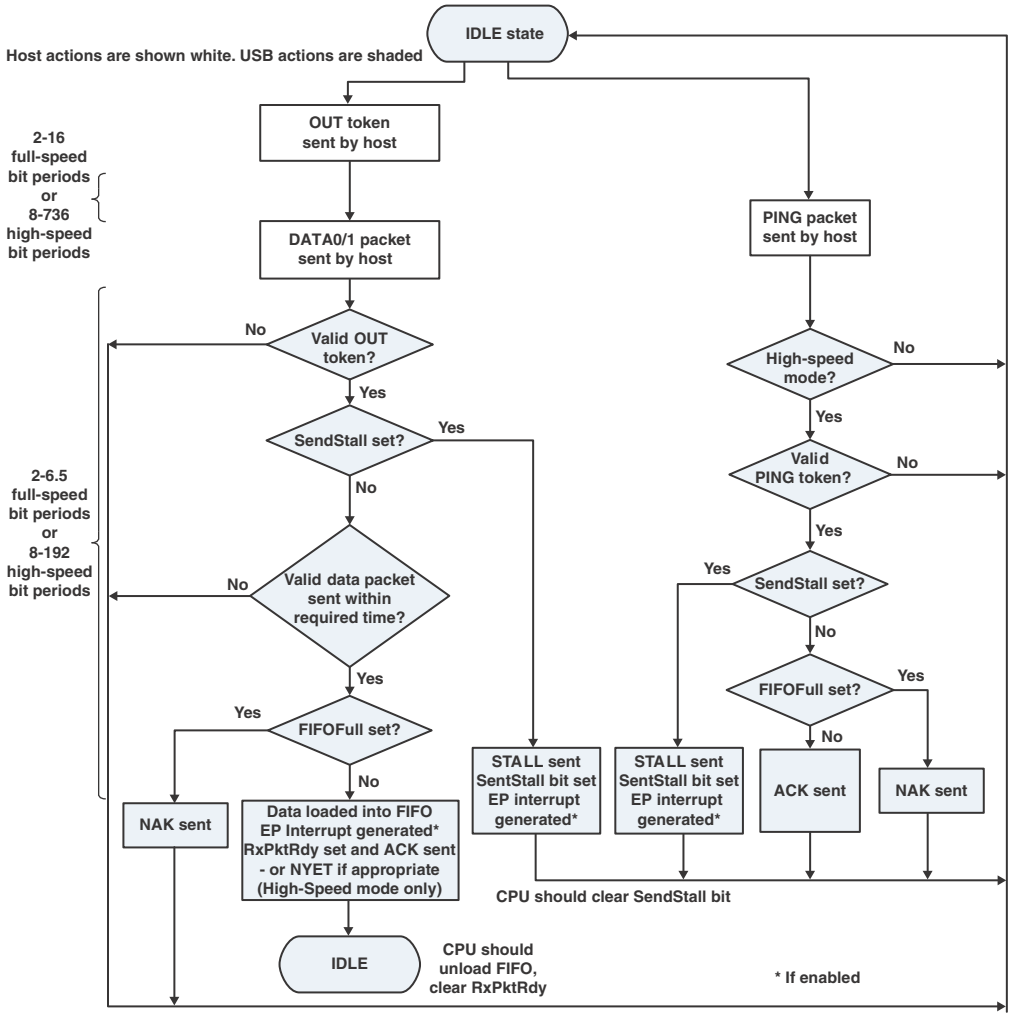


Figure 22-17: Bulk/Low Bandwidth Interrupt Out Transaction

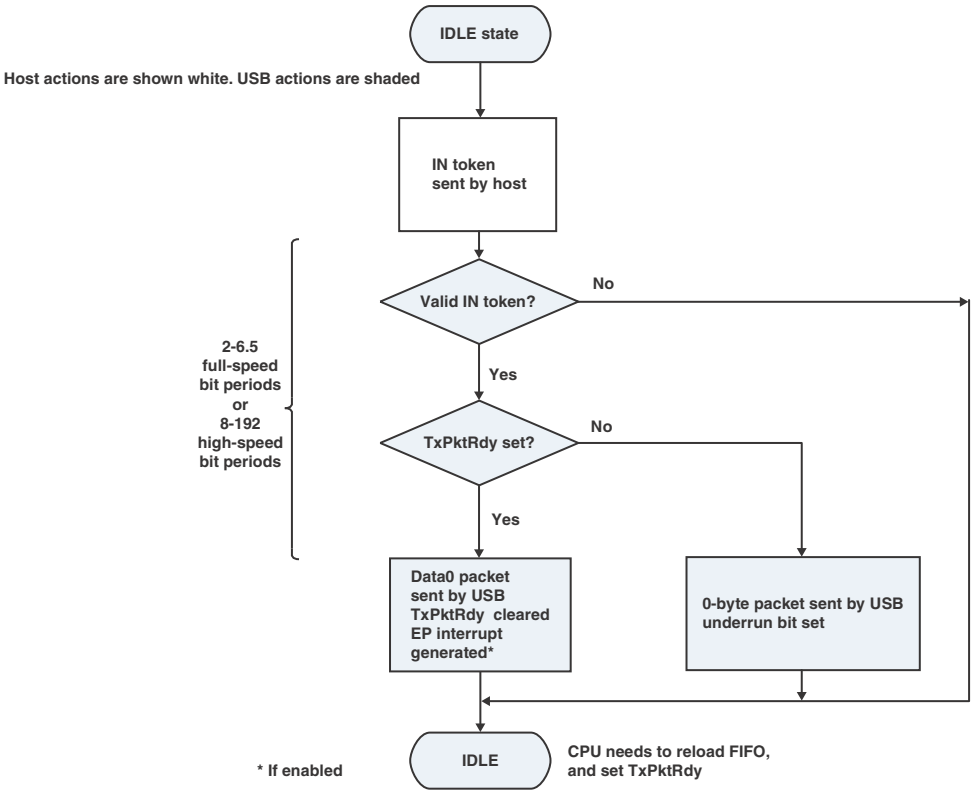


Figure 22-18: Full-speed/Low Bandwidth Isochronous In Transaction

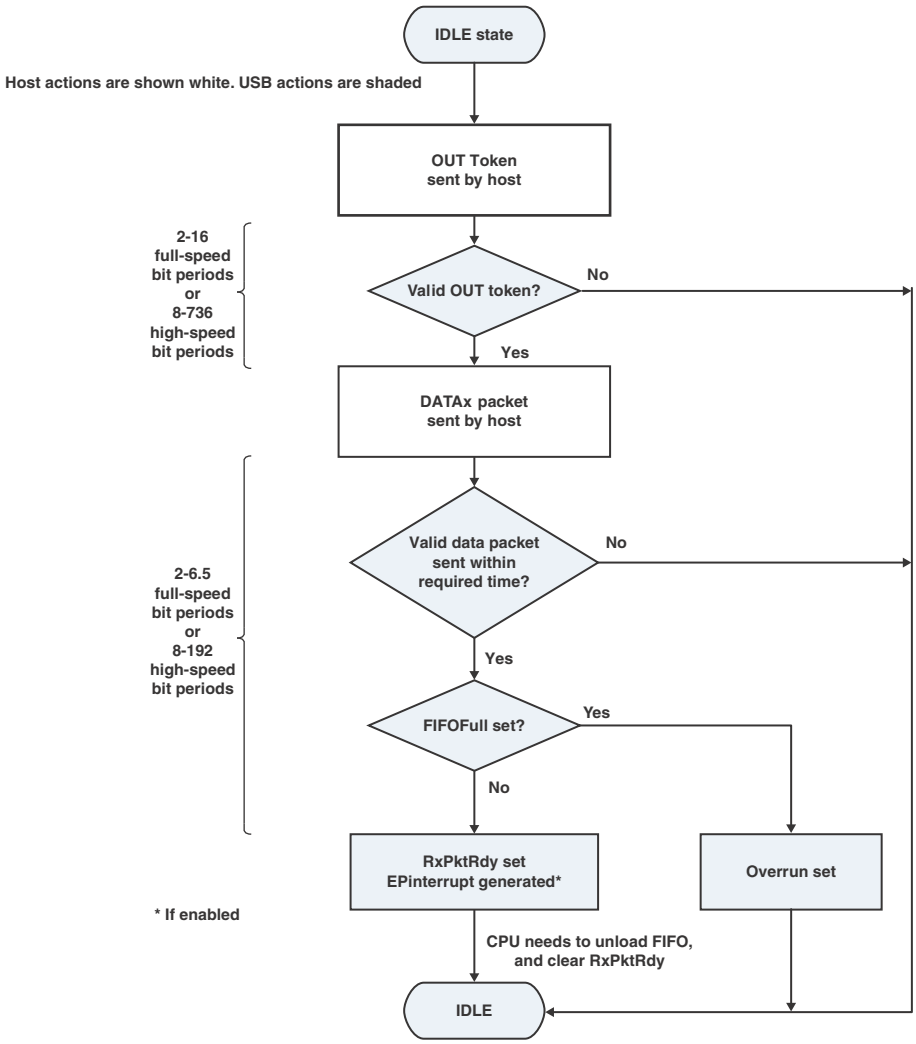


Figure 22-19: Full-speed/Low Bandwidth Isochronous Out Transaction

Host Mode Flow Charts

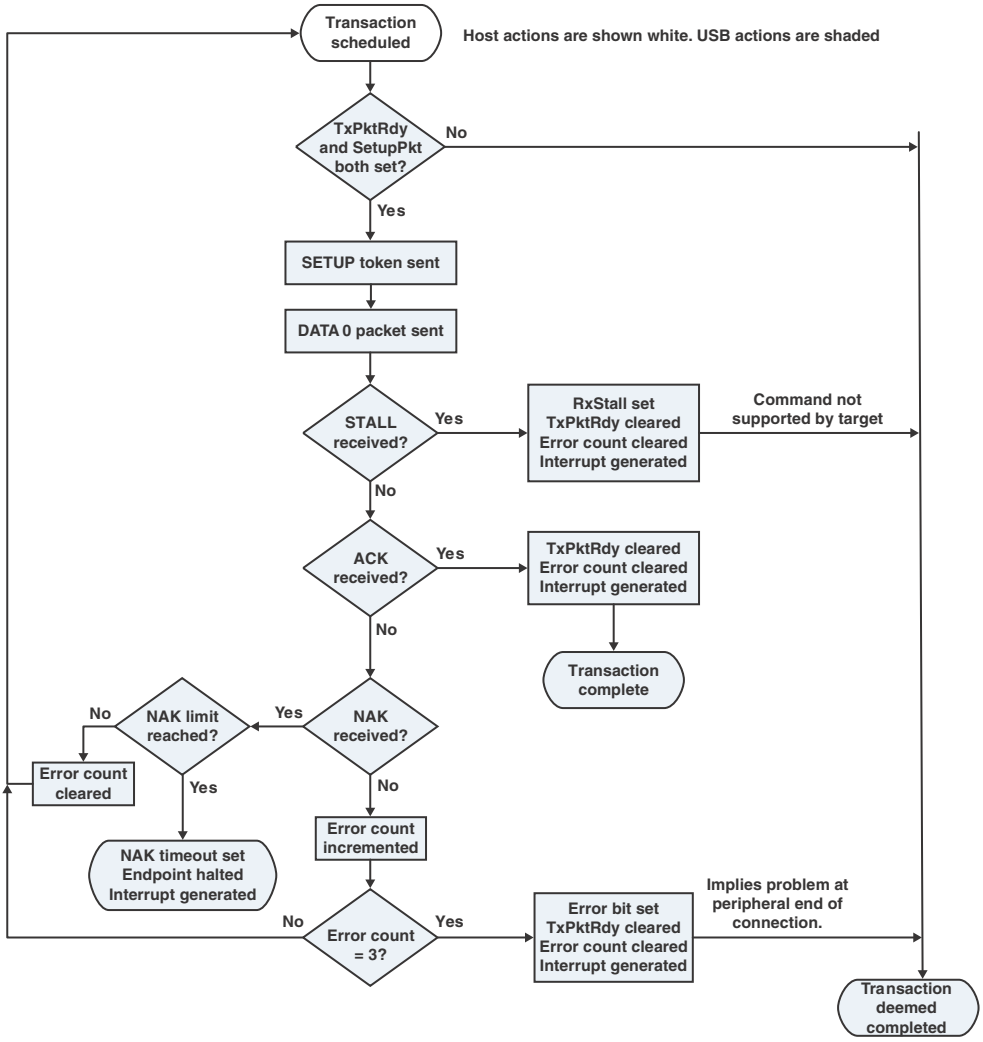


Figure 22-20: USB Control Setup Phase

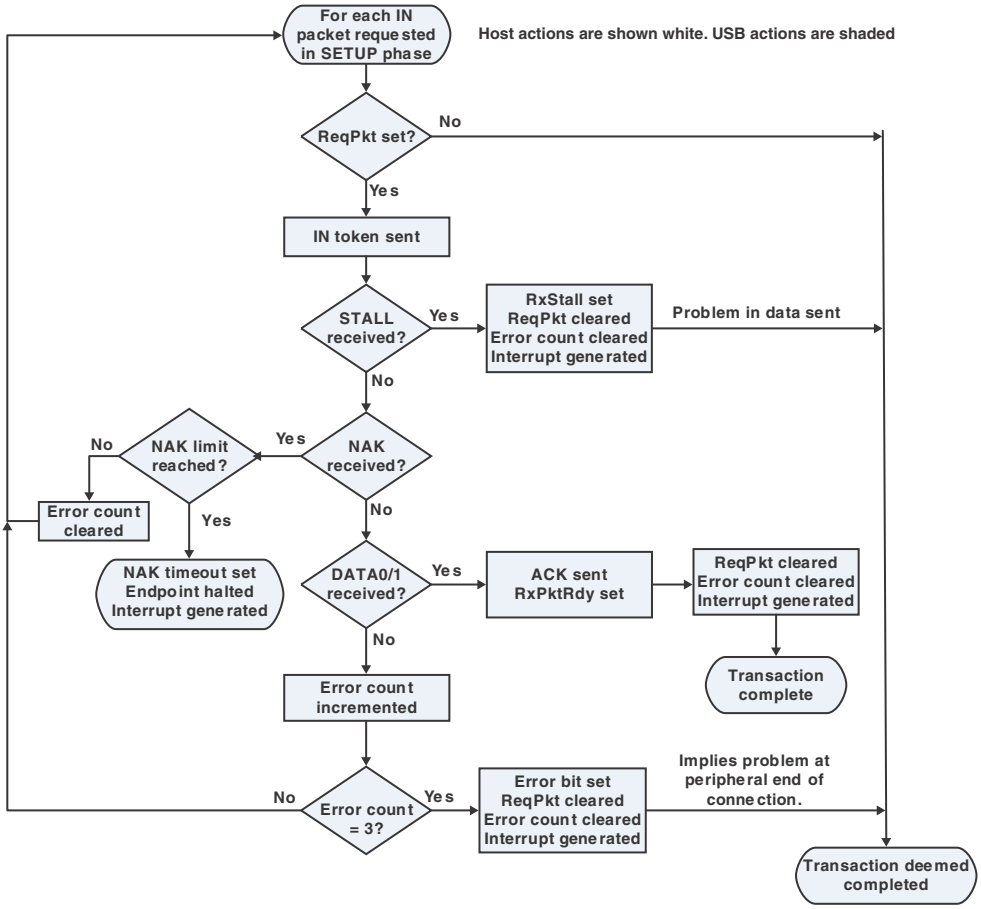


Figure 22-21: Control In Data Phase

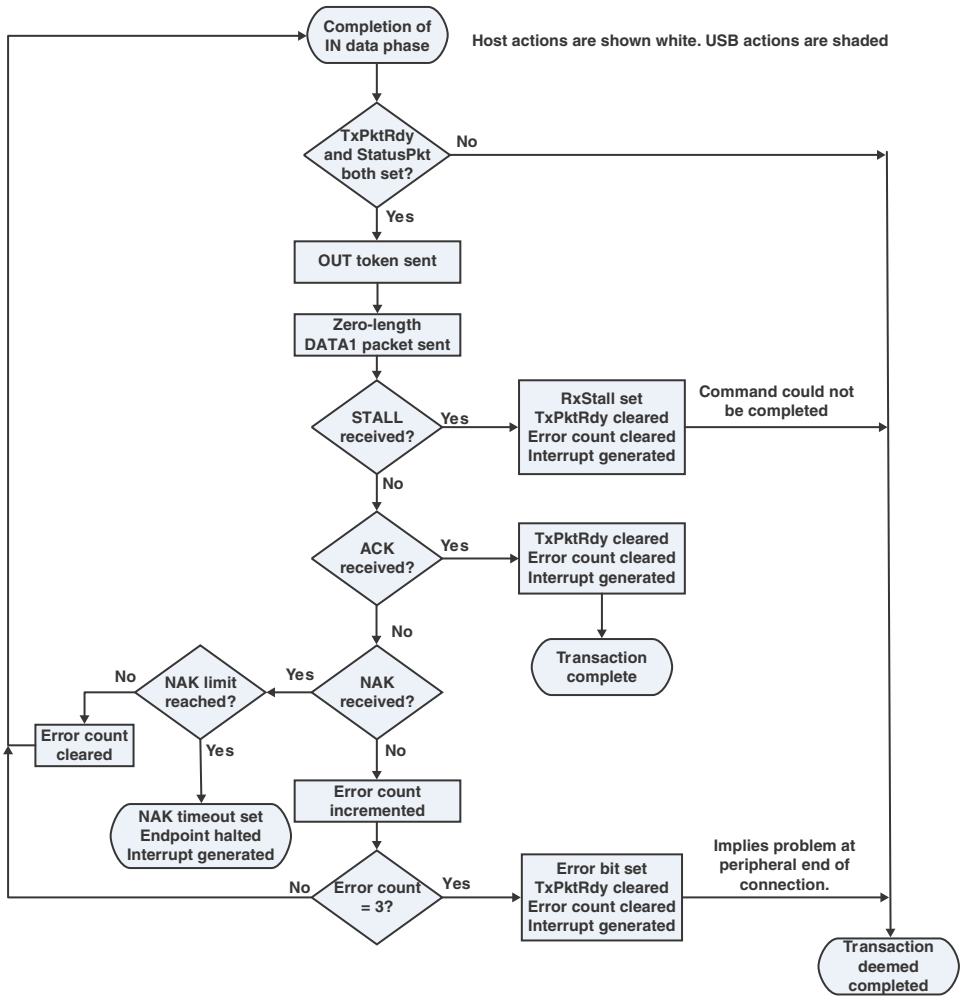


Figure 22-22: Control In Data Status Phase

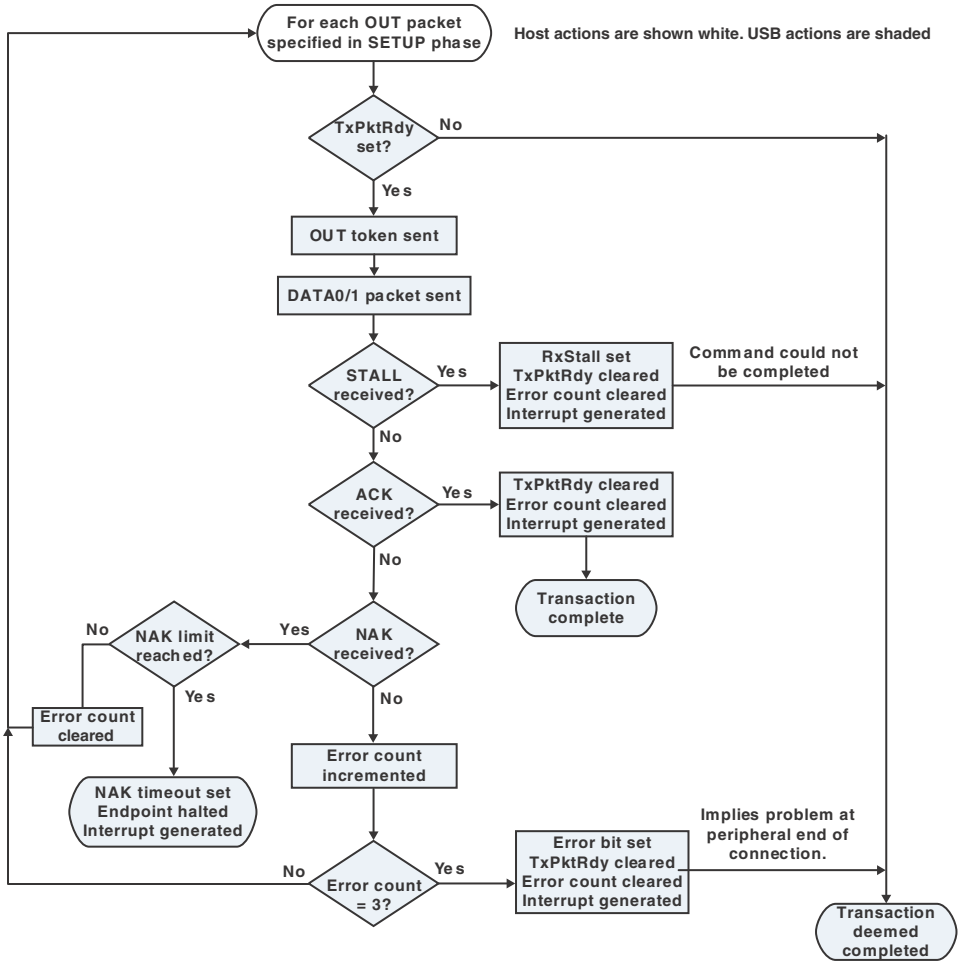


Figure 22-23: Control Out Data Phase

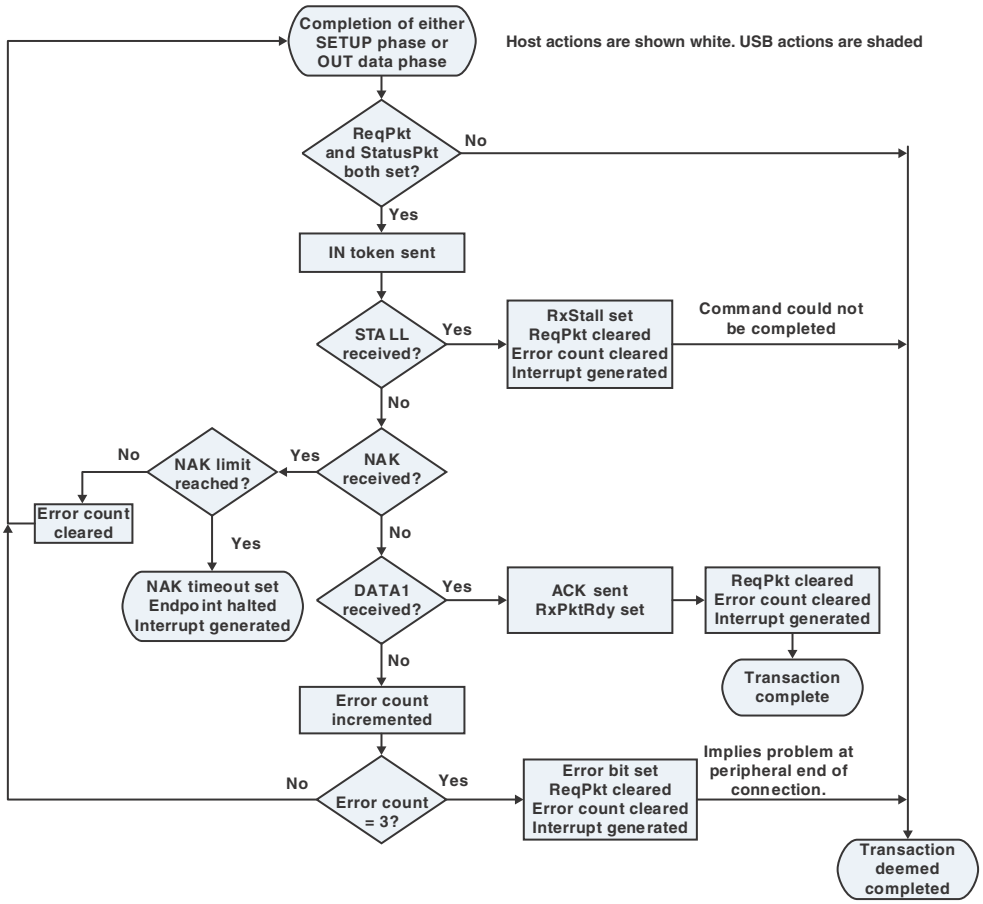


Figure 22-24: Control Out Data Status Phase

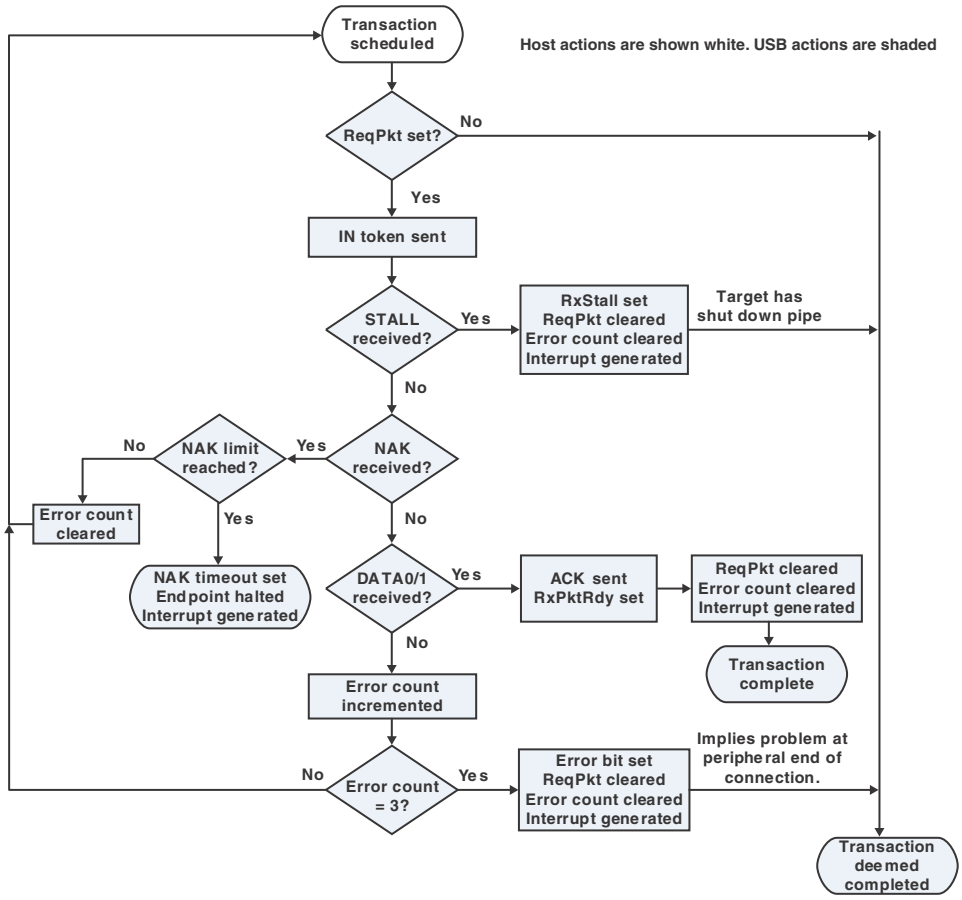


Figure 22-25: Bulk/Low Bandwidth Interrupt In Transaction

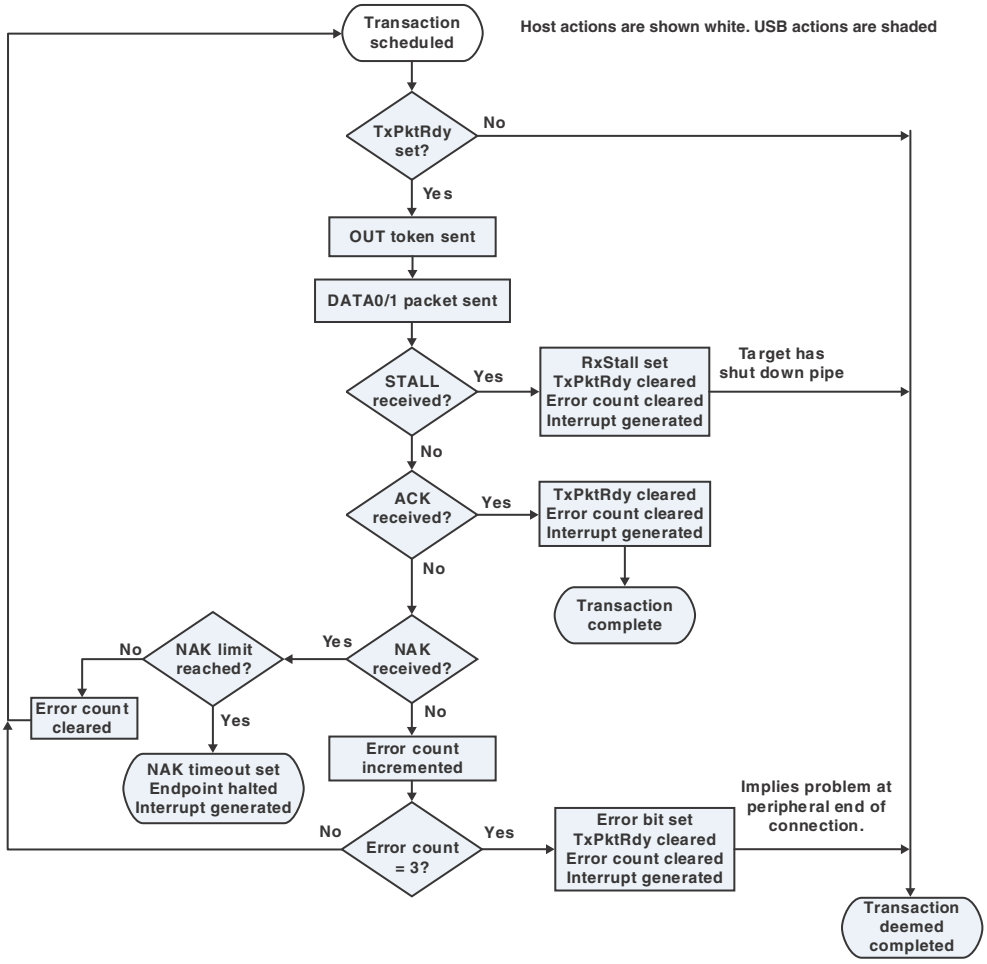


Figure 22-26: Bulk/Low Bandwidth Interrupt Out Transaction

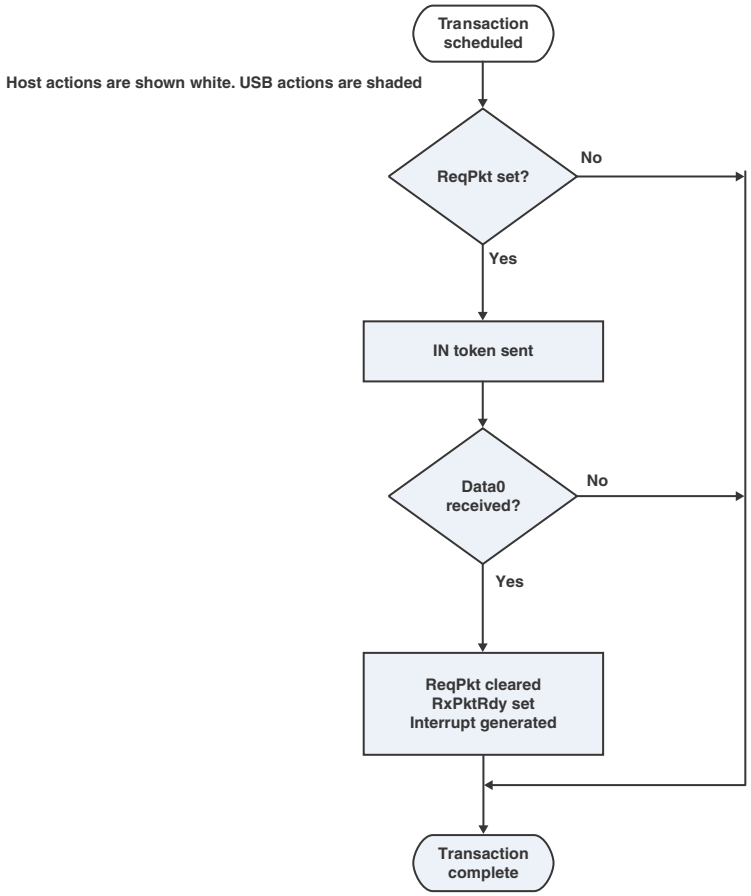


Figure 22-27: Full-speed/Low Bandwidth Isochronous In Transaction

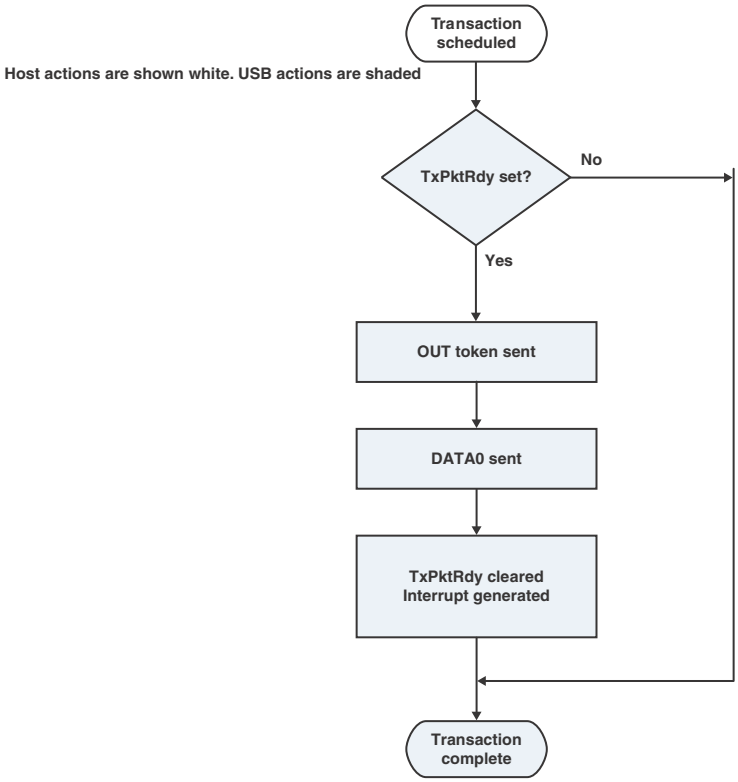


Figure 22-28: Full-speed/Low Bandwidth Isochronous Out Transaction

DMA Mode Flow Charts

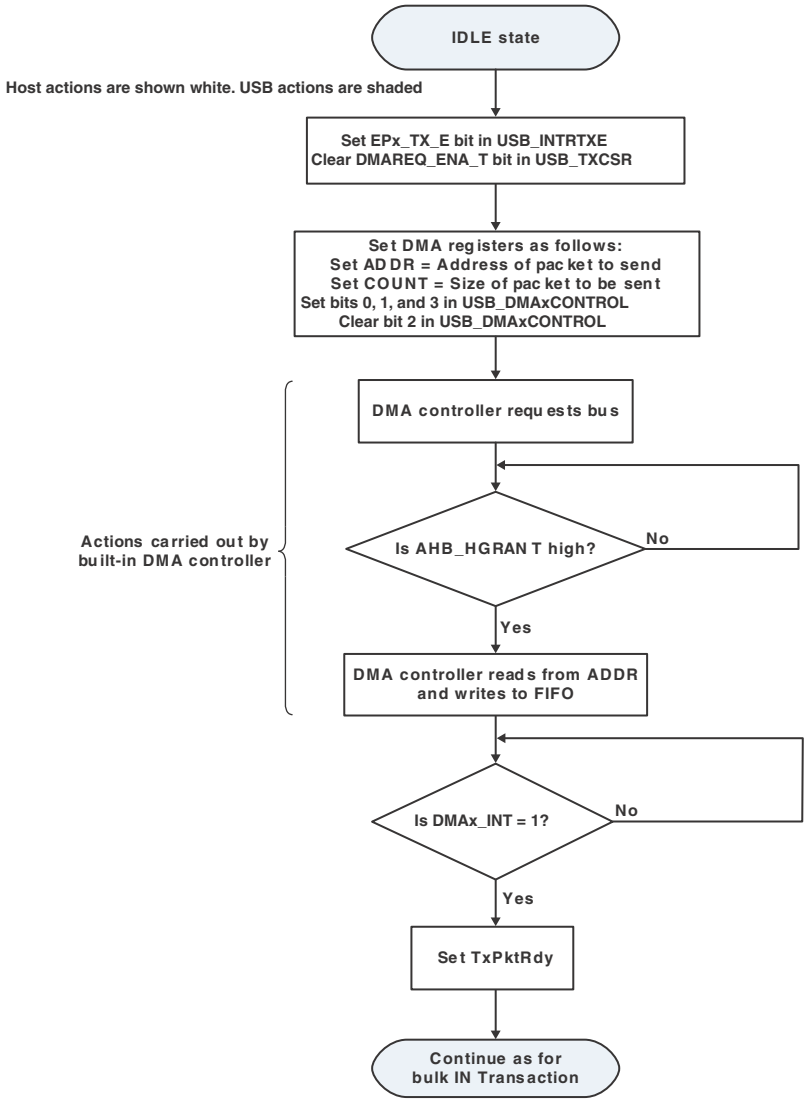


Figure 22-29: Single Packet Transmit During DMA Operation

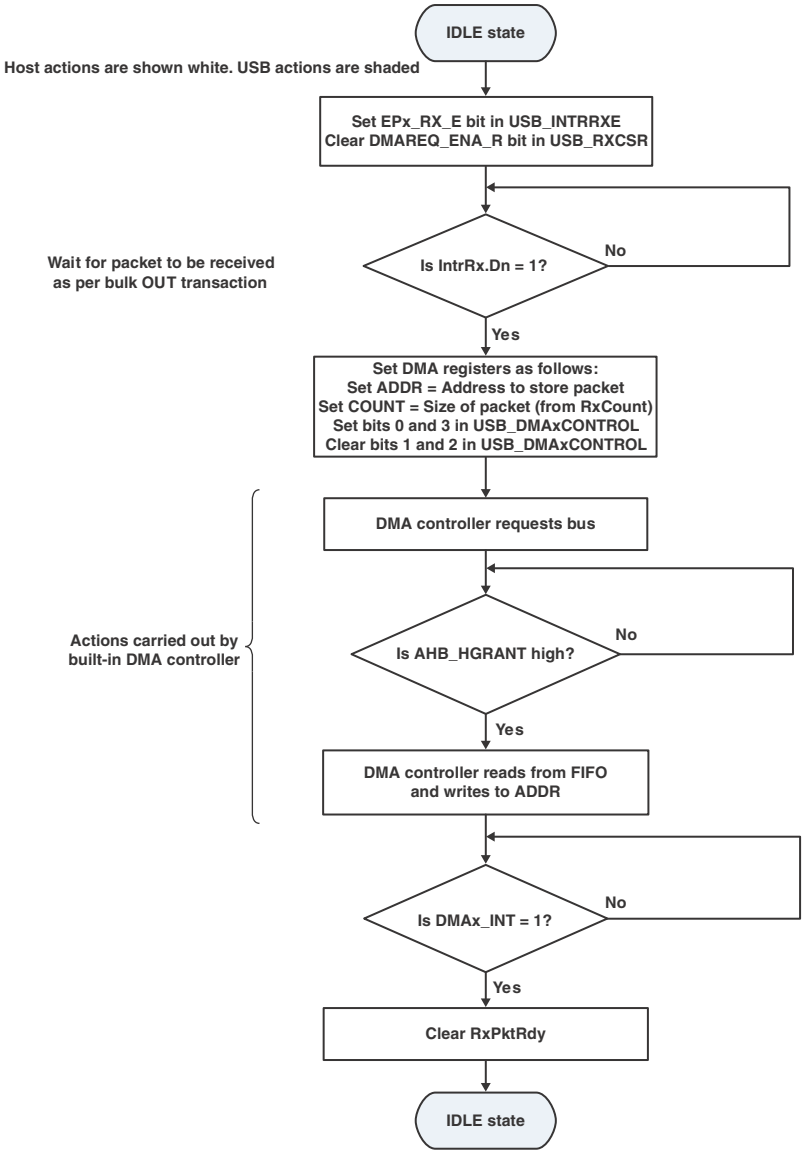


Figure 22-30: Single Packet Receive During DMA Operation

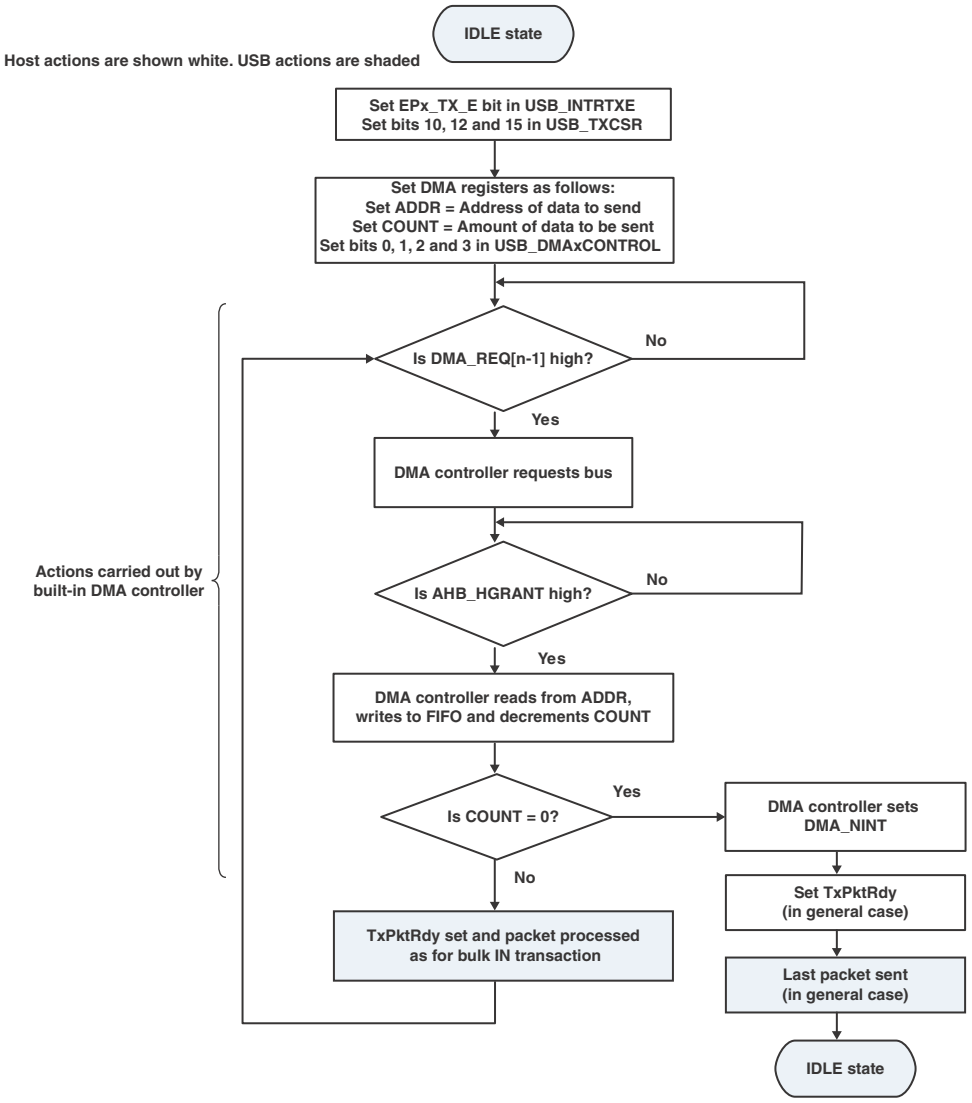


Figure 22-31: Multiple Packet Transmit During DMA Operation

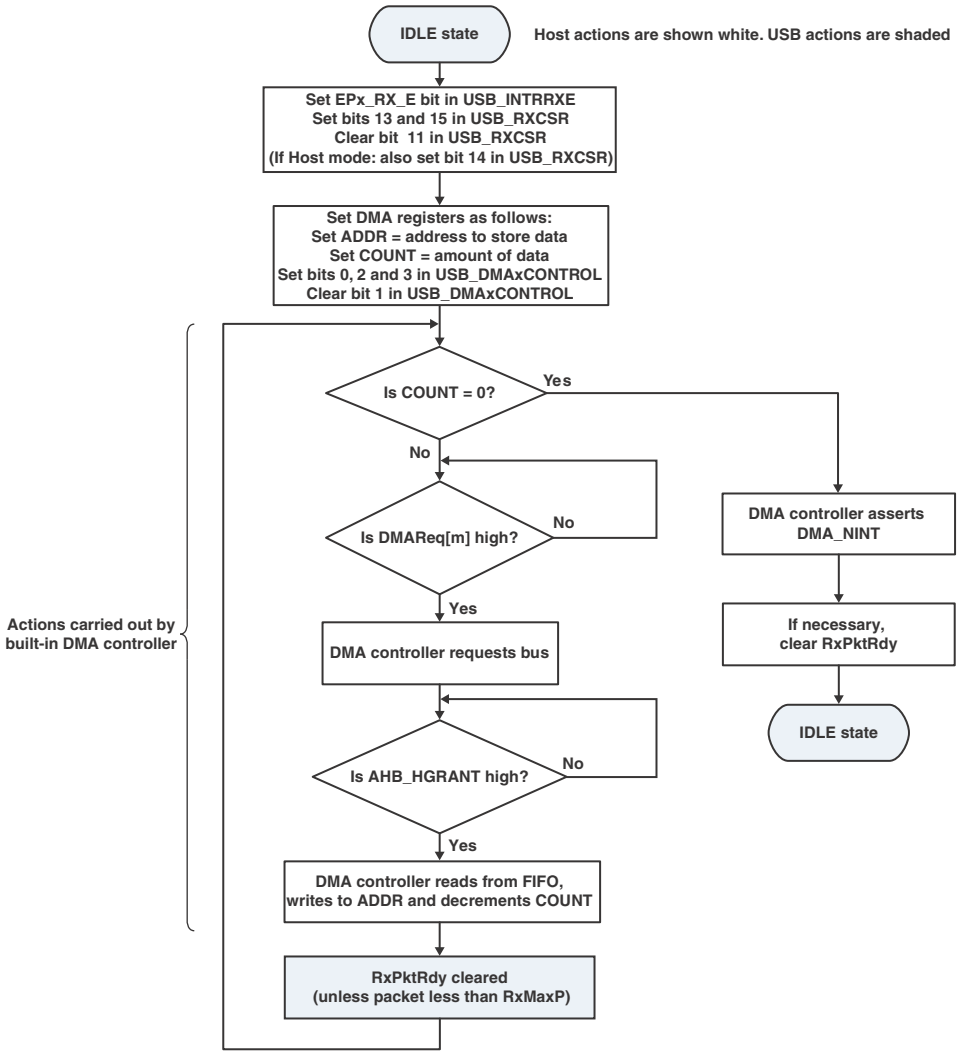


Figure 22-32: Multiple Packet Receive During DMA Operation (Data Size Known)

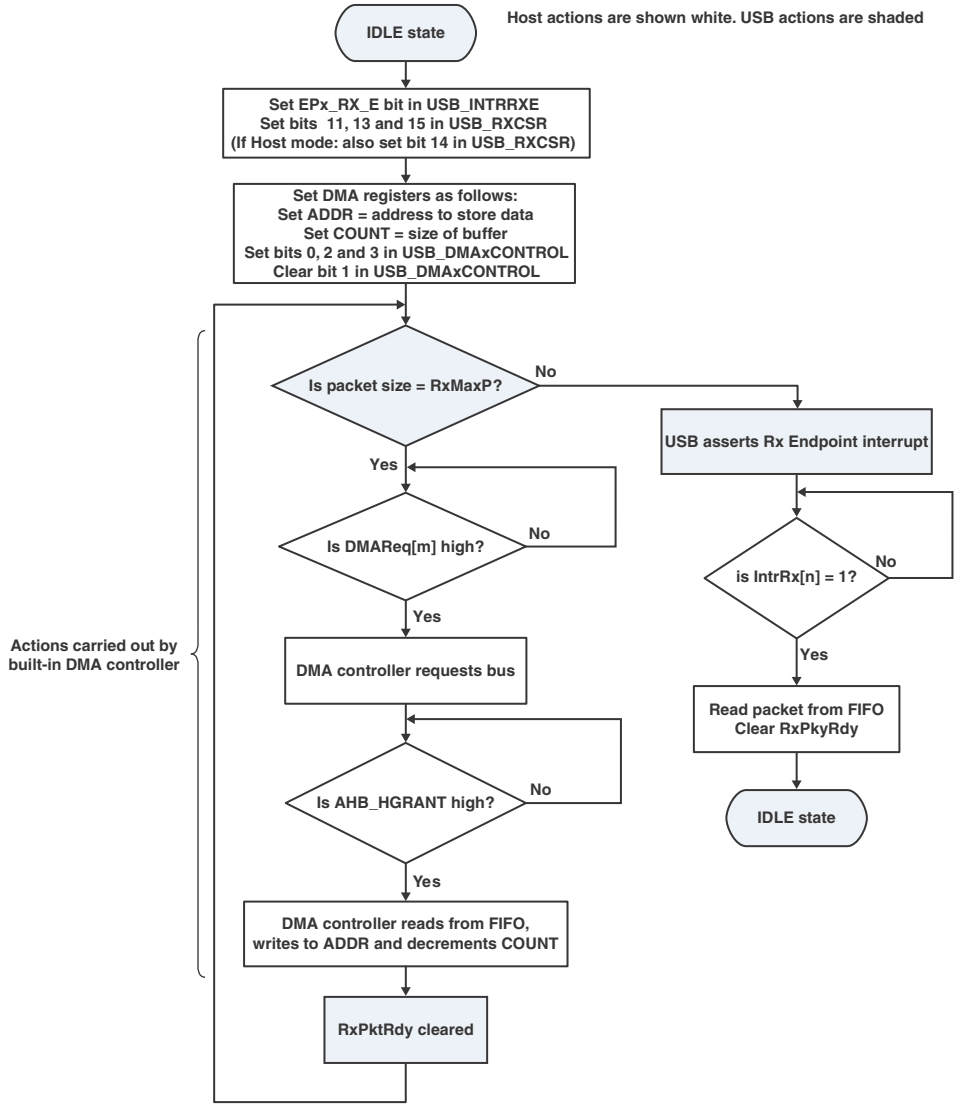


Figure 22-33: Multiple Packet Receive During DMA Operation (Data Size Not-known)

OTG Session Request

In order to conserve power, the USB on-the-go supplement allows VBUS to only be powered up when required and to be turned off when the bus is not in use.

VBUS is always supplied by the A device on the bus. The USB controller determines whether it is the A device or the B device by sampling the `USB_ID` input from the PHY. This signal is pulled low when an A-type plug is sensed (signifying that the USB controller is the A device), but the input is taken high when a B-type plug is sensed (signifying that the USB controller is the B device).

Starting a Session

When the device containing the USB controller wants to start a session, the processor core must set the USB_DEV_CTL.SESSION bit. The USB controller then enables ID pin sensing. This results in the USB_ID input either being taken low if an A-type connection is detected or high if a B-type connection is detected. The USB_DEV_CTL.BDEVICE bit is also set to indicate whether the USB controller has adopted the role of the A device or the B device.

The USB controller is the A device. The USB controller then enters host mode (the A device is always the default host), and waits for VBUS to go above the VBUS valid threshold, as indicated when the USB_DEV_CTL.VBUS bits go to 11.

The USB controller then waits for a peripheral to be connected. When a peripheral is detected, a connect interrupt (USB_IRQ.CON bit) is generated (if enabled) and either the USB_DEV_CTL.FSDEV or USB_DEV_CTL.LSDEV bits is set, depending on whether a full-speed peripheral or a low-speed peripheral was detected. The processor core should then reset this peripheral. To end the session, the processor core should clear the USB_DEV_CTL.SESSION bit.

The USB controller is the B device. The USB controller requests a session using the session request protocol defined in the USB on-the-go supplement. This is accomplished by setting the USB_DEV_CTL.SESSION bit.

At the end of the session, the USB_DEV_CTL.SESSION bit is cleared—usually by the USB controller but it can also be cleared by the processor core if the application software wishes to perform a software disconnect. For more information, see the description of the USB_DEV_CTL register. The USB controller switches on the pull-up resistor on D+. This signals to the A device to end the session.

Detecting Activity

When the other device of the OTG set-up wants to start a session, it either raises VBUS above the session valid threshold (if it is the A device as indicated by the USB_DEV_CTL.VBUS bits=10), or (if it is the B device) first pulses the data line then pulses VBUS. Depending on which of these actions happens, the USB controller can determine whether it is the A device or the B device in the current set-up and act accordingly.

If VBUS is raised above the session valid threshold, the USB controller is the B device. The USB controller sets the USB_DEV_CTL.SESSION bit. When reset signaling is detected on the bus, a reset interrupt (USB_IRQ.RSTBABBLE=1) is generated (if enabled) that the processor core should interpret as the start of a session. The USB controller is in peripheral mode at this point as the B device is the default peripheral.

At the end of the session, the A device turns off the power to VBUS. When VBUS drops below the session valid threshold (as indicated by the USB_DEV_CTL.VBUS bits=01), the USB controller detects this and clears the USB_DEV_CTL.SESSION bit to indicate that the session has ended. A disconnect interrupt (USB_IRQ.DISCON bit) is also generated (if enabled).

If data line/VBUS pulsing is detected, the USB controller is the A device. The controller generates a USB_IRQ.SESSREQ interrupt to indicate that the B device is requesting a session. The processor core should then start a session by setting the USB_DEV_CTL.SESSION bit.

Host Negotiation Protocol

When the USB controller is the A device (USB_ID low, USB_DEV_CTL.BDEVICE=0), the controller automatically enters host mode when a session starts.

When the USB controller is the B device (USB_ID high, USB_DEV_CTL.BDEVICE=1), the controller automatically enters peripheral mode when a session starts. The processor core can request that the USB controller become the host by setting the USB_DEV_CTL.HOSTREQ bit. This bit can be set either when requesting a session start by setting the USB_DEV_CTL.SESSION bit or at any time after a session has started.

When the USB controller next enters suspend mode (no activity on the bus for 3 ms), and assuming the USB_DEV_CTL.HOSTREQ bit remains set, the controller enters host mode and begins host negotiation (as specified in the USB OTG supplement), causing the PHY to disconnect the pull-up resistor on the D+ line. This should cause the A device to switch to peripheral mode and to connect its own pull-up resistor. When the USB controller detects this, it generates a connect interrupt (USB_IRQ.CON bit). The controller also sets the USB_POWER.RESET bit to begin resetting the A device. (The USB controller begins this reset sequence automatically to ensure that reset is started as required within 1 ms of the A device connecting its pull-up resistor). The processor core should wait at least 20 ms, then clear the USB_POWER.RESET bit and enumerate the A device.

When the USB controller-based B device has finished using the bus, the processor core should put it into suspend mode by setting the USB_POWER.SUSPEND. The A device should detect this and either terminate the session or revert to host mode. If the A device is USB controller-based, it generates a disconnect interrupt (USB_IRQ.DISCON bit) if enabled.

Wakeup from Hibernate State

To conserve power when the chip is idle, systems often uses power-down modes to shut down power and clocks to various parts of the chip. Hibernate state saves the most power (core clock, peripherals clocks, and internal power are off; only external power is on).

During the course of normal operation, the software can decide that the chip has been idle for a long enough period that there is no immediate need for the clocks to be active and the chip can be put into a power-down mode such as hibernate. This period of inactivity occurs when there is a USB suspend state (idle on the bus for greater than 3 ms) or if no OTG session is valid. The USB_POWER.SUSPEND bit and USB_DEV_CTL.VBUS status bits are used to indicate these states.

Before the system software (driver) pushes processor into the hibernate state, the software has to make sure that the USB_PHY_CTL.HIBER bit is set. Setting this bit activates the non-idle activity detection logic in the PHY. Any non-idle activity on the USB bus is detected by the non-idle activity detection logic in the analog PHY. This logic wakes up the processor and generates a low to high transition on the SYS_EXTWAKEpin.

To use non-idle activity detection logic as a wakeup source for the processor, enable the USB wakeup source by programming the appropriate bits in the DPM wakeup enable register (DPM_WAKE_EN). After the processor wakes up, USB is listed as the wakeup source in the DPM wakeup status (DPM_WAKE_STAT) register. The SYS_EXTWAKEpin can be used by the external power-up sequence chip to power up SDRAM

or an other external peripheral. The processor typically goes through these steps when it comes out of hibernate state.

After the chip comes out of hibernate state, the software has to make sure that the USB_PHY_CTL.RESTORE bit is set. This setting deactivates the non-idle activity detection logic and ensures proper USB functionality.

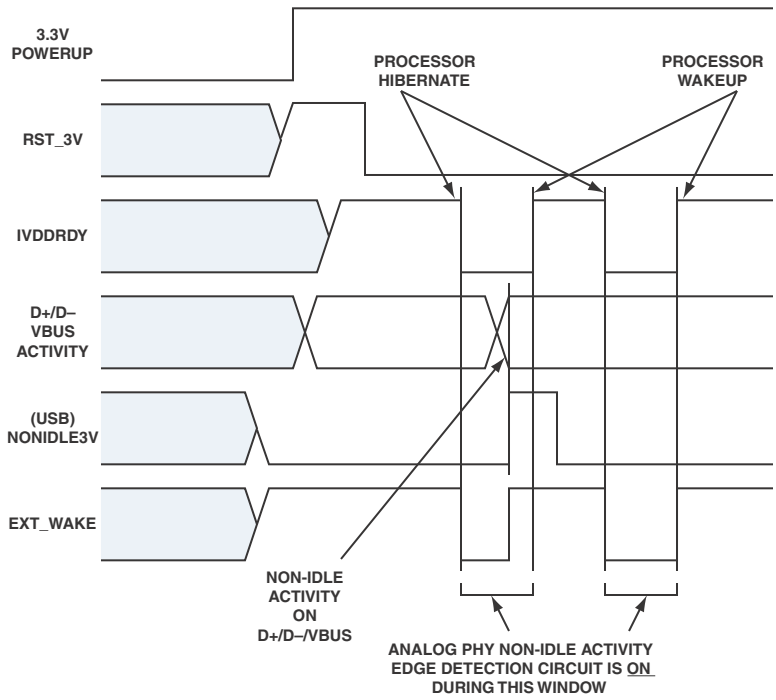


Figure 22-34: Timing Diagram of EXT_WAKE Pin

The interrupt will be asserted when either of the following events occur:

- Non-idle signaling occurs during the USB suspend state (including USB reset signaling)
- VBUS falls below the session valid threshold

Data Transfer

Regardless of whether the USB controller is operating in host or peripheral mode, data is channeled through the endpoint FIFOs to construct packets that are sent or received over the USB. The RX FIFOs are used to receive OUT packets when in peripheral mode and IN packets when operating in host mode. Similarly, the TX FIFOs are used to transmit IN packets when in peripheral mode and OUT packets as a host.

Data may be moved between the FIFOs and memory using either DMA or core accesses. Each endpoint FIFO has its own individually programmable options so that each can be set up separately. Different transfer types must be treated differently by the system. Data transfers of significant size almost certainly

require DMA to move the data around; but smaller packet sizes might be handled completely by the processor.

Each data endpoint supports both double and single-buffering modes. In single-buffered operation, FIFOs are unloaded and loaded on a packet-by-packet basis. Double-buffering imposes less burden on the system by allowing two packets to be buffered in a FIFO before it is necessary to use DMA/interrupts to service the FIFO. Double-buffering mode is automatically enabled when a *MaxPktSize* is set for an endpoint that is equal to or less than half the size in bytes of that FIFO.

Loading/Unloading Packets from Endpoints

Transfers to and from the FIFOs can be 32-bit, 16-bit, or 8-bit. When using core accesses, the same width must be used for transfers associated with one data packet, so that data is consistently byte, half-word or word aligned. The last transfer may, however, contain fewer bytes than the previous transfers in order to complete an 8-bit or 16-bit transfer.

When using the DMA to access the FIFOs, the only requirement is that the starting DMA address be word aligned, or aligned on a 32-bit boundary. The packet transfer starts with a word transfer, but half-word and/or byte transfers may be added at the end to handle any left overs.

DMA Master Channels

The USB controller provides eight DMA master channels to provide a more efficient transfer of larger amounts of data between the FIFOs and the processor core; and to free up the processor core for other tasks. Each of these channels is configured and controlled using the DMA control registers.

Each DMA controller can operate in one of two DMA modes: 0 or 1. When operating in mode 0, the DMA controller only can be programmed to load or unload one packet, so processor intervention is required for each packet transferred over the USB. This mode can be used with any endpoint, whether it uses control, bulk, isochronous, or interrupt transactions.

When operating in DMA mode 1, the DMA controller can only be programmed to load/unload a complete bulk transfer, which can be many packets. After set up, the DMA controller loads or unloads the packets, interrupting the processor only when the transfer has completed. DMA mode 1 can only be used with endpoints that use bulk transactions and is most valuable where large blocks of data are transferred to a bulk endpoint. The USB protocol requires such packets to be split into a series of packets of *MaxPktSize* for the endpoint.

Mode 1 can be used to avoid the overhead of having to interrupt the processor after each individual packet because the processor is only interrupted after the transfer has completed. In some cases, the block of data transferred comprises a predefined number of these packets that the controlling software counts through the transfer process. In other cases, the last packet in the series may be less than the maximum packet size and the receiver may use this short packet to signal the end of the transfer. If the total size of the transfer is an exact multiple of the maximum packet size, the transmitting software should send a null packet for the receiver to detect.

NOTE:

Each channel can be independently programmed for the selected operating mode.

NOTE:

For bulk OUT transfers using DMA mode 1, the DMA request line is asserted only when there is an edge transition of the state of the USB_EPn_RXCSR_H.RXPKTRDY and a payload of *MaxPktSize* has been received. If a data packet has been sitting in the FIFO prior to setting the DMA request mode bits (USB_EPn_RXCSR_H.DMAREQMODE or USB_EPn_RXCSR_P.DMAREQMODE), the DMA request line is not asserted when the DMA is enabled using the USB_DMAn_CTL.EN bit. This causes the data to not be read from the RX FIFO, resulting in a DMA hang. However, since the packet arrived before DMA request mode and DMA request enable bits (USB_EPn_RXCSR_H.DMAREQEN or USB_EPn_RXCSR_P.DMAREQEN) were enabled, an RX interrupt is generated for the corresponding endpoint. Therefore, the software should set the DMA request mode to request mode 0 to unload the pre-received packet. The RX interrupt service routine may be similar to the following.

If USB_EPn_RXCNT == MaxPktSize

Switch to DMA mode 0 and unload the packet (in mode 0, the DMA request enable is always asserted whenever there is data in the FIFO)

Set the USB_EPn_RXCNT to MaxPktSize so as to unload only one packet

If USB_EPn_RXCSR_H.AUTOCLR is set, USB_EPn_RXCSR_H.RXPKTRDY does not need to be cleared manually.

Switch back to DMA Mode 1 and set the count to

(Total_Count – MaxPktSize)

Else

Handle as normal for case of short packet

DMA transfers may be 8-bit, 16-bit, or 32-bit. All transfers associated with one packet (with the exception of the last) must be of the same width, so that the data is consistently byte-aligned or word-aligned. The last transfer may contain fewer bytes than the previous transfers in order to complete an odd-byte or odd-word transfer.

DMA Bus Cycles

The DMA controller uses incrementing bursts of an unspecified length on the peripheral DMA bus. The controller starts a new burst when it is first granted bus mastership (whether at the start of a USB packet or when regaining the bus after losing it after a partial packet) and when the peripheral address starts a new 1K byte block.

When unloading packets from the FIFOs, the DMA controller requests ahead to the USB controller. Although it starts the transfer with two BUSY cycles while it is getting the first word from the FIFO, all subsequent words of the packet are immediately available and no further BUSY cycles are required. The

DMA controller is associated with a two-word buffer, so no data is lost if it loses bus mastership in the middle of unloading a packet. When bus mastership is regained, it can continue unloading the packet without adding any BUSY cycles.

The DMA start address (written to the USB_DMAn_ADDR register) must be word aligned. Split transactions and retries are supported.

The DMA request lines are individually enabled using the appropriate DMA request enable bit (there are four options: TX peripheral and host and RX peripheral and host) and operate in two modes, referred to as DMA request mode 0 and DMA request mode 1. The operating mode is configured using the appropriate DMA request mode bit (there are four options: TX peripheral and host and RX peripheral and host).

NOTE: When operating in host mode, if either the USB_EPn_TXCSR_H.RXSTALL bit or the USB_EPn_TXCSR_H.TXTOERR is set following three failed attempts to transmit a packet, the DMA request line is disabled until the bits have been cleared.

The mode selected also affects the generation of Endpoint interrupts (if enabled). In DMA request mode 0, no interrupt is generated when packets are received but the appropriate Endpoint interrupt is generated to prompt the loading of all packets. In DMA request mode 1, the Endpoint interrupt is suppressed except following the receipt of a short packet (one less than USB_EPn_RXMAXP bytes).

Table 22-7: Endpoint Interrupt Associated with the Receive Packet Ready Bit=1

DMAReqEnab	DMAReqMode	EP Interrupt Generated?
0	X	YES
1	0	NO
1	1	Only is short packet

Table 22-8: Endpoint Interrupt Associated with the Receive Packet Ready Bit=0

DMAReqEnab	DMAReqMode	EP Interrupt Generated?
0	X	YES
1	0	YES
1	1	NO

NOTE: The USB_EPn_TXMAXP/USB_EPn_RXMAXP registers must be set to an even number of bytes for proper interrupt generation in DMA mode 1.

DMA transfers may be 8-bit, 16-bit, or 32-bit as required. However, all transfers associated with one packet (with the exception of the last) must be of the same width so that the data is consistently byte-, word- or double-word-aligned. The last transfer may contain fewer bytes than the previous transfers in order to complete an odd-byte or odd-word transfer.

NOTE: DMA requests should be disabled before the DMA request mode bit is changed. In particular, the USB_EPn_TXCSR_H.DMAREQMODE bit should not be set to zero either before or in the same cycle as the corresponding USB_EPn_TXCSR_H.DMAREQEN bit is cleared to zero.

Transferring Packets Using DMA

Use of the DMA master channels to access the USB controller FIFOs requires that both the appropriate channel and the endpoint be programmed appropriately. Many variations are possible. The following sections detail the standard set ups used for the basic actions of transferring individual and multiple packets.

Individual RX Endpoint Packet

The transfer of individual packets is normally carried out using DMA mode 0. The USB controller RX endpoint is programmed as follows.

1. The relevant bit in the USB_INTRRXE register is set to 1.
2. The DMA enable bit of the appropriate USB_EPn_RXCSR_H.DMAREQEN/USB_EPn_RXCSR_P.DMAREQEN register is set to 0. (There is no need to set the USB controller to support DMA for this operation.)
3. When a packet is received by the USB controller, it generates the appropriate endpoint interrupt (using the USB_INTRRXE register). The processor should then program the appropriate DMA master channel as follows.
    - Configure the USB_DMAn_ADDR register with the memory address to store the packet
    - Configure the USB_DMAn_CNT register with the size of packet (determined by reading the USB controller USB_RQPKTCNTn register)
    - Configure the USB_DMAn_CTL register using the following bit settings:USB_DMAn_CTL.IE=1 USB_DMAn_CTL.EN=1, USB_DMAn_CTL.DIR=0, USB_DMAn_CTL.MODE=0

The DMA controller then requests bus mastership and transfers the packet to memory. It interrupts the processor when it has completed the transfer. The processor should then clear the USB_EPn_RXCSR_H.RXPKTRDY bit.

Individual TX Endpoint Packet

Using DMA mode 0, a USB controller TX endpoint is programmed as follows.

1. The relevant bit in the USB_INTRTXE register is set to 1.
2. The DMA enable bit of the appropriate USB_EPn_TXCSR_H.DMAREQEN/USB_EPn_TXCSR_P.DMAREQEN register is set to 0. (There is no need to set the USB controller to support DMA for this operation.)
3. When the FIFO can accommodate data, the USB controller interrupts the processor with the appropriate TX endpoint interrupt. The processor should then program the DMA channel as follows:
    - Configure the USB_DMAn_ADDR register with the memory address to store the packet
    - Configure the USB_DMAn_CNT register with the size of packet

- Configure the USB_DMAn_CTL register using the following bit settings: USB_DMAn_CTL.IE=1 USB_DMAn_CTL.EN=1, USB_DMAn_CTL.DIR=1, USB_DMAn_CTL.MODE=0

The DMA controller then requests bus mastership and transfers the packet to the USB controller FIFO. When it has completed the transfer, it generates a DMA interrupt. The processor should then set the USB_EPn_TXCSR_H.TXPKTRDY bit.

Multiple RX Endpoint Packets

Multiple packets normally are transferred using DMA mode 1. The DMA controller is programmed using the DMA registers:

- Configure the USB_DMAn_ADDR register with the memory address of data block to send
- Configure the USB_DMAn_CNT register with the maximum size of data buffer
- Configure the USB_DMAn_CTL register using the following bit settings: USB_DMAn_CTL.EN=1, USB_DMAn_CTL.IE=1, USB_DMAn_CTL.DIR=0, USB_DMAn_CTL.MODE=1

The USB controller RX endpoint should now be programmed as follows:

1. The relevant bit in the USB_INTRRX register is set to 1.
2. The USB_EPn_RXCSR_H.AUTOCLR, USB_EPn_RXCSR_H.DMAREQEN and USB_EPn_RXCSR_H.DMAREQMODE bits of the appropriate receive control and status register (host or peripheral) register is set to 1. In host mode, the USB_EPn_RXCSR_H.AUTOREQ and USB_EPn_RXCSR_H.DMAREQMODE bits should also be set to 1.

As each packet is received by the USB controller, the DMA master channel requests bus mastership and transfers the packet to memory. With USB_EPn_RXCSR_H.AUTOCLR set, the USB controller automatically clears its USB_EPn_RXCSR_H.RXPKTRDY bit. This process continues automatically until the USB controller receives a short packet (one of less than the maximum packet size for the endpoint) signifying the end of the transfer. This short packet is not transferred by the DMA controller: instead the USB controller interrupts the processor by generating the appropriate endpoint interrupt. The processor can then read the USB_EPn_RXCNTregister to see the size of the short packet and either unload it manually or reprogram the DMA controller in mode 0 to unload the packet.

The USB_DMAn_ADDR register is incremented as the packets are unloaded, so the processor can determine the size of the transfer by comparing the current value of USB_DMAn_ADDR with the start address of the memory buffer.

If the size of the transfer exceeds the data buffer size, the DMA controller stops unloading the FIFO and interrupts the processor.

Multiple TX Endpoint Packets

Using DMA mode 1 for a TX endpoint, the DMA controller is programmed as follows:

- Configure the USB_DMAn_ADDR register with the memory address of data block to send
- Configure the USB_DMAn_CNT register with the size of the data block
- Configure the USB_DMAn_CTL register using the following bit settings: USB_DMAn_CTL.EN=1, USB_DMAn_CTL.IE=1 USB_DMAn_CTL.DIR=1, USB_DMAn_CTL.MODE=1

The USB controller TX endpoint is programmed as follows:

1. The relevant bit in the USB_INTRTXE register is set to 1.
2. The USB_EPn_TXCSR_H.AUTOSET and USB_EPn_TXCSR_H.DMAREQEN bits of the appropriate transmit control and status register (host or peripheral) is set to 1.

When the FIFO in the USB controller becomes available, the DMA controller requests bus mastership and transfers a packet to the FIFO. With USB_EPn_TXCSR_H.AUTOSET set, the USB controller automatically sets the USB_EPn_TXCSR_H.TXPKTRDY bit. This process continues until the entire data block is transferred to the USB controller.

The DMA controller then interrupts the processor by taking the appropriate USB_DMA_IRQ register bit low. Note that:

- If the last packet loaded was less than the maximum packet size for the endpoint, the USB_EPn_TXCSR_H.TXPKTRDY bit is not set for this packet. The processor should respond to the DMA interrupt by setting the USB_EPn_TXCSR_H.TXPKTRDY bit to allow the last short packet to be sent.
- If the last packet loaded was of the maximum packet size, then the action to take depends on whether the transfer is under the control of an application such as the mass storage software on Windows system that keeps count of the individual packets sent.
- If the transfer is not under such control, the processor should respond to the DMA interrupt by setting the USB_EPn_TXCSR_H.TXPKTRDY bit. This has the effect of sending a null packet for the receiving software to interpret as indicating the end of the transfer.

ADSP-BF60x USB Register Descriptions

Universal Serial Bus Controller (USB) contains the following registers.

Table 22-9: ADSP-BF60x USB Register List

Name	Description
USB_FADDR	Function Address Register
USB_POWER	Power and Device Control Register

Table 22-9: ADSP-BF60x USB Register List (Continued)

Name	Description
USB_INTRTX	Transmit Interrupt Register
USB_INTRRX	Receive Interrupt Register
USB_INTRTXE	Transmit Interrupt Enable Register
USB_INTRRXE	Receive Interrupt Enable Register
USB_IRQ	Common Interrupts Register
USB_IEN	Common Interrupts Enable Register
USB_FRAME	Frame Number Register
USB_INDEX	Index Register
USB_TESTMODE	Testmode Register
USB_FIFOBn	FIFO Byte (8-Bit) Register
USB_FIFOHn	FIFO Half-Word (16-Bit) Register
USB_FIFOn	FIFO Word (32-Bit) Register
USB_DEV_CTL	Device Control Register
USB_TXFIFOSZ	Transmit FIFO Size Register
USB_RXFIFOSZ	Receive FIFO Size Register
USB_TXFIFOADDR	Transmit FIFO Address Register
USB_RXFIFOADDR	Receive FIFO Address Register
USB_EPINFO	Endpoint Information Register
USB_RAMINFO	RAM Information Register
USB_LINKINFO	Link Information Register
USB_VPLEN	VBUS Pulse Length Register
USB_HS_EOF1	High-Speed EOF 1 Register
USB_FS_EOF1	Full-Speed EOF 1 Register
USB_LS_EOF1	Low-Speed EOF 1 Register

Table 22-9: ADSP-BF60x USB Register List (Continued)

Name	Description
USB_SOFT_RST	Software Reset Register
USB_MPn_TXFUNCADDR	MPn Transmit Function Address Register
USB_MPn_TXHUBADDR	MPn Transmit Hub Address Register
USB_MPn_TXHUBPORT	MPn Transmit Hub Port Register
USB_MPn_RXFUNCADDR	MPn Receive Function Address Register
USB_MPn_RXHUBADDR	MPn Receive Hub Address Register
USB_MPn_RXHUBPORT	MPn Receive Hub Port Register
USB_EPn_TXMAXP	EPn Transmit Maximum Packet Length Register
USB_EP0_CSRn_H	EP0 Configuration and Status (Host) Register
USB_EP0_CSRn_P	EP0 Configuration and Status (Peripheral) Register
USB_EPn_TXCSR_H	EPn Transmit Configuration and Status (Host) Register
USB_EPn_TXCSR_P	EPn Transmit Configuration and Status (Peripheral) Register
USB_EPn_RXMAXP	EPn Receive Maximum Packet Length Register
USB_EPn_RXCSR_H	EPn Receive Configuration and Status (Host) Register
USB_EPn_RXCSR_P	EPn Receive Configuration and Status (Peripheral) Register
USB_EP0_CNTn	EP0 Number of Received Bytes Register
USB_EPn_RXCNT	EPn Number of Bytes Received Register
USB_EPn_TXTYPE	EPn Transmit Type Register
USB_EP0_TYPEn	EP0 Connection Type Register
USB_EP0_NAKLIMITn	EP0 NAK Limit Register
USB_EPn_TXINTERVAL	EPn Transmit Polling Interval Register
USB_EPn_RXTYPE	EPn Receive Type Register
USB_EPn_RXINTERVAL	EPn Receive Polling Interval Register
USB_EP0_CFGDATAn	EP0 Configuration Information Register

Table 22-9: ADSP-BF60x USB Register List (Continued)

Name	Description
USB_DMA_IRQ	DMA Interrupt Register
USB_DMAn_CTL	DMA Channel n Control Register
USB_DMAn_ADDR	DMA Channel n Address Register
USB_DMAn_CNT	DMA Channel n Count Register
USB_RQPKTCNTn	EPn Request Packet Count Register
USB_CT_UCH	Chirp Timeout Register
USB_CT_HHSRTN	Host High Speed Return to Normal Register
USB_CT_HSBT	High Speed Timeout Register
USB_LPM_ATTR	LPM Attribute Register
USB_LPM_CTL	LPM Control Register
USB_LPM_IEN	LPM Interrupt Enable Register
USB_LPM_IRQ	LPM Interrupt Status Register
USB_LPM_FADDR	LPM Function Address Register
USB_VBUS_CTL	VBUS Control Register
USB_BAT_CHG	Battery Charging Control Register
USB_PHY_CTL	PHY Control Register
USB_PLL_OSC	PLL and Oscillator Control Register

Function Address Register

The USB_FADDR register contains the device address used in peripheral mode. The processor writes this register with the address received through a SET_ADDRESS command from the host.

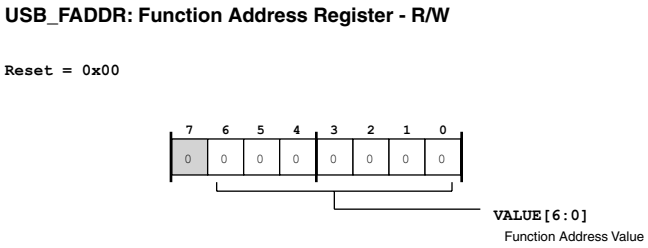


Figure 22-35: USB_FADDR Register Diagram

Table 22-10: USB_FADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Function Address Value. The USB_FADDR.VALUE bits contain the address of the peripheral part of the transaction.

Power and Device Control Register

The USB_POWER register controls suspend and resume signaling and controls some operational aspects of the USB controller.

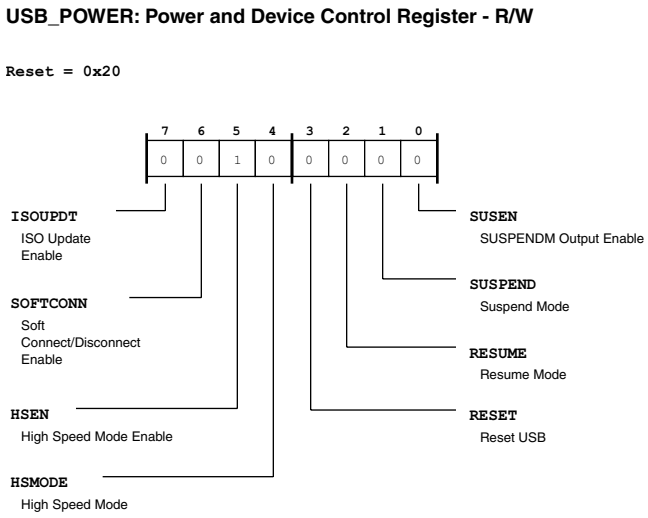


Figure 22-36: USB_POWER Register Diagram

Table 22-11: USB_POWER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	ISOUPDT	ISO Update Enable. The USB_POWER.ISOUPDT bit directs the USB controller to waits for an SOF token from the time TXPKTRDY is set before sending the packet. If an IN token is received before an SOF token, the USB controller send a zero length data packet. This USB_POWER.ISOUPDT bit only affects endpoints performing isochronous transfers. This bit only is valid in peripheral mode (USB_DEV_CTL.HOSTMODE = 0).	
		0	Disable ISO Update
		1	Enable ISO Update
6 (R/W)	SOFTCONN	Soft Connect/Disconnect Enable. In peripheral mode, the D+/- lines default to disconnected. Setting this bit will enable the D+/- termination resistors. This bit is automatically set when the DevCtl.Session bit is written with '1'. The USB_POWER.SOFTCONN bit enables USB controller soft connect/disconnect, enabling the termination resistors for USB_DP (Data +) and USB_DM (Data -) pins. When disabled, these pins are three-stated. Note that USB_POWER.SOFTCONN only is valid in peripheral mode (USB_DEV_CTL.HOSTMODE = 0).	
		0	Disable Soft Connect/Disconnect
		1	Enable Soft Connect/Disconnect
5 (R/W)	HSEN	High Speed Mode Enable. The USB_POWER.HSEN bit enables USB controller negotiation for high speed when the device is reset by the hub/host. If disabled, the USB controller only operates in full-speed mode.	
		0	Disable Negotiation for HS Mode
		1	Enable Negotiation for HS Mode
4 (R/NW)	HSMODE	High Speed Mode. The USB_POWER.HSMODE bit indicates whether or not the USB controller successfully negotiated high-speed mode during a USB controller reset. In peripheral mode (USB_DEV_CTL.HOSTMODE = 0), this bit has valid data when the USB controller completes reset. In host mode (USB_DEV_CTL.HOSTMODE = 1), this bit has valid data when the USB_IRQ.RSTBABBLE bit is cleared, remaining valid for the duration of the session.	
		0	Full Speed Mode (HS fail during reset)
		1	High Speed Mode (HS success during reset)

Table 22-11: USB_POWER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	RESET	Reset USB. The USB_POWER.RESET bit indicates (in both host and peripheral modes) that the USB controller has detected that reset signaling is present on the bus. In peripheral mode (USB_DEV_CTL.HOSTMODE = 0), this bit is read only, but in host mode (USB_DEV_CTL.HOSTMODE = 1), this bit is read/write, permitting the processor core to set the bit and initiate a USB controller reset.	
		0	No Reset
		1	Reset USB
2 (R/W)	RESUME	Resume Mode. The USB_POWER.RESUME bit directs the USB controller to generate resume signaling when the function is in suspend mode (USB_POWER.SUSPEND =1). The processor core should clear this bit after 10 ms (a maximum of 15 ms) to end resume signaling. When the USB controller is in host mode (USB_DEV_CTL.HOSTMODE = 1), the USB controller automatically sets the USB_POWER.RESUME bit when resume signaling from the target is detected while the USB controller is suspended.	
		0	Disable Resume Signaling
		1	Enable Resume Signaling
1 (R/W1S)	SUSPEND	Suspend Mode. When the USB controller is in host mode (USB_DEV_CTL.HOSTMODE = 1), the USB_POWER.SUSPEND bit enables suspend mode. When the USB controller is in peripheral mode (USB_DEV_CTL.HOSTMODE = 0), the USB controller sets the USB_POWER.SUSPEND bit on entry to suspend mode and clears the bit when the processor reads the USB_IRQ register. Note that the USB controller automatically clears this bit if the USB_POWER.RESUME bit is set.	
		0	Disable Suspend Mode (Host)
		1	Enable Suspend Mode (Host)
0 (R/W)	SUSEN	SUSPENDM Output Enable. The USB_POWER.SUSEN bit enables the SUSPENDM output (internal USB controller signal). When enabled, the SUSPENDM output signal is used by the USB controller PHY to power-down its drivers when the USB controller is not active.	
		0	Disable SUSPENDM Output
		1	Enable SUSPENDM Output

Transmit Interrupt Register

The USB_INTRTX register indicates which interrupts are currently active for endpoint 0 and the transmit (Tx) endpoints. Note that the USB controller automatically clears this register when it is read.

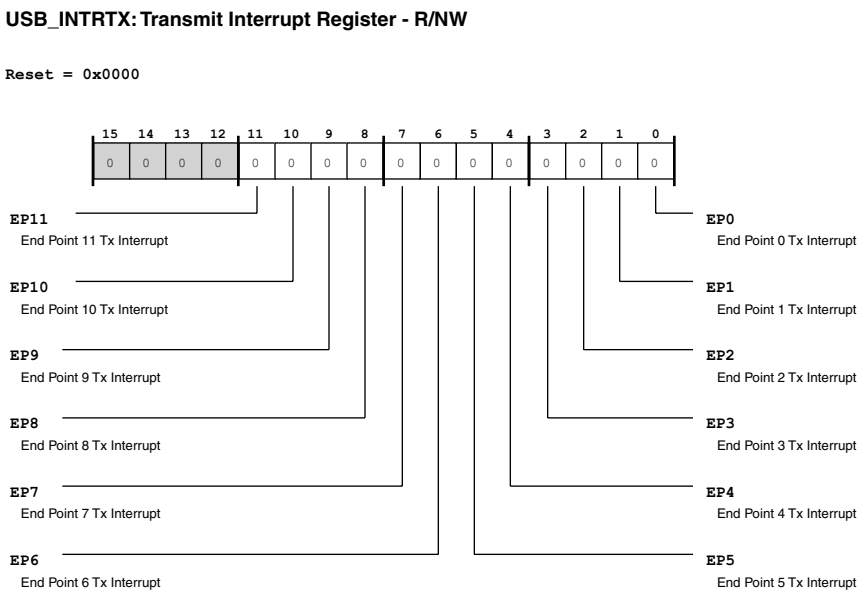


Figure 22-37: USB_INTRTX Register Diagram

Table 22-12: USB_INTRTX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (RC/NW)	EP11	End Point 11 Tx Interrupt. The USB_INTRTX.EP11 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
10 (RC/NW)	EP10	End Point 10 Tx Interrupt. The USB_INTRTX.EP10 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
9 (RC/NW)	EP9	End Point 9 Tx Interrupt. The USB_INTRTX.EP9 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
8 (RC/NW)	EP8	End Point 8 Tx Interrupt. The USB_INTRTX.EP8 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Table 22-12: USB_INTRTX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (RC/NW)	EP7	End Point 7 Tx Interrupt. The USB_INTRTX.EP7 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
6 (RC/NW)	EP6	End Point 6 Tx Interrupt. The USB_INTRTX.EP6 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
5 (RC/NW)	EP5	End Point 5 Tx Interrupt. The USB_INTRTX.EP5 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
4 (RC/NW)	EP4	End Point 4 Tx Interrupt. The USB_INTRTX.EP4 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
3 (RC/NW)	EP3	End Point 3 Tx Interrupt. The USB_INTRTX.EP3 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
2 (RC/NW)	EP2	End Point 2 Tx Interrupt. The USB_INTRTX.EP2 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
1 (RC/NW)	EP1	End Point 1 Tx Interrupt. The USB_INTRTX.EP1 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Table 22-12: USB_INTRTX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (RC/NW)	EP0	End Point 0 Tx Interrupt. The USB_INTRTX.EP0 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Receive Interrupt Register

The USB_INTRRX register indicates which interrupts are currently active for the receive (Rx) endpoints. Note that the USB controller automatically clears this register when it is read.

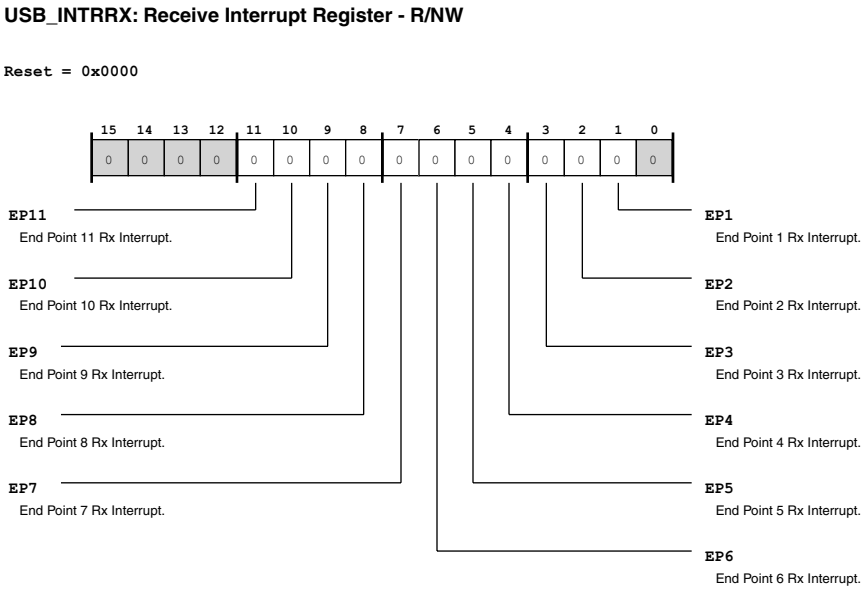


Figure 22-38: USB_INTRRX Register Diagram

Table 22-13: USB_INTRRX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (RC/NW)	EP11	End Point 11 Rx Interrupt.. The USB_INTRRX.EP11 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Table 22-13: USB_INTRRX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (RC/NW)	EP10	End Point 10 Rx Interrupt.. The USB_INTRRX.EP10 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
9 (RC/NW)	EP9	End Point 9 Rx Interrupt.. The USB_INTRRX.EP9 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
8 (RC/NW)	EP8	End Point 8 Rx Interrupt.. The USB_INTRRX.EP8 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
7 (RC/NW)	EP7	End Point 7 Rx Interrupt.. The USB_INTRRX.EP7 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
6 (RC/NW)	EP6	End Point 6 Rx Interrupt.. The USB_INTRRX.EP6 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
5 (RC/NW)	EP5	End Point 5 Rx Interrupt.. The USB_INTRRX.EP5 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
4 (RC/NW)	EP4	End Point 4 Rx Interrupt.. The USB_INTRRX.EP4 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Table 22-13: USB_INTRRX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (RC/NW)	EP3	End Point 3 Rx Interrupt.. The USB_INTRRX.EP3 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
2 (RC/NW)	EP2	End Point 2 Rx Interrupt.. The USB_INTRRX.EP2 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
1 (RC/NW)	EP1	End Point 1 Rx Interrupt.. The USB_INTRRX.EP1 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Transmit Interrupt Enable Register

The USB_INTRTXE register enables interrupts for endpoint 0 and the transmit (Tx) endpoints. Enabling an interrupt in this register directs the USB controller to generate an interrupt if the corresponding interrupt pending bit in the USB_INTRTX register is set.

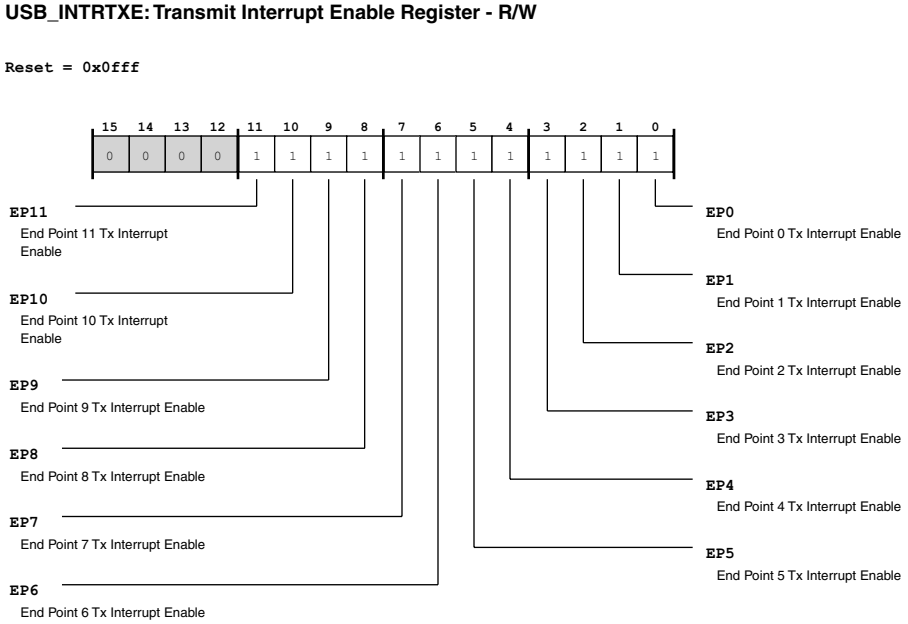


Figure 22-39: USB_INTRTXE Register Diagram

Table 22-14: USB_INTRTXE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	EP11	End Point 11 Tx Interrupt Enable. The USB_INTRTXE.EP11 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
10 (R/W)	EP10	End Point 10 Tx Interrupt Enable. The USB_INTRTXE.EP10 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
9 (R/W)	EP9	End Point 9 Tx Interrupt Enable. The USB_INTRTXE.EP9 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
8 (R/W)	EP8	End Point 8 Tx Interrupt Enable. The USB_INTRTXE.EP8 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
7 (R/W)	EP7	End Point 7 Tx Interrupt Enable. The USB_INTRTXE.EP7 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
6 (R/W)	EP6	End Point 6 Tx Interrupt Enable. The USB_INTRTXE.EP6 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
5 (R/W)	EP5	End Point 5 Tx Interrupt Enable. The USB_INTRTXE.EP5 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
4 (R/W)	EP4	End Point 4 Tx Interrupt Enable. The USB_INTRTXE.EP4 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
3 (R/W)	EP3	End Point 3 Tx Interrupt Enable. The USB_INTRTXE.EP3 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 22-14: USB_INTRTXE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	EP2	End Point 2 Tx Interrupt Enable. The USB_INTRTXE.EP2 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
1 (R/W)	EP1	End Point 1 Tx Interrupt Enable. The USB_INTRTXE.EP1 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
0 (R/W)	EP0	End Point 0 Tx Interrupt Enable. The USB_INTRTXE.EP0 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Receive Interrupt Enable Register

The USB_INTRRXE register enables interrupts for the receive (Rx) endpoints. Enabling an interrupt in this register directs the USB controller to generate an interrupt if the corresponding interrupt pending bit in the USB_INTRRX register is set.

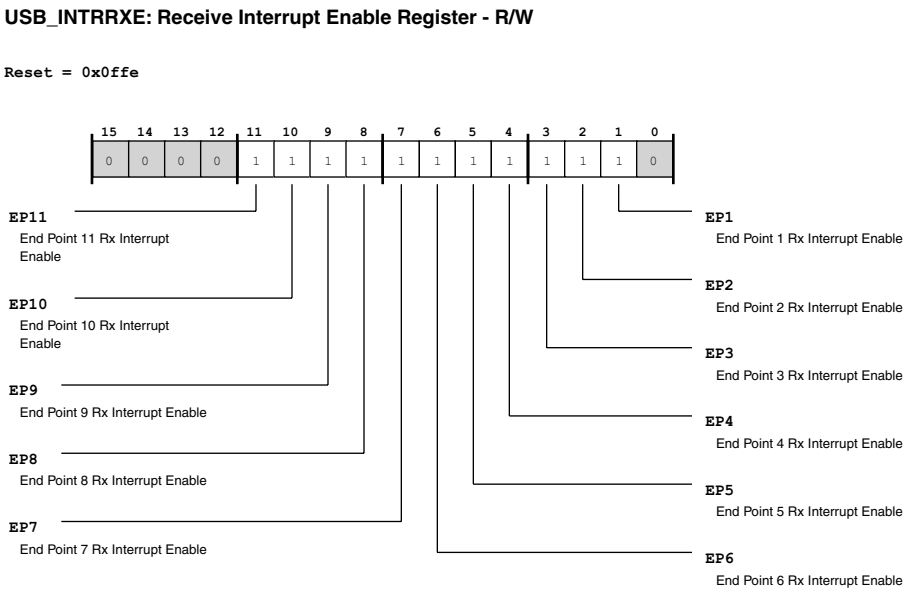


Figure 22-40: USB_INTRRXE Register Diagram

Table 22-15: USB_INTRRXE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	EP11	End Point 11 Rx Interrupt Enable. The USB_INTRRXE.EP11 bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
10 (R/W)	EP10	End Point 10 Rx Interrupt Enable. The USB_INTRRXE.EP10 bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
9 (R/W)	EP9	End Point 9 Rx Interrupt Enable. The USB_INTRRXE.EP9 bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
8 (R/W)	EP8	End Point 8 Rx Interrupt Enable. The USB_INTRRXE.EP8 bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
7 (R/W)	EP7	End Point 7 Rx Interrupt Enable. The USB_INTRRXE.EP7 bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
6 (R/W)	EP6	End Point 6 Rx Interrupt Enable. The USB_INTRRXE.EP6 bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
5 (R/W)	EP5	End Point 5 Rx Interrupt Enable. The USB_INTRRXE.EP5 bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
4 (R/W)	EP4	End Point 4 Rx Interrupt Enable. The USB_INTRRXE.EP4 bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
3 (R/W)	EP3	End Point 3 Rx Interrupt Enable. The USB_INTRRXE.EP3 bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 22-15: USB_INTRRXE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	EP2	End Point 2 Rx Interrupt Enable. The USB_INTRRXE.EP2 bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
1 (R/W)	EP1	End Point 1 Rx Interrupt Enable. The USB_INTRRXE.EP1 bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Common Interrupts Register

The USB_IRQ register indicates which interrupts are currently active for USB controller system sources. Note that the USB controller automatically clears this register when it is read.

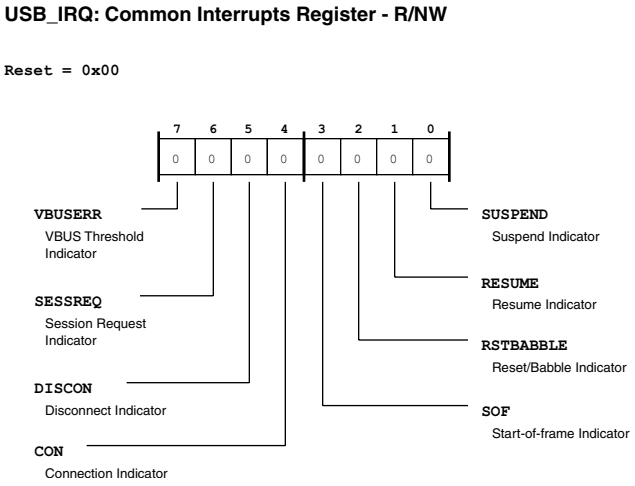


Figure 22-41: USB_IRQ Register Diagram

Table 22-16: USB_IRQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (RC/NW)	VBUSERR	VBUS Threshold Indicator. The USB_IRQ.VBUSERR bit indicates whether the USB controller has detected that the VBUS is below the VBUS valid threshold. This bit is valid only when the USB controller is an A device. Note that the USB_IRQ.VBUSERR bit and the USB_VBUS_CTL.DRVINT bit share an interrupt source line.	
		0	No Interrupt
		1	Interrupt Pending

Table 22-16: USB_IRQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (RC/NW)	SESSREQ	Session Request Indicator. The USB_IRQ.SESSREQ bit indicates whether the USB controller has detected a session request signal. This bit is valid only when the USB controller is an A device.	
		0	No Interrupt
		1	Interrupt Pending
5 (RC/NW)	DISCON	Disconnect Indicator. The USB_IRQ.DISCON bit indicates whether the USB controller has detected a device disconnect (host mode) or has detected a session end (peripheral mode).	
		0	No Interrupt
		1	Interrupt Pending
4 (RC/NW)	CON	Connection Indicator. The USB_IRQ.CON bit indicates whether the USB controller has detected a device connection. This bit is valid only in host mode.	
		0	No Interrupt
		1	Interrupt Pending
3 (RC/NW)	SOF	Start-of-frame Indicator. The USB_IRQ.SOF bit indicates whether the USB controller has detected a start of frame.	
		0	No Interrupt
		1	Interrupt Pending
2 (RC/NW)	RSTBABBLE	Reset/Babble Indicator. The USB_IRQ.RSTBABBLE bit indicates whether the USB controller has detected reset signalling on the bus. In host mode, the USB controller also indicates when the USB controller detects babble. Note that the USB_IRQ.RSTBABBLE bit is only active after the first SOF has been sent.	
		0	No Interrupt
		1	Interrupt Pending
1 (RC/NW)	RESUME	Resume Indicator. The USB_IRQ.RESUME bit indicates whether the USB controller has detected resume signaling on the bus while the USB controller is in suspend mode.	
		0	No Interrupt
		1	Interrupt Pending
0 (RC/NW)	SUSPEND	Suspend Indicator. The USB_IRQ.SUSPEND bit indicates whether the USB controller has detected suspend signalling on the bus. This bit is valid only in peripheral mode.	
		0	No Interrupt
		1	Interrupt Pending

Common Interrupts Enable Register

The USB_IEN register enables interrupts for USB controller system sources. Enabling an interrupt in this register directs the USB controller to generate an interrupt if the corresponding interrupt pending bit in the USB_IRQ register is set.

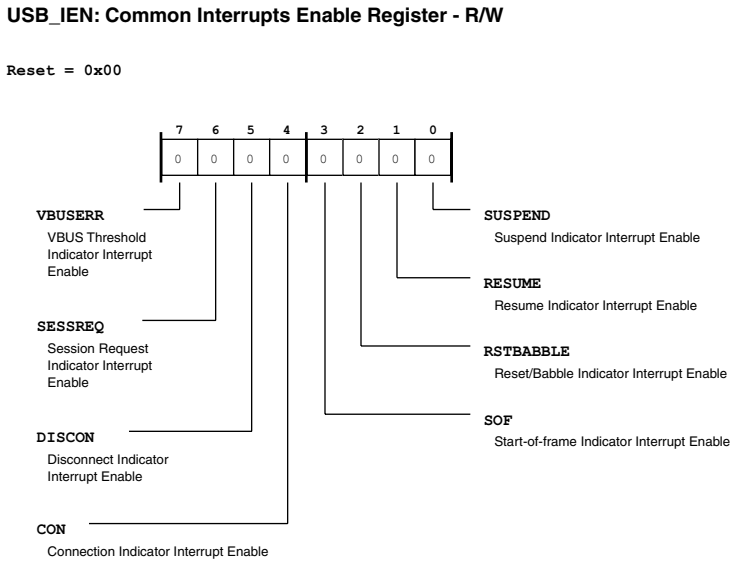


Figure 22-42: USB_IEN Register Diagram

Table 22-17: USB_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	VBUSERR	VBUS Threshold Indicator Interrupt Enable. The USB_IEN.VBUSERR bit enables the USB_IRQ.VBUSERR interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
6 (R/W)	SESSREQ	Session Request Indicator Interrupt Enable. The USB_IEN.SESSREQ bit enables the USB_IRQ.SESSREQ interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
5 (R/W)	DISCON	Disconnect Indicator Interrupt Enable. The USB_IEN.DISCON bit enables the USB_IRQ.DISCON interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
4 (R/W)	CON	Connection Indicator Interrupt Enable. The USB_IEN.CON bit enables the USB_IRQ.CON interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 22-17: USB_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	SOF	Start-of-frame Indicator Interrupt Enable. The USB_IEN.SOF bit enables the USB_IRQ.SOF interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
2 (R/W)	RSTBABBLE	Reset/Babble Indicator Interrupt Enable. The USB_IEN.RSTBABBLE bit enables the USB_IRQ.RSTBABBLE interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
1 (R/W)	RESUME	Resume Indicator Interrupt Enable. The USB_IEN.RESUME bit enables the USB_IRQ.RESUME interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
0 (R/W)	SUSPEND	Suspend Indicator Interrupt Enable. The USB_IEN.SUSPEND bit enables the USB_IRQ.SUSPEND interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt

Frame Number Register

The USB_FRAME register contains the frame number of the last received frame. The data in this register has bit 10 as the MSB and bit 0 as the LSB.

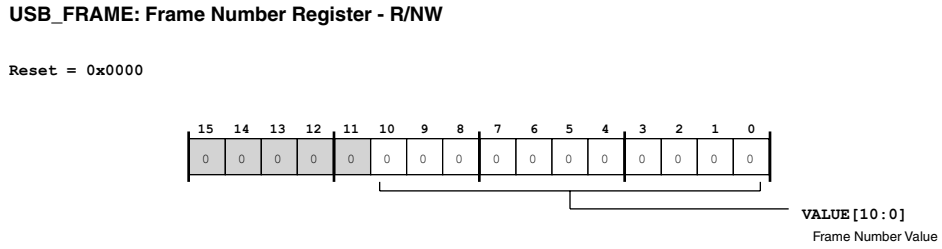


Figure 22-43: USB_FRAME Register Diagram

Table 22-18: USB_FRAME Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/NW)	VALUE	Frame Number Value. The USB_FRAME.VALUE bits contains the frame number of the last received frame. The data in this field has bit 10 as the MSB and bit 0 as the LSB.

Index Register

The USB_INDEX register contains an index value for mirrored addressing of USB controller endpoint control and status registers.

There is one set of registers, but they are mirrored at two address locations if the endpoint is selected by the USB_INDEX register. An endpoint's register set only appears in the indexed location if the USB_INDEX register is written with that endpoint number. You can read/write an endpoint's register in either the directly mapped location which is always visible, or in the indexed location which is only visible if the USB_INDEX register is written with the endpoint number. The USB_INDEX register and indexed address locations only affect address decoding. For example, loading a 0 into the USB_INDEX register selects endpoint 0 access.

The USB_INDEX register can be used for indexed access of the directly mapped control/status registers from USB controller address offset 0x100-0x1FF. For products supporting the dynamic FIFO size feature, the endpoint Tx/Rx size and address registers always use the USB_INDEX register, there is no direct mapping for these endpoint specific registers. The multipoint USB_MPn_TXFUNCADDR, USB_MPn_TXHUBADDR, USB_MPn_TXHUBPORT, USB_MPn_RXFUNCADDR, USB_MPn_RXHUBADDR, and USB_MPn_RXHUBPORT register only have direct mapping, no indexed mapping.

Before accessing an endpoint's control/status registers using the indexed range, write the endpoint number to the USB_INDEX register to ensure that the correct control/status registers appear in the indexed range of the memory map.

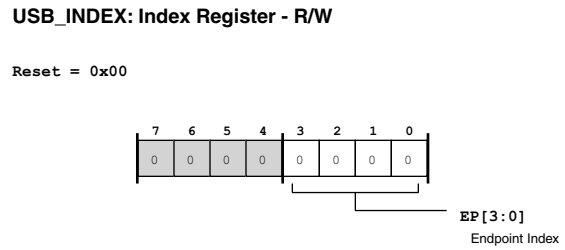


Figure 22-44: USB_INDEX Register Diagram

Table 22-19: USB_INDEX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	EP	Endpoint Index. The USB_INDEX.EP bits selects mirrored access for an endpoints indexed control and status registers. Valid values for this bit field are 0-11.

Testmode Register

The USB_TESTMODE register places the USB controller into test mode state and can also put the USB controller into one of the test modes for high-speed operation. For more information about these modes, see the USB 2.0 specification.

Note that the USB_TESTMODE register is not used in normal operation. Only one of the test mode (bits 0-6) selection bits may be set at a time.

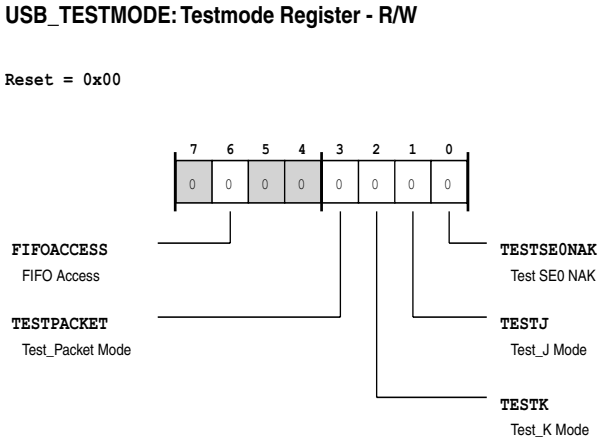


Figure 22-45: USB_TESTMODE Register Diagram

Table 22-20: USB_TESTMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W1A)	FIFOACCESS	FIFO Access. The USB_TESTMODE bit directs the USB controller to transfer the packet in the endpoint 0 Tx FIFO to the endpoint 0 Rx FIFO. The bit is cleared automatically.
3 (R/W)	TESTPACKET	Test_Packet Mode. The USB_TESTMODE.TESTPACKET bit selects Test_Packet test mode, which applies only when the USB controller in high speed mode. In this mode, the USB controller repetitively transmits on the bus a 53-byte test packet, whose form is defined in the USB 2.0 Specification, Section 7.1.20. Note that the test packet has a fixed format and must be loaded into the endpoint 0 FIFO before this test mode is entered.
2 (R/W)	TESTK	Test_K Mode. The USB_TESTMODE.TESTK bit selects Test_K test mode. In this mode, the USB controller transmits a continuous K on the bus.
1 (R/W)	TESTJ	Test_J Mode. The USB_TESTMODE.TESTJ bit selects Test_J test mode. In this mode, the USB controller transmits a continuous J on the bus.

Table 22-20: USB_TESTMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	TESTSE0NAK	Test SE0 NAK. The USB_TESTMODE.TESTSE0NAK bit selects Test_SE0_NAK test mode, which applies only when the USB controller in high speed mode. In this mode, the USB controller remains in high-speed mode, but responds to any valid IN token with a NAK.

FIFO Byte (8-Bit) Register

Writes to the USB_FIFOBn register go to the endpoint Tx FIFO and reads from the USB_FIFOBn register come from the endpoint Rx FIFO. The USB_FIFOBn, USB_FIFOHn, and USB_FIFOn registers are one and the same. These registers exist at the same address. Typically, programs should load and unload the FIFO using word (USB_FIFOn register) writes and reads, which are more efficient. Only if the USB packet is a non-word (4-byte) size should the program use a half-word (USB_FIFOHn register) or byte (USB_FIFOBn register) read or write at the end when loading or unloading the FIFO.

Note that (for correct USB controller operation) programs should not mix byte, half-word, or word accesses, except for the last few bytes if the size of the packet is odd (not a multiple of the size they were using).

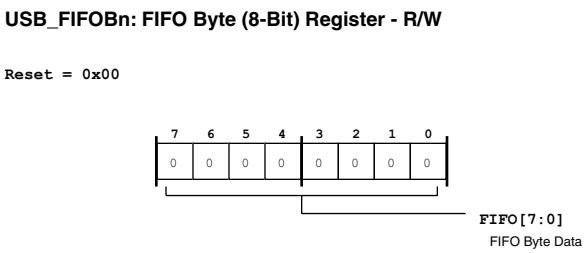


Figure 22-46: USB_FIFOBn Register Diagram

Table 22-21: USB_FIFOBn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FIFO	FIFO Byte Data. The USB_FIFOBn.FIFO bits provide byte access to the USB Tx and Rx endpoint FIFOs.

FIFO Half-Word (16-Bit) Register

Writes to the USB_FIFOHn register go to the endpoint Tx FIFO and reads from the USB_FIFOHn register come from the endpoint Rx FIFO. The USB_FIFOBn, USB_FIFOHn, and USB_FIFOn registers are one and the same. These registers exist at the same address. Typically, programs should load and unload the FIFO

using word (USB_FIFOn register) writes and reads, which are more efficient. Only if the USB packet is a non-word (4-byte) size should the program use a half-word (USB_FIFOHn register) or byte (USB_FIFOBn register) read or write at the end when loading or unloading the FIFO.

Note that (for correct USB controller operation) programs should not mix byte, half-word, or word accesses, except for the last few bytes if the size of the packet is odd (not a multiple of the size they were using).

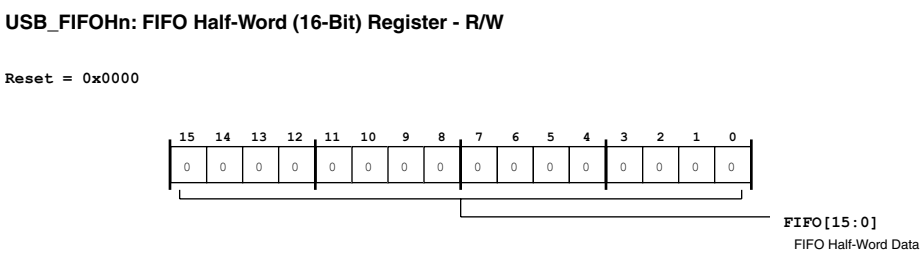


Figure 22-47: USB_FIFOHn Register Diagram

Table 22-22: USB_FIFOHn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	FIFO	FIFO Half-Word Data. The USB_FIFOHn.FIFO bits provide half-word access to the USB Tx and Rx endpoint FIFOs.

FIFO Word (32-Bit) Register

Writes to the USB_FIFOn register go to the endpoint Tx FIFO and reads from the USB_FIFOn register come from the endpoint Rx FIFO. The USB_FIFOBn, USB_FIFOHn, and USB_FIFOn registers are one and the same. These registers exist at the same address. Typically, programs should load and unload the FIFO using word (USB_FIFOn register) writes and reads, which are more efficient. Only if the USB packet is a non-word (4-byte) size should the program use a half-word (USB_FIFOHn register) or byte (USB_FIFOBn register) read or write at the end when loading or unloading the FIFO.

Note that (for correct USB controller operation) programs should not mix byte, half-word, or word accesses, except for the last few bytes if the size of the packet is odd (not a multiple of the size they were using).

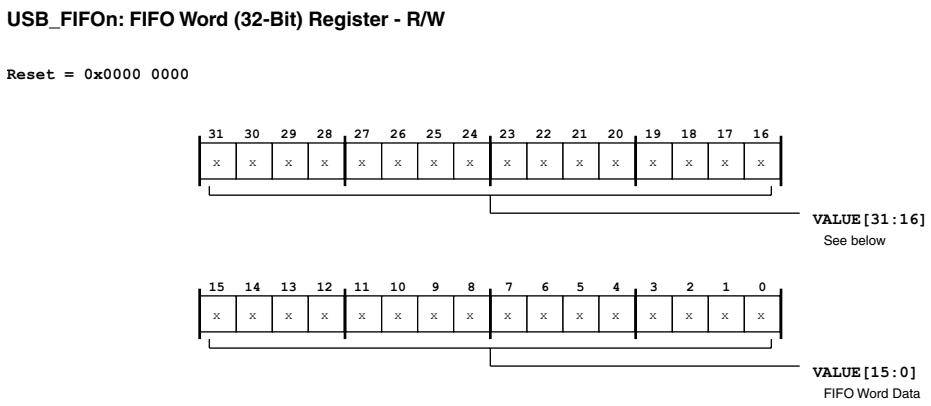


Figure 22-48: USB_FIFOn Register Diagram

Table 22-23: USB_FIFOn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	FIFO Word Data. The USB_FIFOn.VALUE bits provide word access to the USB Tx and Rx endpoint FIFOs.

Device Control Register

The USB_DEV_CTL register selects whether the USB controller is operating in peripheral mode or in host mode and is used for controlling and monitoring the VBUS line.

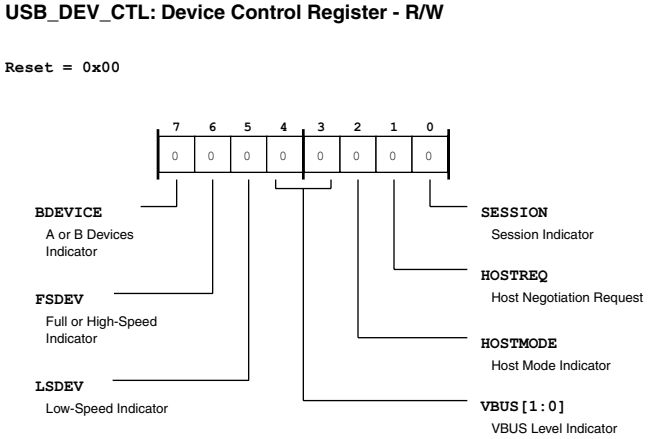


Figure 22-49: USB_DEV_CTL Register Diagram

Table 22-24: USB_DEV_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	BDEVICE	A or B Devices Indicator. The USB_DEV_CTL.BDEVICE bit indicates whether the USB controller is operating as the A device or the B device. This bit is only valid while a session is in progress.	
		0	A Device Detected
		1	B Device Detected
6 (R/NW)	FSDEV	Full or High-Speed Indicator. The USB_DEV_CTL.FSDEV bit is set when a full-speed or high-speed device is detected being connected to the port. High speed devices are distinguished from full-speed by checking for high-speed chirps when the device detects a USB controller reset. This bit is only valid in host mode.	
		0	Not Detected
		1	Full or High Speed Detected
5 (R/NW)	LSDEV	Low-Speed Indicator. The USB_DEV_CTL.LSDEV bit is set when a low-speed device is detected being connected to the port. This bit is only valid in host mode.	
		0	Not Detected
		1	Low Speed Detected
4:3 (R/NW)	VBUS	VBUS Level Indicator. The USB_DEV_CTL.VBUS bits indicated the current VBUS level.	
		0	Below SessionEnd
		1	Above SessionEnd, below AValid
		2	Above AValid, below VBUSValid
		3	Above VBUSValid
2 (R/NW)	HOSTMODE	Host Mode Indicator. The USB_DEV_CTL.HOSTMODE bit is set when the USB controller is acting as a host.	
		0	Peripheral Mode
		1	Host Mode
1 (R/W)	HOSTREQ	Host Negotiation Request. When the USB_DEV_CTL.HOSTREQ bit is set, the USB controller initiates the host negotiation when Suspend mode is entered. This bit is cleared when host negotiation is completed. The USB_DEV_CTL.HOSTREQ bit applies when the USB controller is operating as a B device only.	
		0	No Request
		1	Place Request

Table 22-24: USB_DEV_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	SESSION	Session Indicator. When operating as an A device, the USB_DEV_CTL.SESSION is set or cleared by the processor core to start or end a session. When operating as a B device, the USB_DEV_CTL.SESSION bit is set or cleared by the USB controller when a session starts or ends. This bit is also set by the processor core to initiate the session request protocol. When the USB controller is in Suspend mode, the bit may be cleared by the processor core to perform a software disconnect.	
		0	Not Detected
		1	Detected Session

Transmit FIFO Size Register

The USB_TXFIFOSZ register defines the maximum amount of data that can be transferred through the selected transmit endpoint in a single frame. When setting this value, you must consider the constraints placed by the USB specification on packet sizes for bulk, interrupt and isochronous transactions in full-speed operations. This register provides indexed access to the FIFO (packet) size selection for each Tx endpoint (except endpoint 0).

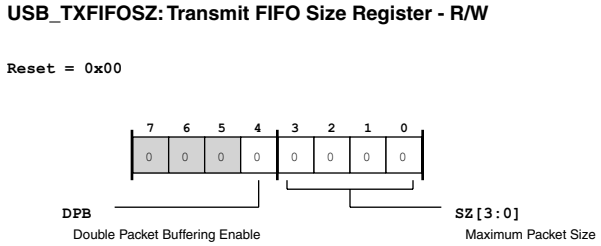


Figure 22-50: USB_TXFIFOSZ Register Diagram

Table 22-25: USB_TXFIFOSZ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	DPB	Double Packet Buffering Enable. The USB_TXFIFOSZ.DPB bit enables double packet buffering, doubling the FIFO (packet) size selected with the USB_TXFIFOSZ.SZ field.	
		0	Single Packet Buffering
		1	Double Packet Buffering

Table 22-25: USB_TXFIFOSZ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/W)	SZ	Maximum Packet Size. The USB_TXFIFOSZ.SZ bits select the maximum FIFO (packet) size according to the formula: FIFOSZ= $2^{(SZ+3)}$. If the USB_TXFIFOSZ.DPB is cleared, the FIFO size is FIFOSZ from this formula. If the USB_TXFIFOSZ.DPB is set, the FIFO is twice this size. For each enumeration value, the enumerations descriptions show the packet size (PktSz=), the FIFO size if DPB=0 (DPB0=), and the FIFO size if DPB=1 (DPB1=); these values are in bytes.	
		0	PktSz=8, DPB0=8, DPB1=16
		1	PktSz=16, DPB0=16, DPB1=32
		2	PktSz=32, DPB0=32, DPB1=64
		3	PktSz=64, DPB0=64, DPB1=128
		4	PktSz=128, DPB0=128, DPB1=256
		5	PktSz=256, DPB0=256, DPB1=512
		6	PktSz=512, DPB0=512, DPB1=1024
		7	PktSz=1024, DPB0=1024, DPB1=2048
		8	PktSz=2048, DPB0=2048, DPB1=4096
		9	PktSz=4096, DPB0=4096, DPB1=8192

Receive FIFO Size Register

The USB_RXFIFOSZ register defines the maximum amount of data that can be transferred through the selected receive endpoint in a single frame. When setting this value, you must consider the constraints placed by the USB specification on packet sizes for bulk, interrupt and isochronous transactions in full-speed operations. This register provides indexed access to the FIFO (packet) size selection for each Rx endpoint (except endpoint 0).

Note that a value greater than the maximum allowed of 1023 for full-speed USB controller operation produces unpredictable results.

Also note that the value written to this register should match the programmed maximum individual packet size (MaxPktSize) of the standard endpoint descriptor for the associated endpoint (see Universal Serial Bus Specification Revision 2.0, Chapter 9). A mismatch could cause unexpected results. The total amount of data represented by the value written to this register must not exceed the Rx FIFO size, and should not exceed half the FIFO size if double-buffering is required.

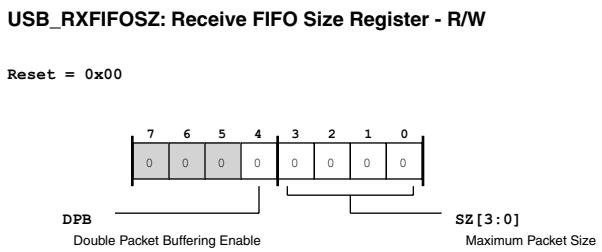


Figure 22-51: USB_RXFIFOSZ Register Diagram

Table 22-26: USB_RXFIFOSZ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	DPB	Double Packet Buffering Enable. The USB_RXFIFOSZ.DPB bit enables double packet buffering, doubling the FIFO (packet) size selected with the USB_RXFIFOSZ.SZ field.	
		0	Single Packet Buffering
		1	Double Packet Buffering
3:0 (R/W)	SZ	Maximum Packet Size. The USB_RXFIFOSZ.SZ bits select the maximum FIFO (packet) size according to the formula: FIFOSZ= $2^{(SZ+3)}$. If the USB_RXFIFOSZ.DPB is cleared, the FIFO size is FIFOSZ from this formula. If the USB_RXFIFOSZ.DPB is set, the FIFO is twice this size. For each enumeration value, the enumerations descriptions show the packet size (PktSz=), the FIFO size if DPB=0 (DPB0=), and the FIFO size if DPB=1 (DPB1=); these values are in bytes.	
		0	PktSz=8, DPB0=8, DPB1=16
		1	PktSz=16, DPB0=16, DPB1=32
		2	PktSz=32, DPB0=32, DPB1=64
		3	PktSz=64, DPB0=64, DPB1=128
		4	PktSz=128, DPB0=128, DPB1=256
		5	PktSz=256, DPB0=256, DPB1=512
		6	PktSz=512, DPB0=512, DPB1=1024
		7	PktSz=1024, DPB0=1024, DPB1=2048
		8	PktSz=2048, DPB0=2048, DPB1=4096
		9	PktSz=4096, DPB0=4096, DPB1=8192

Transmit FIFO Address Register

The USB_TXFIFOADDR sets the start address for the selected Tx FIFO for endpoints 1-11. There is one transmit FIFO address register for each endpoint, except endpoint 0. The USB_TXFIFOADDR register is

indexed and selected by the USB_INDEX register. Note that the endpoint 0 FIFO has a fixed 64-byte size and is always located at address 0.

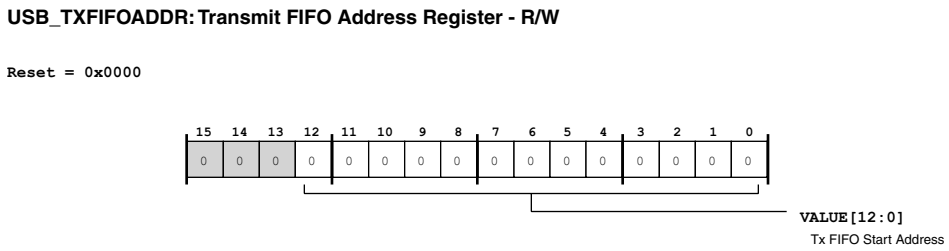


Figure 22-52: USB_TXFIFOADDR Register Diagram

Table 22-27: USB_TXFIFOADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:0 (R/W)	VALUE	Tx FIFO Start Address. The USB_TXFIFOADDR.VALUE bits hold the start address of the selected endpoint FIFO (selected with the USB_INDEX register) in units of 8 bytes, according to the formula: FIFO address = USB_TXFIFOADDR.VALUE * 8

Receive FIFO Address Register

The USB_RXFIFOADDR sets the start address for the selected Rx FIFO for endpoints 1-11. There is one receive FIFO address register for each endpoint, except endpoint 0. The USB_RXFIFOADDR register is indexed and selected by the USB_INDEX register. Note that the endpoint 0 FIFO has a fixed 64-byte size and is always located at address 0.

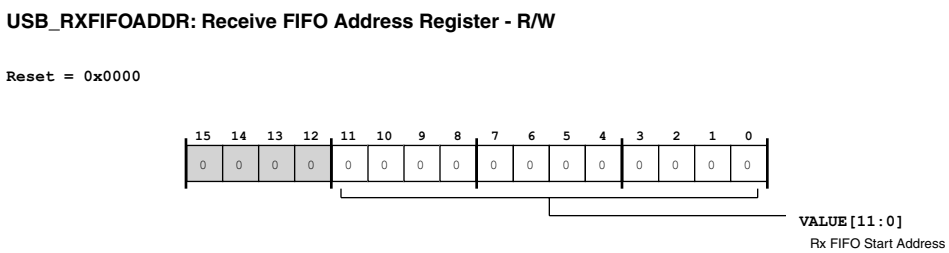


Figure 22-53: USB_RXFIFOADDR Register Diagram

Table 22-28: USB_RXFIFOADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/W)	VALUE	Rx FIFO Start Address. The USB_RXFIFOADDR.VALUE bits hold the start address of the selected endpoint FIFO (selected with the USB_INDEX register) in units of 8 bytes, according to the formula: FIFO address = USB_RXFIFOADDR.VALUE * 8

Endpoint Information Register

The USB_EPINFO register allows read-back of the number of Tx and Rx endpoints available

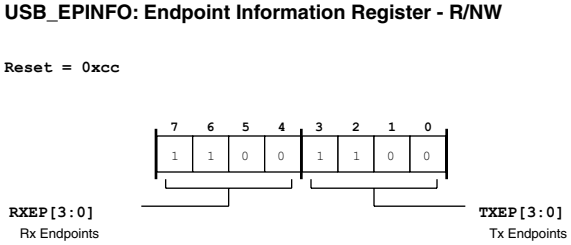


Figure 22-54: USB_EPINFO Register Diagram

Table 22-29: USB_EPINFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	RXEP	Rx Endpoints. The USB_EPINFO.RXEP bits indicate the number of receive endpoints. excluding EP0.
3:0 (R/NW)	TXEP	Tx Endpoints. The USB_EPINFO.TXEP bits indicate the number of transmit endpoints, excluding EP0.

RAM Information Register

The USB_RAMINFO register provides information about the width of the USB controller RAM.

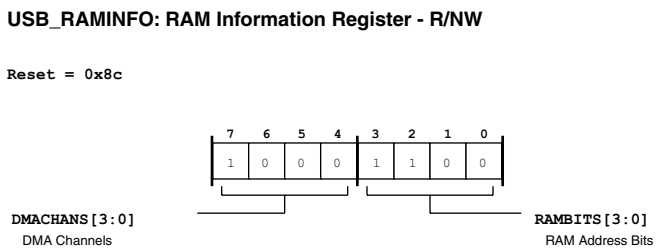


Figure 22-55: USB_RAMINFO Register Diagram

Table 22-30: USB_RAMINFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	DMACHANS	DMA Channels. The USB_RAMINFO.DMACHANS bits indicate the number of DMA channels.
3:0 (R/NW)	RAMBITS	RAM Address Bits. The USB_RAMINFO.RAMBITS bits indicate the number of RAM address bits. The USB controller FIFO RAM is 32-bits wide. The number of bytes in the FIFO RAM may be calculated from the formula: RAM_bytes = $2^{(RAM\_Bits+2)}$

Link Information Register

The USB_LINKINFO register specifies the PHY-related delays.

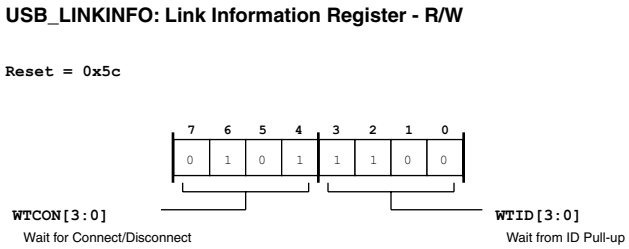


Figure 22-56: USB_LINKINFO Register Diagram

Table 22-31: USB_LINKINFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/W)	WTCON	Wait for Connect/Disconnect. The USB_LINKINFO.WTCON bits set the wait to be applied to allow for the users connect or disconnect filter in units of 533.3ns. The default settings corresponds to 2.667us

Table 22-31: USB_LINKINFO Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	WTID	Wait from ID Pull-up. The USB_LINKINFO.WTID bits set the delay to be applied from IDPULLUP being asserted to IDDIG being considered valid in units of 4.3690ms. The default corresponds to 52.43ms UTMI+ spec says 50ms min. OTG spec does not have timing requirements (it doesn't assume a programmable pull-up that is only sampled during session start). Micro-USB cable spec says that the ID pin is greater than 10 Ohms when shorted and less than 100k Ohms when open.

VBUS Pulse Length Register

The USB_VPLEN register defines the duration of the VBUS pulsing charge for SRP initiation.

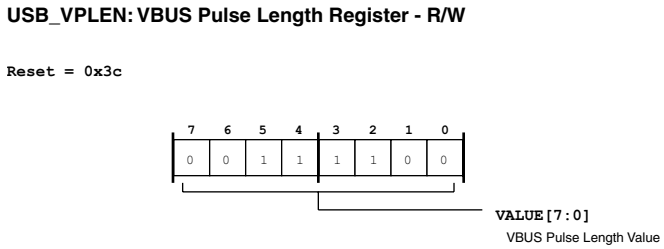


Figure 22-57: USB_VPLEN Register Diagram

Table 22-32: USB_VPLEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	VBUS Pulse Length Value. The USB_VPLEN.VALUE bits sets the duration of the VBUS pulsing charge in units of 546.1us. The default setting corresponds to 32.77ms. Note that VBUS pulsing was removed in the OTG specification v2.0, section 5.1.4.

High-Speed EOF 1 Register

The USB_HS_EOF1 register defines the minimum time gap allowed between the start of the last transaction and the end of frame for high-speed transactions.

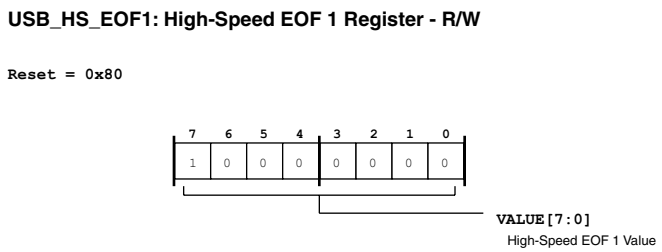


Figure 22-58: USB_HS_EOF1 Register Diagram

Table 22-33: USB_HS_EOF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	High-Speed EOF 1 Value. The USB_HS_EOF1.VALUE sets the time before end of frame to stop beginning new transactions (in units of 133.3ns) for high-speed transactions. The default setting corresponds to 17.07us.

Full-Speed EOF 1 Register

The USB_FS_EOF1 register defines the minimum time gap allowed between the start of the last transaction and the end of frame for full-speed transactions.

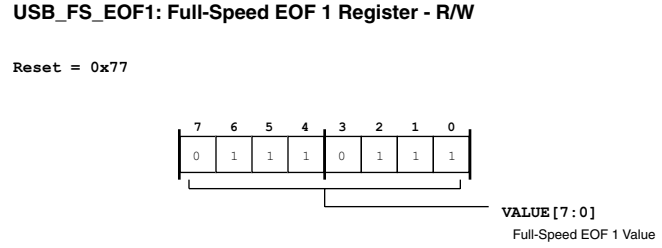


Figure 22-59: USB_FS_EOF1 Register Diagram

Table 22-34: USB_FS_EOF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Full-Speed EOF 1 Value. The USB_FS_EOF1.VALUE bits set the time before end of frame to stop beginning new transactions (in units of 533.3ns) for full-speed transactions. The default setting corresponds to 63.46us.

Low-Speed EOF 1 Register

The USB_LS_EOF1 register defines the minimum time gap allowed between the start of the last transaction and the end of frame for low-speed transactions.

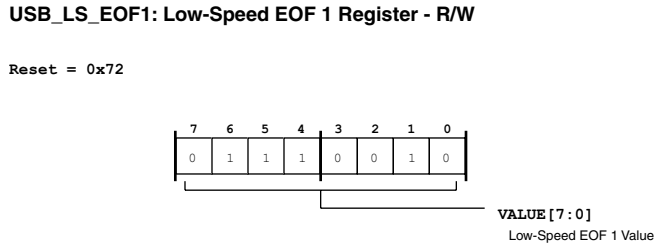


Figure 22-60: USB_LS_EOF1 Register Diagram

Table 22-35: USB_LS_EOF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Low-Speed EOF 1 Value. The USB_LS_EOF1.VALUE bits set the time before end of frame to stop beginning new transactions (in units of 1.067us) for low-speed transactions. The default setting corresponds to 121.6us.

Software Reset Register

The USB_SOFT_RST register provides reset controls for the USB controller CLK domain and XCLK domain. The USB controller PHY operates in the controller's XCLK domain, and the USB controller interface to the processor core operates in the controller's CLK domain. Note that for correct operation, both of the reset control bits (USB_SOFT_RST.RST and USB_SOFT_RST.RSTX) should always be asserted simultaneously.

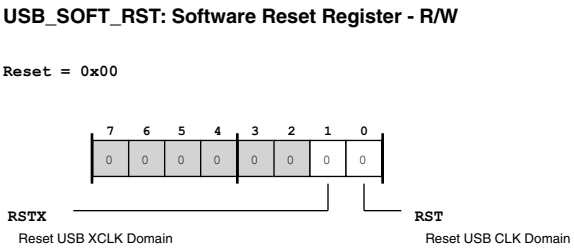


Figure 22-61: USB_SOFT_RST Register Diagram

Table 22-36: USB_SOFT_RST Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1A)	RSTX	Reset USB XCLK Domain. The USB_SOFT_RST.RSTX bit resets logic in the USB XCLK domain. This bit is self clearing. Note that this bit should always be asserted simultaneously with the USB_SOFT_RST.RST bit.	
		0	No Reset
		1	Reset USB XCLK Domain
0 (R/W1A)	RST	Reset USB CLK Domain. The USB_SOFT_RST.RST bit resets logic in the USB CLK domain. This bit is self clearing. Note that this bit should always be asserted simultaneously with the USB_SOFT_RST.RSTX bit.	
		0	No Reset
		1	Reset USB CLK Domain

MPn Transmit Function Address Register

The USB_MPn_TXFUNCADDR register specifies the transmit endpoint's target address in host mode. This register is not used in device mode. Note that the USB_MPn_TXFUNCADDR register must be setup for EP0. (The USB_MPn_RXFUNCADDR register does not exist for EP0.)

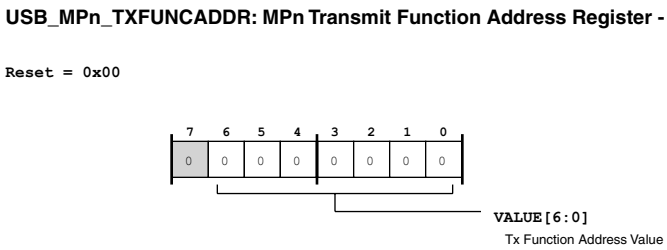


Figure 22-62: USB_MPn_TXFUNCADDR Register Diagram

Table 22-37: USB_MPn_TXFUNCADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Tx Function Address Value. The USB_MPn_TXFUNCADDR.VALUE bits hold the address of the target device for this endpoint.

MPn Transmit Hub Address Register

The USB_MPn_TXHUBADDR register specifies the hub address of the endpoint in host mode. This register is not used in device mode. Note that this register only needs to be programmed when a full-speed or low-speed device is connected to a high-speed hub, which carries out the necessary transaction translation. Also note that EP0 only uses the USB_MPn_TXHUBADDR register. (The USB_MPn_RXHUBADDR register does not exist for EP0.)

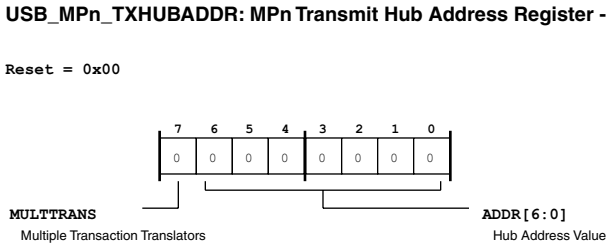


Figure 22-63: USB_MPn_TXHUBADDR Register Diagram

Table 22-38: USB_MPn_TXHUBADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	MULTTRANS	Multiple Transaction Translators. The USB_MPn_TXHUBADDR.MULTTRANS bit should be set if the hub has multiple transaction translators.	
		0	Single Transaction Translator
		1	Multiple Transaction Translators
6:0 (R/W)	ADDR	Hub Address Value. The USB_MPn_TXHUBADDR.ADDR bits hold the address of the hub to which this device is connected.	

MPn Transmit Hub Port Register

The USB_MPn_TXHUBPORT register specifies the hub port for full-speed and low-speed endpoints in host mode. This register is not used in device mode. The USB_MPn_TXHUBPORT register lets the USB controller support SPLIT transactions. EP0 only uses the USB_MPn_TXHUBPORT register. (The USB_MPn_RXHUBPORT register does not exist for EP0.)

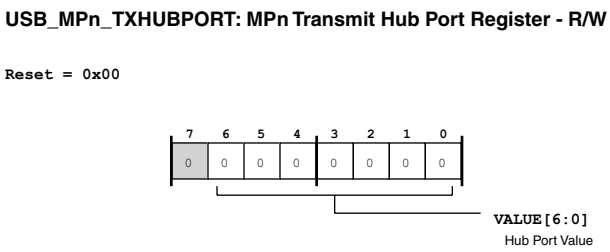


Figure 22-64: USB_MPn_TXHUBPORT Register Diagram

Table 22-39: USB_MPn_TXHUBPORT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Hub Port Value. The USB_MPn_TXHUBPORT.VALUE bits hold the hub port value of the target device for this endpoint.

MPn Receive Function Address Register

The USB_MPn_RXFUNCADDR register specifies the receive endpoint's target address in host mode. This register is not used in device mode. Note that the USB_MPn_RXFUNCADDR register does not exist for EP0.

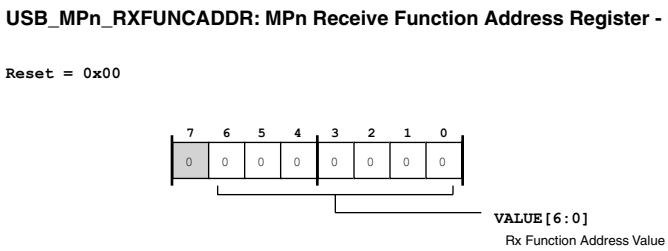


Figure 22-65: USB_MPn_RXFUNCADDR Register Diagram

Table 22-40: USB_MPn_RXFUNCADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Rx Function Address Value. The USB_MPn_RXFUNCADDR.VALUE bits hold the address of the target device for this endpoint.

MPn Receive Hub Address Register

The USB_MPn_RXHUBADDR register specifies the hub address of the endpoint in host mode. This register is not used in device mode. Note that this register only needs to be programmed when a full-speed or

low-speed device is connected to a high-speed hub, which carries out the necessary transaction translation. Note that the USB_MPn_RXHUBADDR register does not exist for EP0.

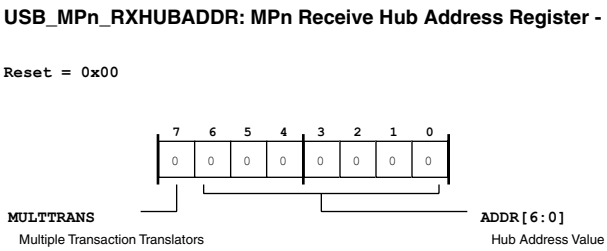


Figure 22-66: USB_MPn_RXHUBADDR Register Diagram

Table 22-41: USB_MPn_RXHUBADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	MULTTRANS	Multiple Transaction Translators. The USB_MPn_RXHUBADDR.MULTTRANS bit should be set if the hub has multiple transaction translators.	
		0	Single Transaction Translator
		1	Multiple Transaction Translators
6:0 (R/W)	ADDR	Hub Address Value. The USB_MPn_RXHUBADDR.ADDR bits hold the address of the hub to which this device is connected.	

MPn Receive Hub Port Register

The USB_MPn_RXHUBPORT register specifies the hub port for full-speed and low-speed endpoints in host mode. This register is not used in device mode. The USB_MPn_RXHUBPORT register lets the USB controller support SPLIT transactions. Note that the USB_MPn_RXHUBPORT register does not exist for EP0.

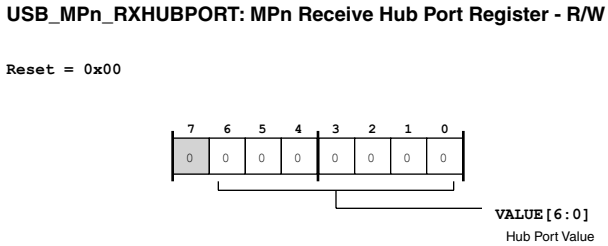


Figure 22-67: USB_MPn_RXHUBPORT Register Diagram

Table 22-42: USB_MPn_RXHUBPORT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Hub Port Value. The USB_MPn_RXHUBPORT.VALUE bits hold the hub port value of the target device for this endpoint.

EPn Transmit Maximum Packet Length Register

The USB_EPn_TXMAXP register defines the maximum amount of data that can be transferred through the selected transmit endpoint in a single frame. When setting this value, you must consider the constraints placed by the USB specification on packet sizes for bulk, interrupt and isochronous transactions in full-speed operations. The USB_EPn_TXMAXP register provides indexed access to the maximum packet length register for each Tx endpoint, except endpoint 0.

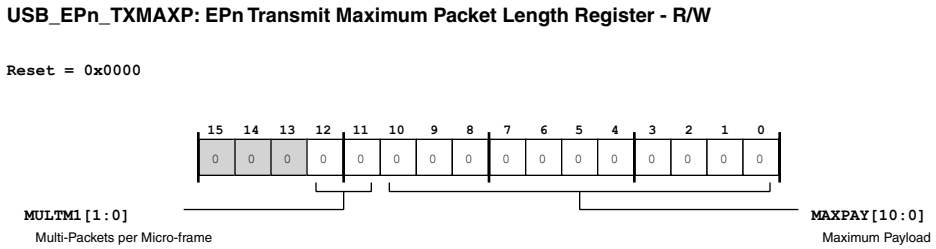


Figure 22-68: USB_EPn_TXMAXP Register Diagram

Table 22-43: USB_EPn_TXMAXP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:11 (R/W)	MULTM1	Multi-Packets per Micro-frame. The USB_EPn_TXMAXP.MULTM1 bits select the number of high-speed high-bandwidth isochronous or interrupt packets that may be transferred in a microframe. The valid number of packets per microframe is 1-3 which corresponds to settings 0-2. If this field is not zero, the USB controller splits the FIFO data into multiple packets less than or equal to the maximum payload size.
10:0 (R/W)	MAXPAY	Maximum Payload. The USB_EPn_TXMAXP.MAXPAY bits select the maximum number of bytes that may be transferred per transaction. This field can be up to 1024 but is subject to constraints by the USB specification based on endpoint mode and speed. This field should not exceed the FIFO size for the endpoint, or half the FIFO size if double buffering is used. This value should match the wMaxPacketSize field of the standard endpoint descriptor (USB 2.0 spec, section 9). The USB_EPn_TXMAXP.MAXPAY bits must be set to an even number of bytes for proper interrupt generation in DMA mode 1.

EP0 Configuration and Status (Host) Register

The USB_EP0_CSRn_H register provides control and status bits for endpoint 0 in host mode. Note that some bits may be set to clear automatically.

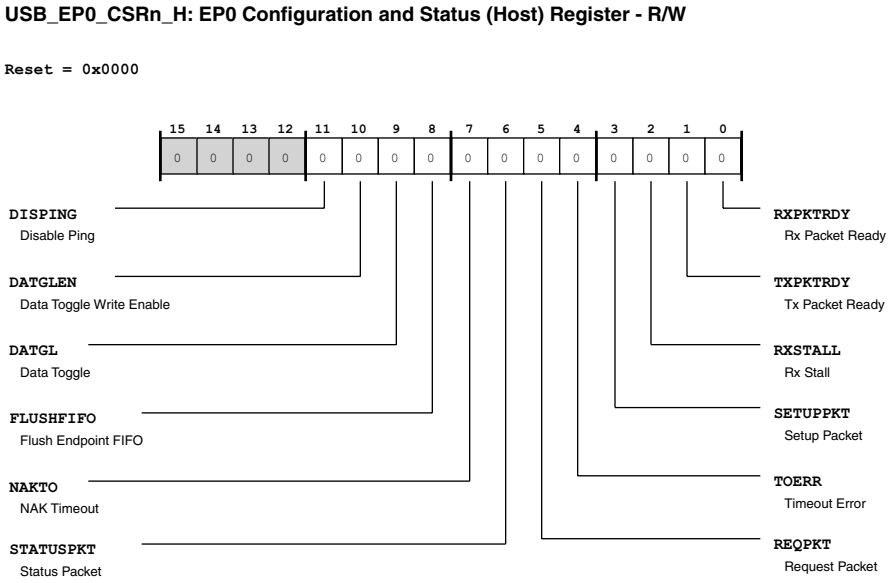


Figure 22-69: USB_EP0_CSRn_H Register Diagram

Table 22-44: USB_EP0_CSRn_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	DISPING	Disable Ping. The USB_EP0_CSRn_H.DISPING bit disables (in host mode) high-speed PING tokens for the data and status phases of a control transfer.	
		0	Issue PING tokens
		1	Do not issue PING
10 (R/W1A)	DATGLEN	Data Toggle Write Enable. The USB_EP0_CSRn_H.DATGLEN bit enables (in host mode) the USB controller to write the current state of the endpoint 0 USB_EP0_CSRn_H.DATGL bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL
9 (R/W)	DATGL	Data Toggle. The USB_EP0_CSRn_H.DATGL bit indicates (in host mode) the current state of the endpoint 0 data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is Set
		1	DATA1 is Set

Table 22-44: USB_EP0_CSRn_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1A)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EP0_CSRn_H.FLUSHFIFO bit directs (in host mode) the USB controller to flush data from the endpoint 0 FIFO and clear the USB_EP0_CSRn_H.TXPKTRDY and USB_EP0_CSRn_H.RXPKTRDY bits. The USB_EP0_CSRn_H.FLUSHFIFO bit should only be set if the USB_EP0_CSRn_H.TXPKTRDY and USB_EP0_CSRn_H.RXPKTRDY bits are set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
7 (R/W0C)	NAKTO	NAK Timeout. The USB_EP0_CSRn_H.NAKTO bit indicates (in host mode) when endpoint 0 is halted following the receipt of NAK responses for longer than the time set by the USB_EP0_NAKLIMITn register. The processor core should clear this bit to allow the endpoint to continue.	
		0	No Status
		1	Endpoint Halted (NAK Timeout)
6 (R/W)	STATUSPKT	Status Packet. The USB_EP0_CSRn_H.STATUSPKT bit directs (in host mode) the USB controller to perform a status stage transaction. This bit is set at the same time as the USB_EP0_CSRn_H.TXPKTRDY USB_EP0_CSRn_H.RXPKTRDY. Setting this bit ensures that the data toggle is set to 1 so that a DATA1 packet is used for the status stage transaction.	
		0	No Request
		1	Request Status Transaction
5 (R/W)	REQPKT	Request Packet. The USB_EP0_CSRn_H.REQPKT bit directs (in host mode) the USB controller to request an IN transaction. This bit is cleared when USB_EP0_CSRn_H.RXPKTRDY is set.	
		0	No Request
		1	Send IN Tokens to Device
4 (R/W0C)	TOERR	Timeout Error. The USB_EP0_CSRn_H.TOERR bit indicates (in host mode) when three attempts have been made to perform a transaction with no response from the peripheral. The processor core should clear this bit. An interrupt is generated when this bit is set.	
		0	No Status
		1	Timeout Error

Table 22-44: USB_EP0_CSRn_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W1S)	SETUPPKT	Setup Packet. The USB_EP0_CSRn_H.SETUPPKT bit directs (in host mode) the USB controller to send a SETUP token instead of an OUT token for the transaction. This bit is set at the same time as the USB_EP0_CSRn_H.TXPKTRDY bit is set.	
		0	No Request
		1	Send SETUP token
2 (R/W0C)	RXSTALL	Rx Stall. The USB_EP0_CSRn_H.RXSTALL bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
1 (R/W1S)	TXPKTRDY	Tx Packet Ready. The USB_EP0_CSRn_H.TXPKTRDY bit should be set (in host mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The USB_EP0_CSRn_H.RXPKTRDY is set (in host mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EP0 Configuration and Status (Peripheral) Register

The USB_EP0_CSRn_P register provides control and status bits for endpoint 0 in peripheral mode. Note that some bits may be set to clear automatically.

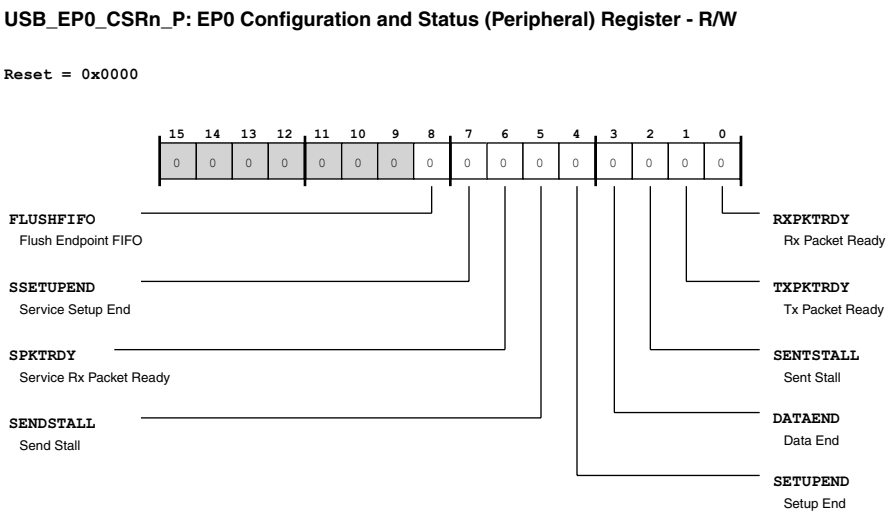


Figure 22-70: USB_EP0_CSRn_P Register Diagram

Table 22-45: USB_EP0_CSRn_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1A)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EP0_CSRn_P.FLUSHFIFO bit directs (in peripheral mode) the USB controller to flush data from the endpoint 0 FIFO and clear the USB_EP0_CSRn_P.TXPKTRDY and USB_EP0_CSRn_P.RXPKTRDY bits. The USB_EP0_CSRn_P.FLUSHFIFO bit should only be set if the USB_EP0_CSRn_P.TXPKTRDY and USB_EP0_CSRn_P.RXPKTRDY bits are set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
7 (R/W1A)	SSETUPEND	Service Setup End. The USB_EP0_CSRn_P.SSETUPEND bit is set (in peripheral mode) by the processor core to clear the USB_EP0_CSRn_P.SETUPEND. This bit is cleared automatically.	
		0	No Action
		1	Clear SETUPEND Bit
6 (R/W1A)	SPKTRDY	Service Rx Packet Ready. The USB_EP0_CSRn_P.SPKTRDY bit is set (in peripheral mode) by the processor core to clear the USB_EP0_CSRn_P.RXPKTRDY bit. This bit is cleared automatically.	
		0	No Action
		1	Clear RXPKTRDY Bit

Table 22-45: USB_EP0_CSRn_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	SENDSTALL	Send Stall. The USB_EP0_CSRn_P.SENDSTALL bit is set (in peripheral mode) by the processor core to terminate the current transaction. The STALL handshake is transmitted, then this bit automatically is cleared.	
		0	No Action
		1	Terminate Current Transaction
4 (R/NW)	SETUPEND	Setup End. The USB_EP0_CSRn_P.SETUPEND bit indicates (in peripheral mode) when a control transaction ends before the USB_EP0_CSRn_P.DATAEND bit is set. An interrupt is generated and the FIFO is flushed at this time. This bit is cleared when the processor core sets the USB_EP0_CSRn_P.SSETUPEND bit.	
		0	No Status
		1	Setup Ended before DATAEND
3 (R/W1S)	DATAEND	Data End. The USB_EP0_CSRn_P.DATAEND bit is set (in peripheral mode) by the processor core sets when the core: • Sets the USB_EP0_CSRn_P.TXPKTRDY bit for the last data packet. • Clears the USB_EP0_CSRn_P.RXPKTRDY bit after unloading the last data packet. • Sets the USB_EP0_CSRn_P.TXPKTRDY bit for a zero length data packet. The USB_EP0_CSRn_P.DATAEND bit is cleared automatically.	
		0	No Status
		1	Data End Condition
2 (R/W0C)	SENTSTALL	Sent Stall. The USB_EP0_CSRn_P.SENTSTALL bit is set (in peripheral mode) when a STALL handshake is transmitted. The processor core should clear this bit.	
		0	No Status
		1	Transmitted STALL Handshake
1 (R/W1S)	TXPKTRDY	Tx Packet Ready. The USB_EP0_CSRn_P.TXPKTRDY bit should be set (in peripheral mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	
		1	Set this bit after loading a data packet into the FIFO

Table 22-45: USB_EP0_CSRn_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The USB_EP0_CSRn_P.RXPKTRDY is set (in peripheral mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core clears this bit by setting the USB_EP0_CSRn_P.SPKTRDY bit.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EPn Transmit Configuration and Status (Host) Register

The USB_EPn_TXCSR_H register provides (in host mode) control and status bits for transfers through the currently selected transmit endpoint.

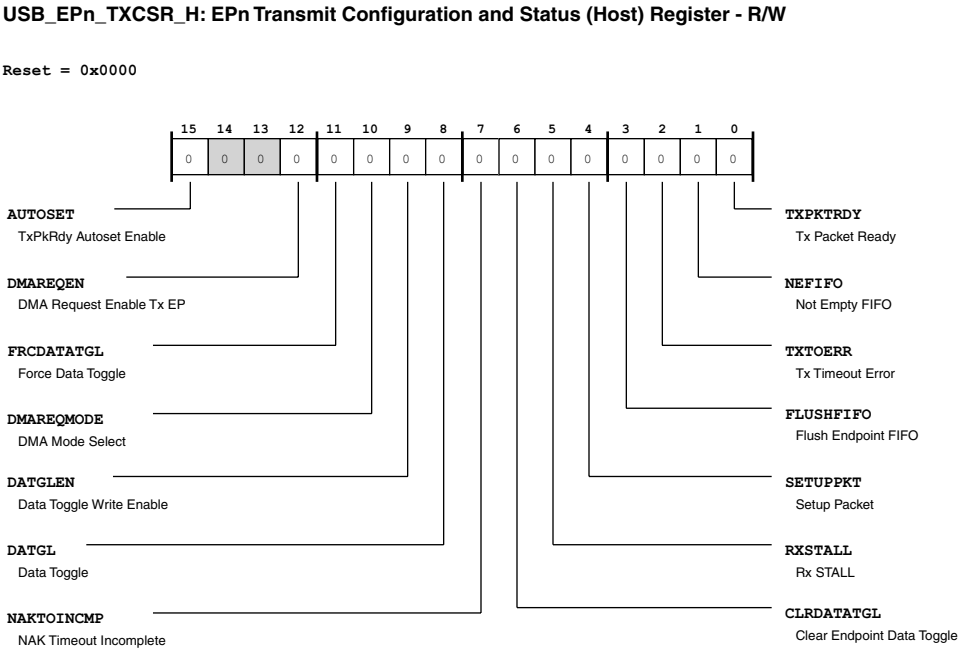


Figure 22-71: USB_EPn_TXCSR_H Register Diagram

Table 22-46: USB_EPn_TXCSR_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOSET	TxPkRdy Autoset Enable. The USB_EPn_TXCSR_H.AUTOSET bit enables (in host mode) automatic setting of the USB_EPn_TXCSR_H.TXPKTRDY bit when the maximum data packet size (USB_EPn_TXMAXP) is loaded into the transmit FIFO. The USB_EPn_TXMAXP value must be a word (4-byte) multiple. If a packet less than the maximum packet size is loaded, the USB_EPn_TXCSR_H.TXPKTRDY bit needs to be set manually. This USB_EPn_TXCSR_H.AUTOSET bit should not be set for high bandwidth endpoints (endpoints with USB_EPn_TXMAXP value greater than 1).	
		0	Disable Autoset
		1	Enable Autoset
12 (R/W)	DMAREQEN	DMA Request Enable Tx EP. The USB_EPn_TXCSR_H.DMAREQEN bit enables (in host mode) DMA requests for this transmit endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request
11 (R/W)	FRCDATATGL	Force Data Toggle. The USB_EPn_TXCSR_H.FRCDATATGL bit forces (in host mode) the endpoint data toggle to switch and clears the data packet from the FIFO, regardless of whether an ACK was received. This feature can be used by interrupt transmit endpoints that are used to communicate rate feedback for isochronous endpoints.	
		0	No Action
		1	Toggle Endpoint Data
10 (R/W)	DMAREQMODE	DMA Mode Select. The USB_EPn_TXCSR_H.DMAREQMODE bit selects (in host mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the USB_EPn_TXCSR_H.DMAREQEN bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
9 (R/W1A)	DATGLEN	Data Toggle Write Enable. The USB_EPn_TXCSR_H.DATGLEN bit enables (in host mode) the USB controller to write the current state of the endpoint USB_EPn_TXCSR_H.DATGL bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL

Table 22-46: USB_EPn_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	DATGL	Data Toggle. The USB_EPn_TXCSR_H.DATGL bit indicates (in host mode) the current state of the endpoint data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is set
		1	DATA1 is set
7 (R/W0C)	NAKTOINCMP	NAK Timeout Incomplete. The USB_EPn_TXCSR_H.NAKTOINCMP bit indicates (for bulk endpoints in host mode) when the transmit endpoint is halted following the receipt of NAK responses for longer than the time set in the USB_EPn_TXINTERVAL register. The processor should clear this bit, allowing the endpoint to continue. For high-bandwidth isochronous endpoints in host mode, this bit indicates when no response is received from the device to which the packet is being sent.	
		0	No Status
		1	NAK Timeout Over Maximum
6 (R/W1A)	CLRDATATGL	Clear Endpoint Data Toggle. The USB_EPn_TXCSR_H.CLRDATATGL bit is set (in host mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
5 (R/W0C)	RXSTALL	Rx STALL. The USB_EPn_TXCSR_H.RXSTALL bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
4 (R/W)	SETUPPKT	Setup Packet. The USB_EPn_TXCSR_H.SETUPPKT bit directs (in host mode) the USB controller to send a SETUP token instead of an OUT token for the transaction. This bit is set at the same time as the USB_EPn_TXCSR_H.TXPKTRDY bit is set.	
		0	No Request
		1	Send SETUP Token
3 (R/W1A)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EPn_TXCSR_H.FLUSHFIFO bit directs (in host mode) the USB controller to flush data from the endpoint FIFO and clear the USB_EPn_TXCSR_H.TXPKTRDY bit. The USB_EPn_TXCSR_H.FLUSHFIFO bit should only be set if the USB_EPn_TXCSR_H.TXPKTRDY bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush endpoint FIFO

Table 22-46: USB_EPn_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W0C)	TXTOERR	Tx Timeout Error. The USB_EPn_TXCSR_H.TXTOERR bit indicates (in host mode) when three attempts have been made to send a packet and no handshake packet has been received. The USB controller generates an interrupt for this condition, clears the USB_EPn_TXCSR_H.TXPKTRDY bit, and flushes the FIFO. The processor should clear this bit. Note that USB_EPn_TXCSR_H.TXTOERR is valid only when the endpoint is operating in bulk or interrupt mode.	
		0	No Status
		1	Tx Timeout Error
1 (R/NW)	NEFIFO	Not Empty FIFO. The USB_EPn_TXCSR_H.NEFIFO bit indicates (in host mode) when there is at least one packet in the transmit FIFO. This bit is cleared automatically when a data packet has been transmitted. If the endpoints transmit interrupt is enabled (in USB_INTRTXE), the USB controller generates an interrupt for this condition. Note that the USB_EPn_TXCSR_H.TXPKTRDY bit is also automatically cleared prior to loading a second packet into a double-buffered FIFO.	
		0	FIFO Empty
		1	FIFO Not Empty
0 (R/W1S)	TXPKTRDY	Tx Packet Ready. The USB_EPn_TXCSR_H.TXPKTRDY bit should be set (in host mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

EPn Transmit Configuration and Status (Peripheral) Register

The USB_EPn_TXCSR_P register provides (in peripheral mode) control and status bits for transfers through the currently selected transmit endpoint.

USB_EPn_TXCSR_P: EPn Transmit Configuration and Status (Peripheral) Register - R/W

Reset = 0x0000

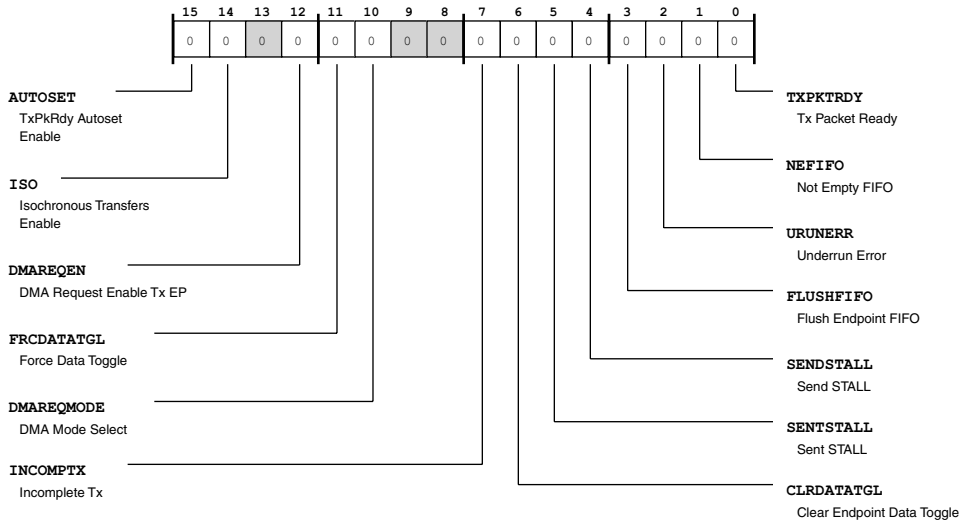


Figure 22-72: USB_EPn_TXCSR_P Register Diagram

Table 22-47: USB_EPn_TXCSR_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOSET	TxPkRdy Autoset Enable. The USB_EPn_TXCSR_P.AUTOSET bit enables (in peripheral mode) automatic setting of the USB_EPn_TXCSR_P.TXPKTRDY bit when the maximum data packet size (USB_EPn_TXMAXP) is loaded into the transmit FIFO. The USB_EPn_TXMAXP value must be a word (4-byte) multiple. If a packet less than the maximum packet size is loaded, the USB_EPn_TXCSR_P.TXPKTRDY bit needs to be set manually. This USB_EPn_TXCSR_P.AUTOSET bit should not be set for high bandwidth endpoints (endpoints with USB_EPn_TXMAXP value greater than 1).	
		0	Disable Autoset
		1	Enable Autoset
14 (R/W)	ISO	Isochronous Transfers Enable. The USB_EPn_TXCSR_P.ISO bit enables (in peripheral mode) the transmit endpoint for isochronous transfers. This bit should be disabled for bulk or interrupt endpoints.	
		0	Disable Tx EP Isochronous Transfers
		1	Enable Tx EP Isochronous Transfers
12 (R/W)	DMAREQEN	DMA Request Enable Tx EP. The USB_EPn_TXCSR_P.DMAREQEN bit enables (in peripheral mode) DMA requests for this transmit endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 22-47: USB_EPn_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	FRCDATATGL	Force Data Toggle. The USB_EPn_TXCSR_P.FRCDATATGL bit forces (in peripheral mode) the endpoint data toggle to switch and clears the data packet from the FIFO, regardless of whether an ACK was received. This feature can be used by interrupt transmit endpoints that are used to communicate rate feedback for isochronous endpoints.	
		0	No Action
		1	Toggle Endpoint Data
10 (R/W)	DMAREQMODE	DMA Mode Select. The USB_EPn_TXCSR_P.DMAREQMODE bit selects (in peripheral mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the USB_EPn_TXCSR_P.DMAREQEN bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
7 (R/W0C)	INCOMPTX	Incomplete Tx. The USB_EPn_TXCSR_P.INCOMPTX bit indicates (for high-bandwidth isochronous endpoints in peripheral mode) when a large packet has been split into two or three packets for transmission, but insufficient IN tokens have been received to send all parts.	
		0	No Status
		1	Incomplete Tx (Insufficient IN Tokens)
6 (R/W1A)	CLRDATATGL	Clear Endpoint Data Toggle. The USB_EPn_TXCSR_P.CLRDATATGL bit is set (in peripheral mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
5 (R/W0C)	SENTSTALL	Sent STALL. The USB_EPn_TXCSR_P.SENTSTALL bit indicates (in peripheral mode) when the USB controller transmits a STALL handshake. When this condition occurs, the USB controller flushes the FIFO and clears the USB_EPn_TXCSR_P.TXPKTRDY bit. The processor should clear this bit.	
		0	No Status
		1	STALL Handshake Transmitted
4 (R/W)	SENDSTALL	Send STALL. The USB_EPn_TXCSR_P.SENDSTALL bit (in peripheral mode) is set by the processor to issue a STALL handshake to an IN token. The processor clears this bit to terminate the stall condition. This bit has no effect for isochronous transfers.	
		0	No Request
		1	Request STALL Handshake Transmission

Table 22-47: USB_EPn_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W1A)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EPn_TXCSR_P.FLUSHFIFO bit directs (in peripheral mode) the USB controller to flush data from the endpoint FIFO and clear the USB_EPn_TXCSR_P.TXPKTRDY bit. The USB_EPn_TXCSR_P.FLUSHFIFO bit should only be set if the USB_EPn_TXCSR_P.TXPKTRDY bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush endpoint FIFO
2 (R/W0C)	URUNERR	Underrun Error. The USB_EPn_TXCSR_P.URUNERR bit indicates (in peripheral mode) when an IN token is received while the USB_EPn_TXCSR_P.TXPKTRDY bit is not set. The processor should clear this bit.	
		0	No Status
		1	Underrun Error
1 (R/NW)	NEFIFO	Not Empty FIFO. The USB_EPn_TXCSR_P.NEFIFO bit indicates (in peripheral mode) when there is at least one packet in the transmit FIFO. This bit is cleared automatically when a data packet has been transmitted. If the endpoints transmit interrupt is enabled (in USB_INTRTXE), the USB controller generates an interrupt for this condition. Note that the USB_EPn_TXCSR_P.TXPKTRDY bit is also automatically cleared prior to loading a second packet into a double-buffered FIFO.	
		0	FIFO Empty
		1	FIFO Not Empty
0 (R/W1S)	TXPKTRDY	Tx Packet Ready. The USB_EPn_TXCSR_P.TXPKTRDY bit should be set (in peripheral mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

EPn Receive Maximum Packet Length Register

The USB_EPn_RXMAXP register defines the maximum amount of data that can be transferred through the selected receive endpoint in a single frame.

Note that a value greater than the maximum allowed of 1023 for full-speed USB operation produces unpredictable results. Also note that the total amount of data represented by the value written to this register must not exceed the receive FIFO size, and should not exceed half the FIFO size if double-buffering is required.

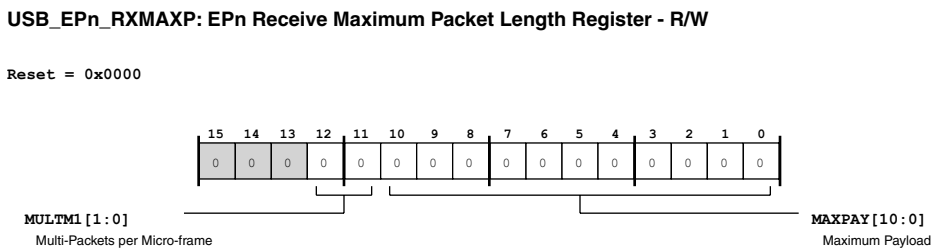


Figure 22-73: USB_EPn_RXMAXP Register Diagram

Table 22-48: USB_EPn_RXMAXP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:11 (R/W)	MULTM1	Multi-Packets per Micro-frame. The USB_EPn_RXMAXP.MULTM1 bits select the number of high-speed high-bandwidth isochronous or interrupt packets that may be transferred in a microframe. The valid number of packets per microframe is 1-3 which corresponds to settings 0-2. If this field is not zero, the USB controller combines multiple packets received within a microframe into a single packet in the FIFO.
10:0 (R/W)	MAXPAY	Maximum Payload. The USB_EPn_RXMAXP.MAXPAY bits select the maximum number of bytes that may be transferred per transaction. This field can be up to 1024 but is subject to constraints by the USB specification based on endpoint mode and speed. This field should not exceed the FIFO size for the endpoint, or half the FIFO size if double buffering is used. This value should match the wMaxPacketSize field of the standard endpoint descriptor (USB 2.0 spec, section 9). The USB_EPn_RXMAXP.MAXPAY bits must be set to an even number of bytes for proper interrupt generation in DMA mode 1.

EPn Receive Configuration and Status (Host) Register

The USB_EPn_RXCSR_H register provides (in host mode) control and status bits for transfers through the currently selected receive endpoint.

USB_EPn_RXCSR_H: EPn Receive Configuration and Status (Host) Register - R/W

Reset = 0x0000

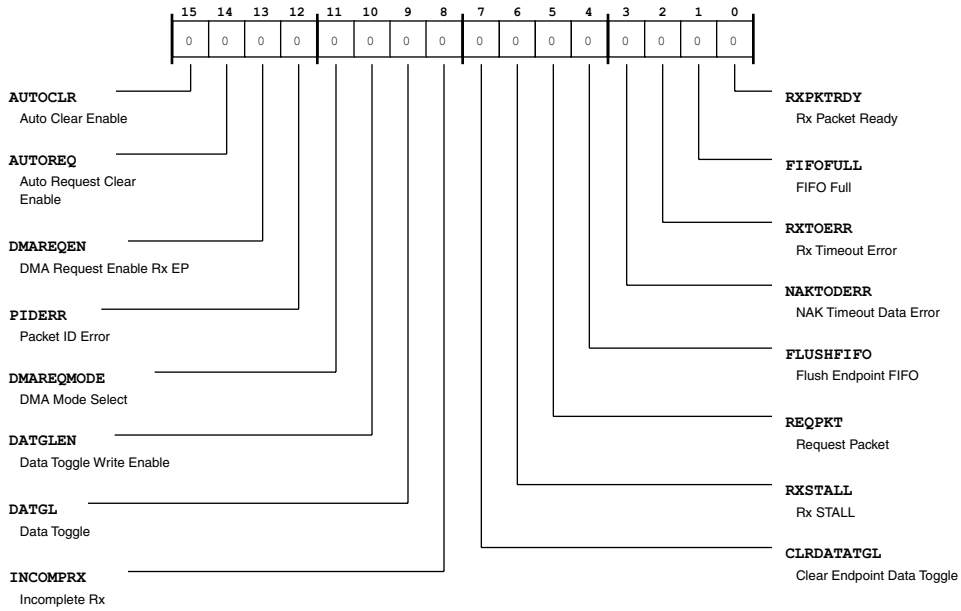


Figure 22-74: USB_EPn_RXCSR_H Register Diagram

Table 22-49: USB_EPn_RXCSR_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOCLR	Auto Clear Enable. The USB_EPn_RXCSR_H.AUTOCLR bit directs (in host mode) the USB controller to automatically clear the USB_EPn_RXCSR_H.RXPKTRDY bit when a packet of size USB_EPn_RXMAXP bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, the processor must clear USB_EPn_RXCSR_H.RXPKTRDY manually. When using the DMA to unload the receive FIFO, data is read from the receive FIFO in four byte chunks, regardless of the USB_EPn_RXMAXP value. The USB controller auto clears the USB_EPn_RXCSR_H.RXPKTRDY bit as follows. (In the following: Remainder=(RxMaxP/4), and PktSz-Clearing-RxPktRdy=Actual-Bytes-Read-Packet-Sizes-That-Clear-RxPktRdy.) • Remainder=0, Bytes-Read=RxMaxP, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2, RxMaxP-3 • Remainder=3, Bytes Read=RxMaxP+1, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2 • Remainder=2, Bytes Read=RxMaxP+2, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1 • Remainder=1, Bytes Read=RxMaxP+3, PktSz-Clearing-RxPktRdy=RxMaxP Note that the USB_EPn_RXCSR_H.AUTOCLR bit should not be set for high-bandwidth isochronous endpoints.	
		0	Disable Auto Clear
		1	Enable Auto Clear
14 (R/W)	AUTOREQ	Auto Request Clear Enable. The USB_EPn_RXCSR_H.AUTOREQ bit directs (in host mode) the USB controller to automatically clear the USB_EPn_RXCSR_H.REQPKT bit when USB_EPn_RXCSR_H.RXPKTRDY bit is cleared. This bit is automatically cleared when a short packet is received.	
		0	Disable Auto Request Clear
		1	Enable Auto Request Clear
13 (R/W)	DMAREQEN	DMA Request Enable Rx EP. The USB_EPn_RXCSR_H.DMAREQEN bit enables (in host mode) DMA requests for this receive endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request
12 (R/W0C)	PIDERR	Packet ID Error. The USB_EPn_RXCSR_H.PIDERR bit indicates (in host mode) when a PID error occurs for isochronous transactions. This bit is ignored in host mode for bulk or interrupt transactions.	
		0	No Status
		1	PID Error

Table 22-49: USB_EPn_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	DMAREQMODE	DMA Mode Select. The USB_EPn_RXCSR_H.DMAREQMODE bit selects (in host mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the USB_EPn_RXCSR_H.DMAREQEN bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
10 (R/W1A)	DATGLEN	Data Toggle Write Enable. The USB_EPn_RXCSR_H.DATGLEN bit enables (in host mode) the USB controller to write the current state of the endpoint USB_EPn_RXCSR_H.DATGL bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL
9 (R/W)	DATGL	Data Toggle. The USB_EPn_RXCSR_H.DATGL bit indicates (in host mode) the current state of the endpoint data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is Set
		1	DATA1 is Set
8 (R/W)	INCOMPRX	Incomplete Rx. The USB_EPn_RXCSR_H.INCOMPRX bit indicates (in host mode for high-bandwidth isochronous or interrupt transfers) when the received packet is incomplete because parts of the packet were not received. This bit is cleared when USB_EPn_RXCSR_H.RXPKTRDY is cleared. For all other modes, this bit is zero.	
		0	No Status
		1	Incomplete Rx
7 (R/W1A)	CLRDATATGL	Clear Endpoint Data Toggle. The USB_EPn_RXCSR_H.CLRDATATGL bit is set (in host mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
6 (R/W0C)	RXSTALL	Rx STALL. The USB_EPn_RXCSR_H.RXSTALL bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device

Table 22-49: USB_EPn_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	REQPKT	Request Packet. The USB_EPn_RXCSR_H.REQPKT bit directs (in host mode) the USB controller to request an IN transaction. This bit is cleared when USB_EPn_RXCSR_H.RXPKTRDY is set.	
		0	No Request
		1	Send IN Tokens to Device
4 (R/W1A)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EPn_RXCSR_H.FLUSHFIFO bit directs (in host mode) the USB controller to flush data from the endpoint FIFO and clear the USB_EPn_RXCSR_H.RXPKTRDY bit. The USB_EPn_RXCSR_H.FLUSHFIFO bit should only be set if the USB_EPn_RXCSR_H.RXPKTRDY bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
3 (R/W0C)	NAKTODERR	NAK Timeout Data Error. The USB_EPn_RXCSR_H.NAKTODERR bit indicates (in host mode for isochronous transfers) a NAK timeout data error when the USB_EPn_RXCSR_H.RXPKTRDY bit is set and the data packet has a CRC or bit-stuff error. This bit is cleared when the USB_EPn_RXCSR_H.RXPKTRDY bit is cleared. The USB_EPn_RXCSR_H.NAKTODERR bit indicates (in host mod for bulk transfers) when a receive endpoint is halted following the receipt of NAK responses greater than the limit set in the USB_EPn_RXINTERVAL register. The processor should clear this bit to allow the endpoint to continue. If double packet buffering is enabled, the USB_EPn_RXCSR_H.REQPKT bit should also be set in the same cycle as this bit is cleared.	
		0	No Status
		1	NAK Timeout Data Error
2 (R/W0C)	RXTOERR	Rx Timeout Error. The USB_EPn_RXCSR_H.RXTOERR bit indicates (in host mode) when three attempts have been made to receive a packet and no data packet has been received. The USB controller generates an interrupt for this condition. The processor should clear this bit. Note that USB_EPn_RXCSR_H.RXTOERR is valid only when the endpoint is operating in bulk or interrupt mode.	
		0	No Status
		1	Rx Timeout Error
1 (R/NW)	FIFOFULL	FIFO Full. The USB_EPn_RXCSR_H.FIFOFULL bit indicates (in host mode) when no more packets can be loaded into the receive FIFO.	
		0	No Status
		1	FIFO Full

Table 22-49: USB_EPn_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The USB_EPn_RXCSR_H.RXPKTRDY is set (in host mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EPn Receive Configuration and Status (Peripheral) Register

The USB_EPn_RXCSR_P register provides (in peripheral mode) control and status bits for transfers through the currently selected receive endpoint.

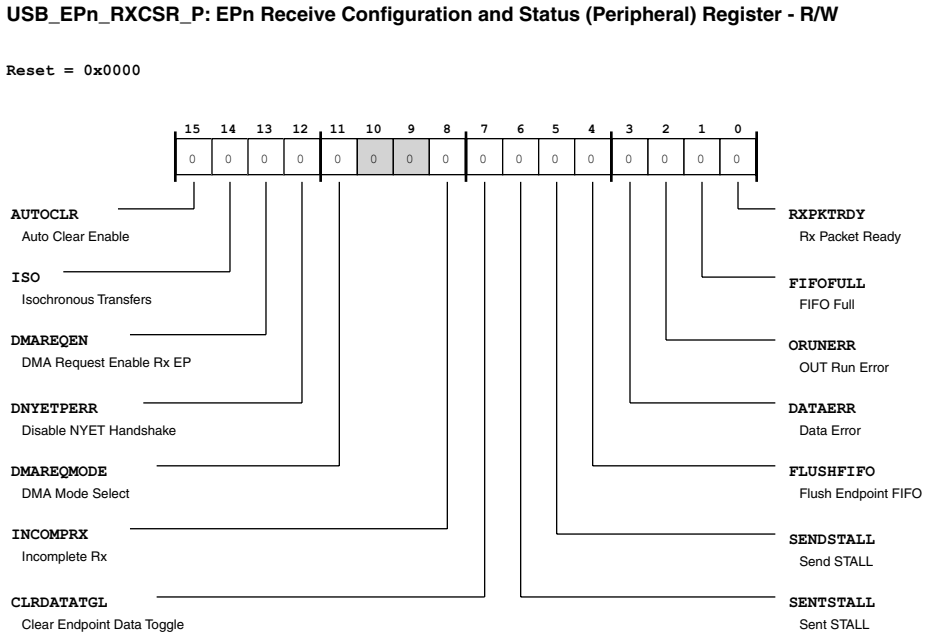


Figure 22-75: USB_EPn_RXCSR_P Register Diagram

Table 22-50: USB_EPn_RXCSR_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOCLR	Auto Clear Enable. The USB_EPn_RXCSR_P.AUTOCLR bit directs (in peripheral mode) the USB controller to automatically clear the USB_EPn_RXCSR_P.RXPKTRDY bit when a packet of size USB_EPn_RXMAXP bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, the processor must clear USB_EPn_RXCSR_P.RXPKTRDY manually. When using the DMA to unload the receive FIFO, data is read from the receive FIFO in four byte chunks, regardless of the USB_EPn_RXMAXP value. The USB controller auto clears the USB_EPn_RXCSR_P.RXPKTRDY bit as follows. (In the following: Remainder=(RxMaxP/4), and PktSz-Clearing-RxPktRdy=Actual-Bytes-Read-Packet-Sizes-That-Clear-RxPktRdy.) • Remainder=0, Bytes-Read=RxMaxP, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2, RxMaxP-3 • Remainder=3, Bytes Read=RxMaxP+1, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2 • Remainder=2, Bytes Read=RxMaxP+2, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1 • Remainder=1, Bytes Read=RxMaxP+3, PktSz-Clearing-RxPktRdy=RxMaxP Note that the USB_EPn_RXCSR_P.AUTOCLR bit should not be set for high-bandwidth isochronous endpoints.	
		0	Disable Auto Clear
		1	Enable Auto Clear
14 (R/W)	ISO	Isochronous Transfers. The USB_EPn_RXCSR_P.ISO bit selects (in peripheral mode) between isochronous transfers and bulk/interrupt transfers.	
		0	This bit should be cleared for bulk or interrupt transfers.
		1	This bit should be set for isochronous transfers.
13 (R/W)	DMAREQEN	DMA Request Enable Rx EP. The USB_EPn_RXCSR_P.DMAREQEN bit enables (in peripheral mode) DMA requests for this receive endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 22-50: USB_EPn_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	DNYETPERR	Disable NYET Handshake. The USB_EPn_RXCSR_P.DNYETPERR bit disables (in peripheral mode for high speed bulk/interrupt transactions) NYET handshakes. When this bit is set, all successful receive packets are ACK'd, including the point at which the FIFO becomes full. The USB_EPn_RXCSR_P.DNYETPERR bit must be set for all interrupt endpoints in high speed mode.	
		0	Enable NYET Handshake
		1	Disable NYET Handshake
11 (R/W)	DMAREQMODE	DMA Mode Select. The USB_EPn_RXCSR_P.DMAREQMODE bit selects (in peripheral mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the USB_EPn_RXCSR_P.DMAREQEN bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
8 (R/W)	INCOMPRX	Incomplete Rx. The USB_EPn_RXCSR_P.INCOMPRX bit indicates (in peripheral mode for high-bandwidth isochronous or interrupt transfers) when the received packet is incomplete because parts of the packet were not received. This bit is cleared when USB_EPn_RXCSR_P.RXPKTRDY is cleared. For all other modes, this bit is zero.	
		0	No Status
		1	Incomplete Rx
7 (R/W1A)	CLRDATATGL	Clear Endpoint Data Toggle. The USB_EPn_RXCSR_P.CLRDATATGL bit is set (in peripheral mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
6 (R/W0C)	SENTSTALL	Sent STALL. The USB_EPn_RXCSR_P.SENTSTALL bit indicates (in peripheral mode) when a STALL handshake is transmitted. The processor should clear this bit.	
		0	No Status
		1	STALL Handshake Transmitted
5 (R/W)	SENDSTALL	Send STALL. The USB_EPn_RXCSR_P.SENDSTALL bit is set (in peripheral mode) by the processor to send a STALL handshake. The processor clears this bit to terminate the stall condition. This bit has no effect for isochronous transfers.	
		0	No Action
		1	Request STALL Handshake

Table 22-50: USB_EPn_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W1A)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EPn_RXCSR_P.FLUSHFIFO bit directs (in peripheral mode) the USB controller to flush data from the endpoint FIFO and clear the USB_EPn_RXCSR_P.RXPKTRDY bit. The USB_EPn_RXCSR_P.FLUSHFIFO bit should only be set if the USB_EPn_RXCSR_P.RXPKTRDY bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
3 (R/NW)	DATAERR	Data Error. The USB_EPn_RXCSR_P.DATAERR bit indicates (in peripheral mode for isochronous transfers) when the USB_EPn_RXCSR_P.RXPKTRDY bit is set and the data packet has a CRC or bit-stuff error. This bit is cleared when USB_EPn_RXCSR_P.RXPKTRDY is cleared. The USB_EPn_RXCSR_P.DATAERR bit is always zero for bulk endpoints in peripheral mode.	
		0	No Status
		1	Data Error
2 (R/W0C)	ORUNERR	OUT Run Error. The USB_EPn_RXCSR_P.ORUNERR bit indicates (in peripheral mode for isochronous transfers) when an OUT packet cannot be loaded into the receive FIFO. The processor should clear this bit. The USB_EPn_RXCSR_P.ORUNERR bit always returns zero in bulk mode.	
		0	No Status
		1	OUT Run Error
1 (R/NW)	FIFOFULL	FIFO Full. The USB_EPn_RXCSR_P.FIFOFULL bit indicates (in peripheral mode) when no more packets can be loaded into the receive FIFO.	
		0	No Status
		1	FIFO Full
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The USB_EPn_RXCSR_P.RXPKTRDY is set (in peripheral mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EP0 Number of Received Bytes Register

The USB_EP0_CNTn register indicates the number of received data bytes in the endpoint 0 FIFO. The value returned changes as the contents of the FIFO change and is only valid while the USB_EP0_CSRn_H.RXPKTRDY bit or USB_EP0_CSRn_P.RXPKTRDY bit is set.

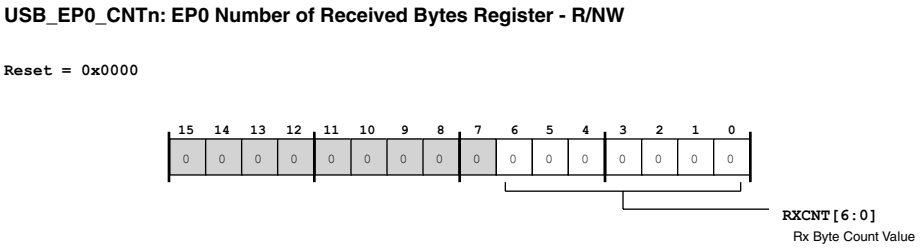


Figure 22-76: USB_EP0_CNTn Register Diagram

Table 22-51: USB_EP0_CNTn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/NW)	RXCNT	Rx Byte Count Value. The USB_EP0_CNTn.RXCNT bits holds the number of data bytes currently inline ready to be read from the Rx FIFO. The value returned changes as the FIFO is unloaded and is only valid while USB_EP0_CSRn_H.RXPKTRDY bit or USB_EP0_CSRn_P.RXPKTRDY bit is set.

EPn Number of Bytes Received Register

The USB_EPn_RXCNT register indicates the number of received data bytes in the endpoint receive FIFO. The value returned changes as the contents of the FIFO change and is only valid while the USB_EPn_RXCSR_H.RXPKTRDY bit or USB_EPn_RXCSR_P.RXPKTRDY bit is set.

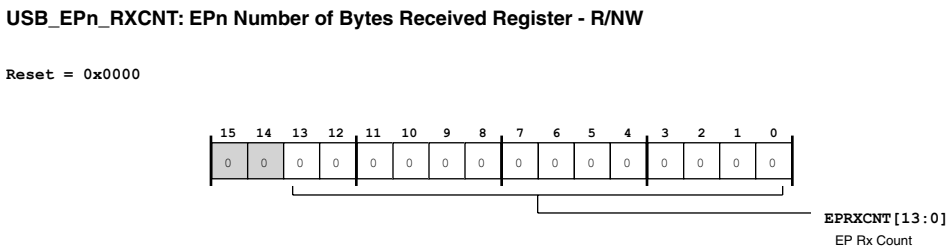


Figure 22-77: USB_EPn_RXCNT Register Diagram

Table 22-52: USB_EPn_RXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13:0 (R/NW)	EPRXCNT	EP Rx Count. The USB_EPn_RXCNT.EPRXCNT bits hold the number of data bytes ready to be read from the receive FIFO.

EPn Transmit Type Register

The USB_EPn_TXTYPE register selects the endpoint number and transaction protocol to use for the currently selected transmit endpoint. There is a USB_EPn_TXTYPE register for each transmit endpoint. Note that this register is only used in host mode.

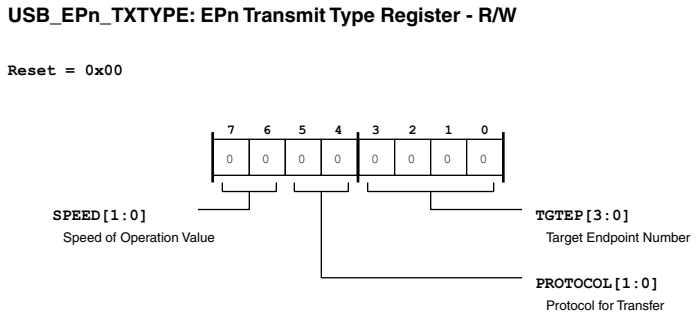


Figure 22-78: USB_EPn_TXTYPE Register Diagram

Table 22-53: USB_EPn_TXTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	SPEED	Speed of Operation Value. The USB_EPn_TXTYPE.SPEED bits select the USB controller operating speed for the endpoint when acting as a host connected to devices through a hub. In these instances. the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When not connected to devices through a hub, program this field with 00.	
		0	Same Speed as the Core
		1	High Speed
		2	Full Speed
		3	Low Speed
5:4 (R/W)	PROTOCOL	Protocol for Transfer. The USB_EPn_TXTYPE.PROTOCOL bits select the transfer protocol for the endpoint.	
		0	Control
		1	Isochronous
		2	Bulk
		3	Interrupt

Table 22-53: USB_EPn_TXTYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/W)	TGTEP	Target Endpoint Number. The USB_EPn_TXTYPE.TGTEP bits select (for endpoints 1-11) the target endpoint. This value should be set to the endpoint number contained in the transmit endpoint descriptor returned during device enumeration. Endpoint 0 always uses target endpoint number 0. (Enumeration values not shown are reserved.)	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15

EP0 Connection Type Register

The USB_EP0_TYPEn register selects the USB controller operating speed for endpoint 0 when acting as a host connected to devices through a hub.

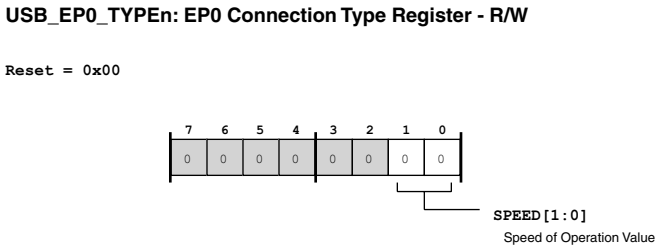


Figure 22-79: USB_EP0_TYPEn Register Diagram

Table 22-54: USB_EP0_TYPEn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	SPEED	Speed of Operation Value. The USB_EP0_TYPEn.SPEED bits select the USB controller operating speed for endpoint 0 when acting as a host connected to devices through a hub. In these instances. the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When not connected to devices through a hub, program this field with 00.	
		0	Same Speed as Processor Core
		1	High Speed
		2	Full Speed
		3	Low Speed

EP0 NAK Limit Register

The USB_EP0_NAKLIMITn register determines the number of frames/micro-frames after which endpoint 0 should timeout on receiving a stream of NAK responses for bulk endpoints.

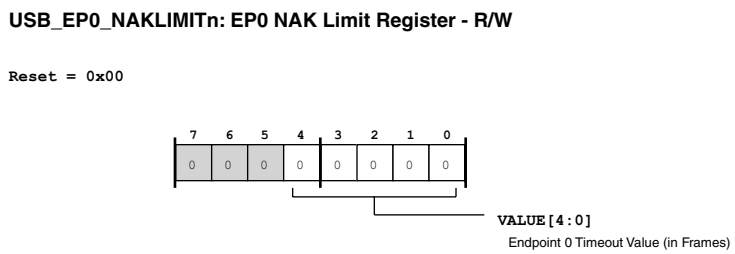


Figure 22-80: USB_EP0_NAKLIMITn Register Diagram

Table 22-55: USB_EP0_NAKLIMITn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	VALUE	Endpoint 0 Timeout Value (in Frames). The USB_EP0_NAKLIMITn.VALUE bits hold the endpoint 0 timeout value (number of frames).

EPn Transmit Polling Interval Register

The USB_EPn_TXINTERVAL register defines the polling interval for the currently selected transmit endpoint for interrupt, isochronous, and bulk transfers. There is a USB_EPn_TXINTERVAL register for each config-

ured transmit endpoint, except endpoint 0. The transfer types relate to speed, interval value, and interval operation as follows:

- Interrupt: Speed=Low Speed or Full Speed, USB_EPn_TXINTERVAL=1-255, and Operation=Polling interval is m frames.
- Interrupt: Speed=High Speed, USB_EPn_TXINTERVAL=1-16, and Operation=Polling interval is $2^{(m-1)}$ micro-frames.
- Isochronous: Speed=Full Speed or High Speed, USB_EPn_TXINTERVAL=1-16, and Operation=Polling interval is $2^{(m-1)}$ frames or micro-frames.
- Bulk: Speed=Full Speed or High Speed, USB_EPn_TXINTERVAL=2-16, and Operation=NAK Limit is $2^{(m-1)}$ frames or micro-frames.

Note that a USB_EPn_TXINTERVAL value of 0 or 1 disables the NAK timeout function.

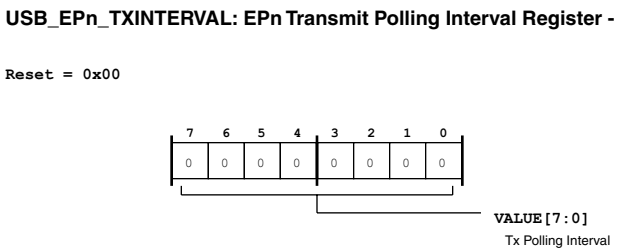


Figure 22-81: USB_EPn_TXINTERVAL Register Diagram

Table 22-56: USB_EPn_TXINTERVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Tx Polling Interval. The USB_EPn_TXINTERVAL.VALUE bits define the polling interval value for interrupt and isochronous transfers and select the number of frames/microframes after which the endpoint should timeout on receiving a stream of NAK responses for bulk and control endpoints. Note that the USB controller halts transfers to control endpoints if the host receives NAK responses for more frames than the limit set by this register.

EPn Receive Type Register

The USB_EPn_RXTYPE register selects the endpoint number and transaction protocol to use for the currently selected receive endpoint. There is a USB_EPn_RXTYPE register for each receive endpoint. Note that this register is only used in host mode.

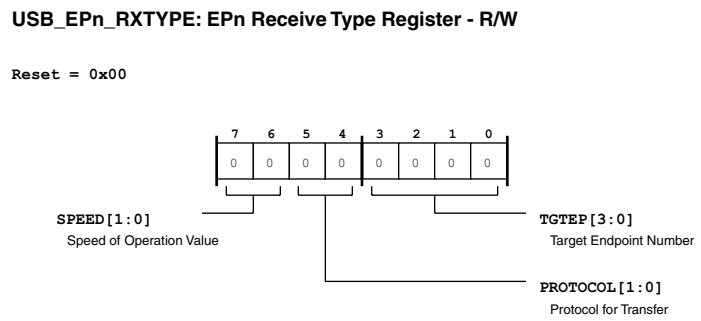


Figure 22-82: USB_EPn_RXTYPE Register Diagram

Table 22-57: USB_EPn_RXTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	SPEED	Speed of Operation Value. The USB_EPn_RXTYPE.SPEED bits select the USB controller operating speed for the endpoint when acting as a host connected to devices through a hub. In these instances. the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When not connected to devices through a hub, program this field with 00.	
		0	Same Speed as the Core
		1	High Speed
		2	Full Speed
		3	Low Speed
5:4 (R/W)	PROTOCOL	Protocol for Transfer. The USB_EPn_RXTYPE.PROTOCOL bits select the transfer protocol for the endpoint.	
		0	Control
		1	Isochronous
		2	Bulk
		3	Interrupt

Table 22-57: USB_EPn_RXTYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/W)	TGTEP	Target Endpoint Number. The USB_EPn_RXTYPE.TGTEP bits select (for endpoints 1-11) the target endpoint. This value should be set to the endpoint number contained in the receive endpoint descriptor returned during device enumeration. Endpoint 0 always uses target endpoint number 0. (Enumeration values not shown are reserved.)	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15

EPn Receive Polling Interval Register

The USB_EPn_RXINTERVAL register defines the polling interval for the currently selected receive endpoint for interrupt, isochronous, and bulk transfers. There is a USB_EPn_RXINTERVAL register for each configured receive endpoint, except endpoint 0. The transfer types relate to speed, interval value, and interval operation as follows:

- Interrupt: Speed=Low Speed or Full Speed, USB_EPn_RXINTERVAL=1-255, and Operation=Polling interval is m frames.
- Interrupt: Speed=High Speed, USB_EPn_RXINTERVAL=1-16, and Operation=Polling interval is $2^{(m-1)}$ micro-frames.
- Isochronous: Speed=Full Speed or High Speed, USB_EPn_RXINTERVAL=1-16, and Operation=Polling interval is $2^{(m-1)}$ frames or micro-frames.

- Bulk: Speed=Full Speed or High Speed, USB_EPn_RXINTERVAL=2-16, and Operation=NAK Limit is $2^{(m-1)}$ frames or micro-frames.

Note that a USB_EPn_RXINTERVAL value of 0 or 1 disables the NAK timeout function.

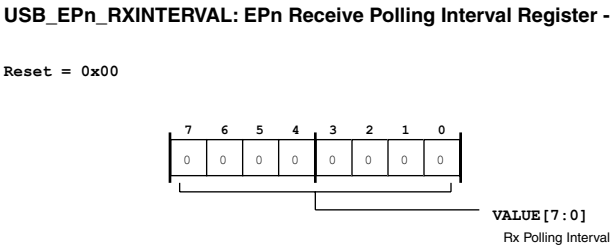


Figure 22-83: USB_EPn_RXINTERVAL Register Diagram

Table 22-58: USB_EPn_RXINTERVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Rx Polling Interval. The USB_EPn_RXINTERVAL.VALUE bits define the polling interval value for interrupt and isochronous transfers and select the number of frames/microframes after which the endpoint should timeout on receiving a stream of NAK responses for bulk and control endpoints. Note that the USB controller halts transfers to control endpoints if the host receives NAK responses for more frames than the limit set by this register.

EP0 Configuration Information Register

The USB_EP0_CFGDATAn register describes the USB controller hardware configuration. This register only exists for endpoint 0.

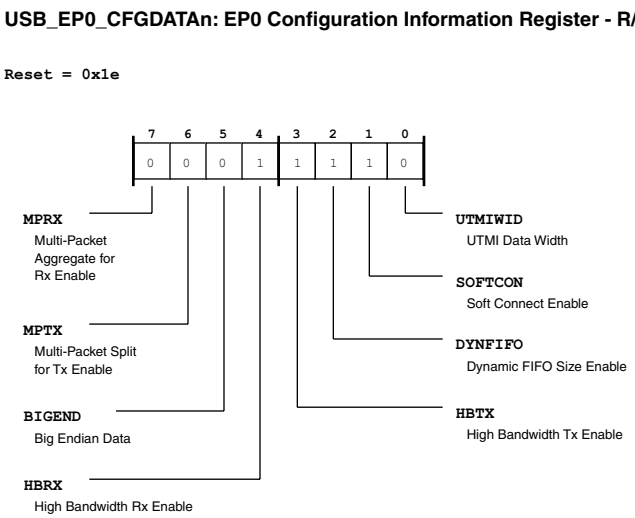


Figure 22-84: USB_EP0_CFGDATAn Register Diagram

Table 22-59: USB_EP0_CFGDATAn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R0/NW)	MPRX	Multi-Packet Aggregate for Rx Enable. The USB_EP0_CFGDATAn.MPRX bit indicates whether the USB controller aggregates receive packets into bulk packets before the processor core reads the data.	
		0	No Aggregate Rx Bulk Packets
		1	Aggregate Rx Bulk Packets
6 (R0/NW)	MPTX	Multi-Packet Split for Tx Enable. The USB_EP0_CFGDATAn.MPTX bit indicates whether the USB controller permits transmit of large packets through writing to bulk endpoints. The USB controller splits the transmit data into packets, which are appropriately sized for transmit.	
		0	No Split Tx Bulk Packets
		1	Split Tx Bulk Packets
5 (R0/NW)	BIGEND	Big Endian Data. The USB_EP0_CFGDATAn.BIGEND bit indicates whether the USB controller uses big endian configuration or little endian configuration.	
		0	Little Endian Configuration
		1	Big Endian Configuration
4 (R1/NW)	HBRX	High Bandwidth Rx Enable. The USB_EP0_CFGDATAn.HBRX bit indicates whether the USB controller supports high-bandwidth receive ISO endpoint support.	
		0	No High Bandwidth Rx
		1	High Bandwidth Rx

Table 22-59: USB_EP0_CFGDATAn Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R1/NW)	HBTX	High Bandwidth Tx Enable. The USB_EP0_CFGDATAn.HBTX bit indicates whether the USB controller supports high bandwidth transmit ISO endpoint support.	
		0	No High Bandwidth Tx
		1	High Bandwidth Tx
2 (R1/NW)	DYNFIFO	Dynamic FIFO Size Enable. The USB_EP0_CFGDATAn.DYNFIFO bit indicates whether the USB controller uses dynamic FIFO size support (on products supporting this feature), enabling the dynamic FIFO registers. These registers are accessed using the configuration set in the endpoints indexed FIFO size and FIFO address registers, except for endpoint 0.	
		0	No Dynamic FIFO Size
		1	Dynamic FIFO Size
1 (R1/NW)	SOFTCON	Soft Connect Enable. The USB_EP0_CFGDATAn.SOFTCON bit indicates whether the USB controller uses soft connect.	
		0	No Soft Connect
		1	Soft Connect
0 (R0/W)	UTMIWID	UTMI Data Width. The USB_EP0_CFGDATAn.UTMIWID bit indicates whether the USB controller uses an 8-bit or 16-bit UTMI data width.	
		0	8-bit UTMI Data Width
		1	16-bit UTMI Data Width

DMA Interrupt Register

The USB_DMA_IRQ register indicates which of the DMA master channels have a pending interrupt. The USB controller generates the interrupt when the corresponding DMA count register (USB_DMAn_CNT) reaches zero. The USB controller clears this register when it is read.

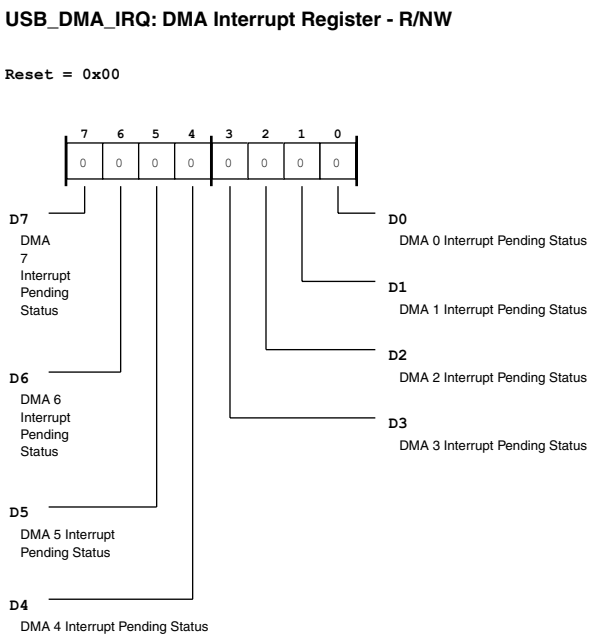


Figure 22-85: USB_DMA_IRQ Register Diagram

Table 22-60: USB_DMA_IRQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (RC/NW)	D7	DMA 7 Interrupt Pending Status. The USB_DMA_IRQ.D7 indicates whether there is a DMA 7 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
6 (RC/NW)	D6	DMA 6 Interrupt Pending Status. The USB_DMA_IRQ.D6 indicates whether there is a DMA 6 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
5 (RC/NW)	D5	DMA 5 Interrupt Pending Status. The USB_DMA_IRQ.D5 indicates whether there is a DMA 5 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
4 (RC/NW)	D4	DMA 4 Interrupt Pending Status. The USB_DMA_IRQ.D4 indicates whether there is a DMA 4 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt

Table 22-60: USB_DMA_IRQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (RC/NW)	D3	DMA 3 Interrupt Pending Status. The USB_DMA_IRQ.D3 indicates whether there is a DMA 3 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
2 (RC/NW)	D2	DMA 2 Interrupt Pending Status. The USB_DMA_IRQ.D2 indicates whether there is a DMA 2 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
1 (RC/NW)	D1	DMA 1 Interrupt Pending Status. The USB_DMA_IRQ.D1 indicates whether there is a DMA 1 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
0 (RC/NW)	D0	DMA 0 Interrupt Pending Status. The USB_DMA_IRQ.D0 indicates whether there is a DMA 0 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt

DMA Channel n Control Register

There is a USB_DMAn_CTL register for each DMA master channel. This register assigns, configures, and controls each endpoint with a corresponding DMA master channel.

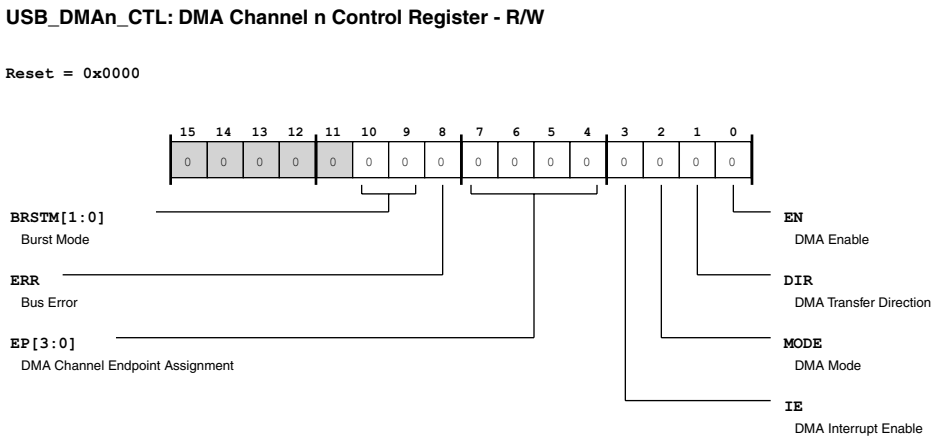


Figure 22-86: USB_DMAn_CTL Register Diagram

Table 22-61: USB_DMAn_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10:9 (R/W)	BRSTM	Burst Mode. The USB_DMAn_CTL.BRSTM bits select the type or length of burst transfer used by the corresponding DMA channel to transfer data.	
		0	Unspecified Length
		1	INCR4 or Unspecified Length
		2	INCR8, INCR4, or Unspecified Length
		3	INCR16, INCR8, INCR4, or Unspecified Length
8 (R/W)	ERR	Bus Error. The USB_DMAn_CTL.ERR bit indicates when a peripheral bus error has been encountered by the master channel. This bit is cleared by software.	
		0	No Status
		1	Bus Error
7:4 (R/W)	EP	DMA Channel Endpoint Assignment. The USB_DMAn_CTL.EP bits select the endpoint assignments for the DMA channel. (Enumeration values not shown are reserved.)	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15
3 (R/W)	IE	DMA Interrupt Enable. The USB_DMAn_CTL.IE bit enables DMA interrupts for the DMA channel, enabling operation of the channels corresponding bit in the USB_DMA_IRQ register.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 22-61: USB_DMAn_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	MODE	DMA Mode. The USB_DMAn_CTL.MODE bit selects whether the DMA channel operates in DMA mode 0 or operates in DMA mode 1. Note that DMA mode 1 may only be used with bulk endpoints.	
		0	DMA Mode 0
		1	DMA Mode 1
1 (R/W)	DIR	DMA Transfer Direction. The USB_DMAn_CTL.DIR bit selects the DMA channel transfer direction, which must be selected for use with receive endpoints (DMA write=0) or transmit endpoints (DMA read=1).	
		0	DMA Write (for Rx Endpoint)
		1	DMA Read (for Tx Endpoint)
0 (R/W)	EN	DMA Enable. The USB_DMAn_CTL.EN bit enables the DMA channel starts the DMA transfer.	
		0	Disable DMA
		1	Enable DMA (Start Transfer)

DMA Channel n Address Register

The USB_DMAn_ADDR register indicates the location in on-chip memory where DMA data is written or read. The address must be aligned to 32-bit words (The lower two address bits are always zero.) This register increments as the DMA transfer progresses.

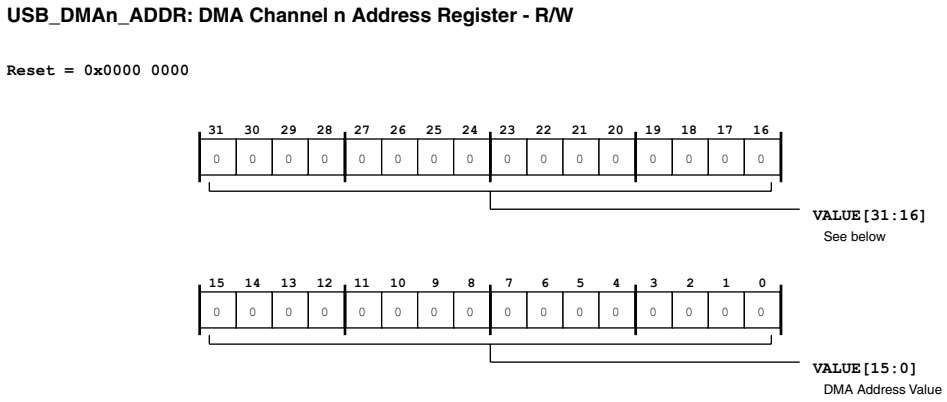


Figure 22-87: USB_DMAn_ADDR Register Diagram

Table 22-62: USB_DMAn_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DMA Address Value. The USB_DMAn_ADDR.VALUE bits hold the address value for the location in on-chip memory where DMA data is written or read.

DMA Channel n Count Register

The USB_DMAn_CNT register holds the DMA count, indicating the number of bytes to be transferred for a given DMA work block. If this field is set to zero, no data is transferred, and an interrupt is generated.

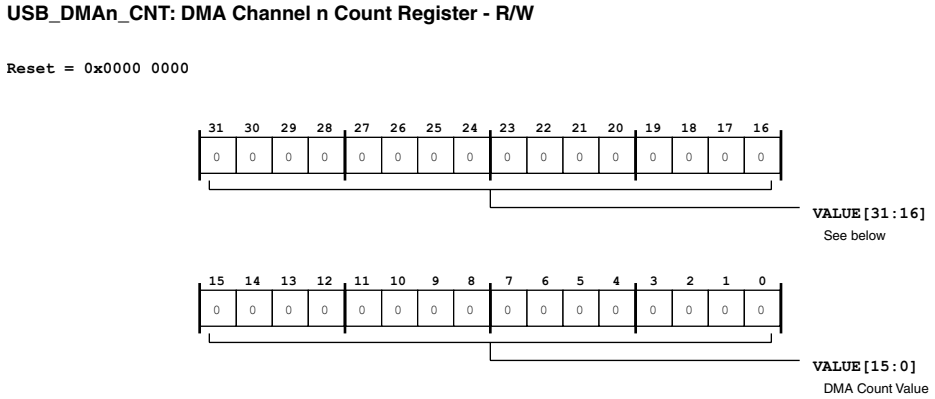


Figure 22-88: USB_DMAn_CNT Register Diagram

Table 22-63: USB_DMAn_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DMA Count Value. The USB_DMAn_CNT.VALUE bits indicate the number of bytes to be transferred for a given DMA work block.

EPn Request Packet Count Register

The USB_RQPKTCNTn register specifies (in host mode) the number of packets to request in a block transfer of one or more bulk packets of size USB_EPn_RXMAXP from a receive endpoint. This register only applies for receive endpoints 1 through 11 in host mode.

USB_RQPKTCNTn: EPn Request Packet Count Register - R/W

Reset = 0x0000

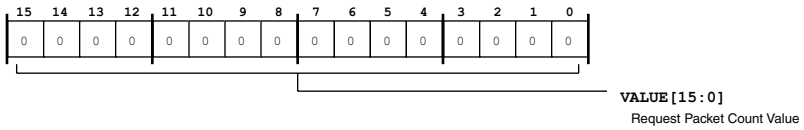


Figure 22-89: USB_RQPKTCNTn Register Diagram

Table 22-64: USB_RQPKTCNTn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Request Packet Count Value. The USB_RQPKTCNTn.VALUE bits specify the number of bulk packets to request in a block transfer from a receive endpoint. This field is used in conjunction with Auto Request feature (USB_EPn_RXCSR_H.AUTOREQ).

Chirp Timeout Register

The USB_CT_UCH register selects chirp timeout value. The is value is multiplied by 4 times the XCLK period (or 67ns). The default setting is 1.1ms.

USB_CT_UCH: Chirp Timeout Register - R/W

Reset = 0x4074

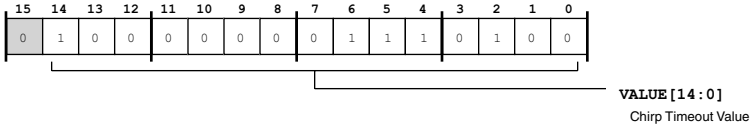


Figure 22-90: USB_CT_UCH Register Diagram

Table 22-65: USB_CT_UCH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/W)	VALUE	Chirp Timeout Value. The USB_CT_UCH.VALUE bits select the chirp timeout value.

Host High Speed Return to Normal Register

The USB_CT_HHSRTN register selects the delay from end of high speed resume signaling (acting as a host) to return to normal mode operation. This value is multiplied by 4 times the XCLK period (or 16.7ns). The default setting corresponds to a delay of 100us.

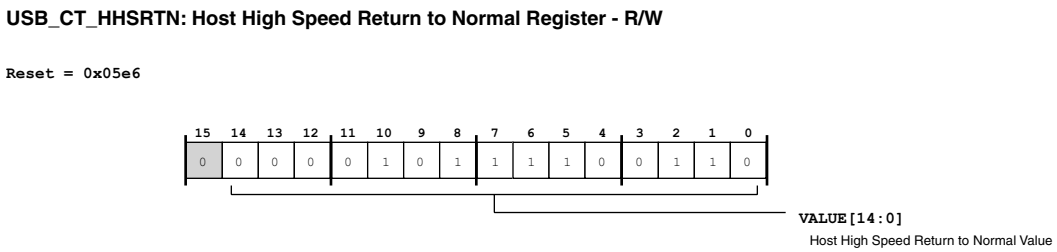


Figure 22-91: USB_CT_HHSRTN Register Diagram

Table 22-66: USB_CT_HHSRTN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/W)	VALUE	Host High Speed Return to Normal Value. The USB_CT_HHSRTN register selects the delay from end of high speed resume signaling (acting as a host) to return to normal mode operation.

High Speed Timeout Register

The USB_CT_HSBT register selects an amount of time to add to the minimum high speed timeout in units of 64 bit times. The USB 2.0 specification section 7.1.19.2 states that the controller must not timeout less than 736 bit times and must timeout after 816 bit times. The value in USB_CT_HSBT is multiplied by 64-bit times and added to the minimum 736 bit times. Settings less than 1 violate the USB 2.0 specification, making the controller non-compliant.

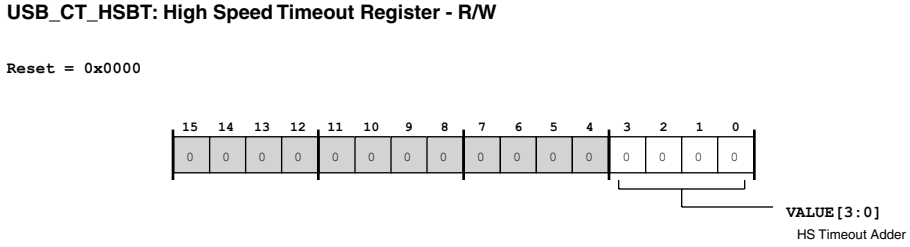


Figure 22-92: USB_CT_HSBT Register Diagram

Table 22-67: USB_CT_HSBT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/W)	VALUE	HS Timeout Adder. The USB_CT_HSBT.VALUE bits selects an amount of time to add to the minimum high speed timeout in units of 64 bit times.	
		0	HS Timeout = 736 (bit time)
		1	HS Timeout = 800 (bit time)
		2	HS Timeout = 864 (bit time)

LPM Attribute Register

The USB_LPM_ATTR register defines the link power management (LPM) attributes for LPM transactions and sleep/wake operation. In peripheral mode, the USB_LPM_ATTR register contains values received in the most recent, accepted (ACK'd) LPM transaction. In host mode, the USB_LPM_ATTR register contains values (loaded by software) that set up the next LPM transaction. The USB controller inserts the LPM values within the next LPM transaction.

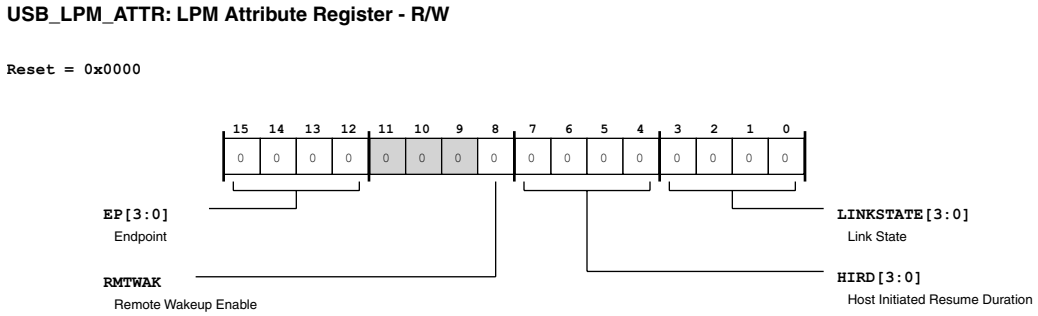


Figure 22-93: USB_LPM_ATTR Register Diagram

Table 22-68: USB_LPM_ATTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:12 (R/W)	EP	Endpoint. The USB_LPM_ATTR.EP bits select the endpoint in the token packet of the LPM transaction.	
8 (R/W)	RMTWAK	Remote Wakeup Enable. The USB_LPM_ATTR.RMTWAK bit enables remote wakeup. This bit is applied on a temporary basis only and is only applied to the current suspend state. After the current suspend cycle, the remote wakeup capability that was negotiated during enumeration applies.	
		0	Disable Remote Wakeup
		1	Enable Remote Wakeup

Table 22-68: USB_LPM_ATTR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7:4 (R/W)	HIRD	Host Initiated Resume Duration. The USB_LPM_ATTR.HIRD bits select the host initiated resume duration. This value is the minimum time that the host drives resume on the bus. The value in this register corresponds to an actual resume time of: Resume Time = 50us + HIRD*75us. This equation produces results in a range of 50us to 1200us.	
3:0 (R/W)	LINKSTATE	Link State. The USB_LPM_ATTR.LINKSTATE bits is value is provided by the host to the peripheral to indicate what state the peripheral must transition to after the receipt and acceptance of a LPM transaction. (Enumerations not shown are reserved.)	
		1	Sleep State (L1)

LPM Control Register

The USB_LPM_CTL register controls link power management (LPM) operations, including LPM enable, NAK, resume, and mode transition.

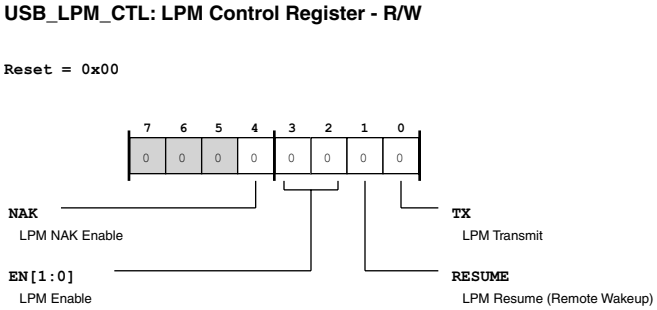


Figure 22-94: USB_LPM_CTL Register Diagram

Table 22-69: USB_LPM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	NAK	LPM NAK Enable. The USB_LPM_CTL.NAK bit enables (in peripheral mode) a NAK-all-non-LPM transactions mode for all end points, forcing a NAK response to all transactions other than an LPM transaction. This bit only takes effect after the controller has been LPM suspended. In this case, the USB controller continues to NAK, until this bit has been cleared by software.	
		0	Disable LPM NAK
		1	Enable LPM NAK

Table 22-69: USB_LPM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:2 (R/W)	EN	LPM Enable. The USB_LPM_CTL.EN bits enable (In peripheral mode) LPM operations. The LPM operation may be enabled at different levels, which determine the response of the USB controller to LPM transactions.	
		0	Disable LPM LPM and extended transactions are not supported. The USB controller does not respond to LPM transactions, and these transaction timeout.
		1	Disable LPM LPM and extended transactions are not supported. The USB controller does not respond to LPM transactions, and these transaction timeout.
		2	Enable Extended Transactions LPM is not supported, but extended transactions are supported. The USB controller responds to an LPM transaction with a STALL.
		3	Enable LPM and Extended Transactions Both LPM and extended transactions are supported. The USB controller responds with a NYET or an ACK as determined by the value of LPMXMT and other conditions.
1 (R/W)	RESUME	LPM Resume (Remote Wakeup). The USB_LPM_CTL.RESUME bit initiates resume (remote wakeup). This bits operation differs from the USB_POWER.RESUME bit in that the LPM resume signal timing is controlled by hardware. When set, the USB controller asserts resume signaling for 50us in host mode or asserts resume signaling for the time specified by the USB_LPM_ATTR.HIRD field in device mode. The USB_LPM_CTL.RESUME bit is self clearing.	
		0	No Action
		1	LPM Resume

Table 22-69: USB_LPM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	TX	LPM Transmit. The USB_LPM_CTL.TX bit puts the USB controller in LPM transmit mode, but this mode has differing operations in host mode versus peripheral mode. In peripheral mode, this bit is set by software to instruct the controller to transition to the L1 state upon receipt of the next LPM transaction. This bit is only effective if LPM enable (USB_LPM_CTL.EN) is set to 0x3. The LPM transmit enable bit can be set in the same cycle as LPM enable. If USB_LPM_CTL.TX and USB_LPM_CTL.EN are enabled, the USB controller can respond in the following ways: • If no data is pending (all transmit FIFOs are empty), the USB controller responds with an ACK, clears the USB_LPM_CTL.TX bit, and generates a software interrupt. • If data is pending (data resides in at least one transmit FIFO), the USB controller responds with a NYET, does not clear the USB_LPM_CTL.TX bit, and generates a software interrupt. In host mode, this bit is set by software to transmit an LPM transaction. This bit is self clearing. The USB controller clears this bit immediately on receipt of any token or after three timeouts have occurred.	
		0	Disable LPM Tx
		1	Enable LPM Tx

LPM Interrupt Enable Register

The USB_LPM_IEN register enables the link power management (LPM) related interrupts. When an interrupt is enabled in this register and the corresponding interrupt is pending in USB_LPM_IRQ, the USB controller generates the interrupt. When an interrupt is disabled in this register, the corresponding interrupt may be pending in USB_LPM_IRQ, but the USB controller does not generate an interrupt.

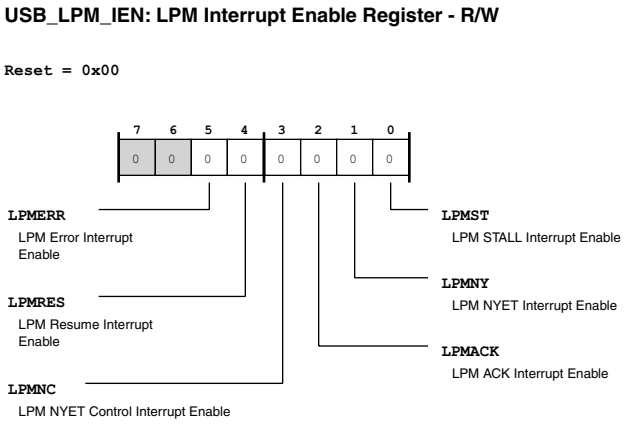


Figure 22-95: USB_LPM_IEN Register Diagram

Table 22-70: USB_LPM_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	LPMERR	LPM Error Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt
4 (R/W)	LPMRES	LPM Resume Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt
3 (R/W)	LPMNC	LPM NYET Control Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt
2 (R/W)	LPMACK	LPM ACK Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt
1 (R/W)	LPMNY	LPM NYET Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt
0 (R/W)	LPMST	LPM STALL Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt

LPM Interrupt Status Register

The USB_LPM_IRQ register indicates link power management (LPM) related interrupt status. The USB controller clears this register when it is read.

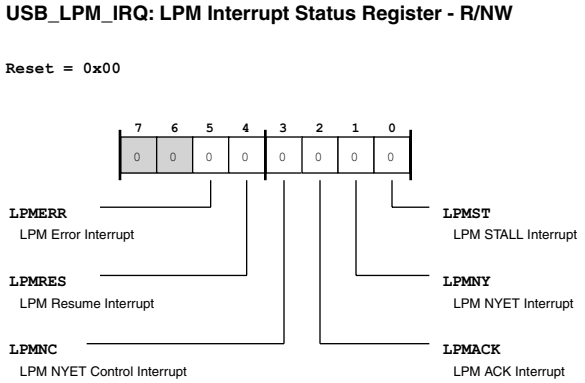


Figure 22-96: USB_LPM_IRQ Register Diagram

Table 22-71: USB_LPM_IRQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (RC/NW)	LPMERR	LPM Error Interrupt. The USB_LPM_IRQ.LPMERR bit indicates an LPM error interrupt condition, but this interrupt has differing conditions for host mode versus peripheral mode. In peripheral mode, this bit is set if an LPM transaction is received that has a LinkState field that is not supported. The USB controller responds to the transaction with a STALL. Note that the USB controller updates the USB_LPM_ATTR register, so software can observe the non compliant LPM packet payload. In host mode, this bit is set if the response to a LPM transaction is received with a bit stuff or PID error. No suspend occurs and the state of the device is now unknown.	
		0	No Interrupt Pending
		1	Interrupt Pending
4 (RC/NW)	LPMRES	LPM Resume Interrupt. The USB_LPM_IRQ.LPMRES bit indicates that the USB controller has been resumed for any reason. This bit is mutually exclusive from the USB_POWER.RESUME bit.	
		0	No Interrupt Pending
		1	Interrupt Pending
3 (RC/NW)	LPMNC	LPM NYET Control Interrupt. The USB_LPM_IRQ.LPMNC bit indicates an LPM NYET control interrupt condition, but this interrupt has differing conditions for host mode versus peripheral mode. In peripheral mode, this bit is set when an LPM transaction is received, and the USB controller responds with a NYET due to data pending in the transmit FIFOs. This interrupt may only occur when the USB_LPM_CTL.EN field is set to 11, the USB_LPM_CTL.TX field is set to 1, and there is data pending in the transmit FIFOs. In host mode, this bit is set when an LPM transaction has been transmitted, but has failed to complete. The transaction failure must be because a timeout occurred or be because there were bit errors in the response for three attempts.	
		0	No Interrupt Pending
		1	Interrupt Pending
2 (RC/NW)	LPMACK	LPM ACK Interrupt. The USB_LPM_IRQ.LPMACK bit indicates an LPM ACK interrupt condition, but this interrupt has differing conditions for host mode versus peripheral mode. In peripheral mode, this bit is set when an LPM transaction is received, and the USB controller responds with an ACK. This interrupt may only occur when the USB_LPM_CTL.EN field is set to 11, the USB_LPM_CTL.TX field is set to 1, and there is no data pending in the controller transmit FIFOs. In host mode, this bit is set when an LPM transaction is transmitted, and the device responds with an ACK.	
		0	No Interrupt Pending
		1	Interrupt Pending

Table 22-71: USB_LPM_IRQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (RC/NW)	LPMNY	LPM NYET Interrupt. The USB_LPM_IRQ.LPMNY bit indicates an LPM NYET interrupt condition, but this interrupt has differing conditions for host mode versus peripheral mode. In peripheral mode, this bit is set when an LPM transaction is received, and the USB controller responds with a NYET. This interrupt may only occur when the USB_LPM_CTL.EN field is set to 11, and the USB_LPM_CTL.TX field is set to 0. In host mode, this bit is set when an LPM transaction is transmitted and the device responds with a NYET.	
		0	No Interrupt Pending
		1	Interrupt Pending
0 (RC/NW)	LPMST	LPM STALL Interrupt. The USB_LPM_IRQ.LPMST bit indicates an LPM STALL interrupt condition, but this interrupt has differing conditions for host mode versus peripheral mode. This bit is set when an LPM transaction is received, and the USB controller responds with a STALL. This interrupt may only occur when the USB_LPM_CTL.EN field is set to 01. In host mode, this bit is set when an LPM transaction is transmitted, and the device responds with a STALL.	
		0	No Interrupt Pending
		1	Interrupt Pending

LPM Function Address Register

The USB_LPM_FADDR register selects the link power management (LPM) function address.

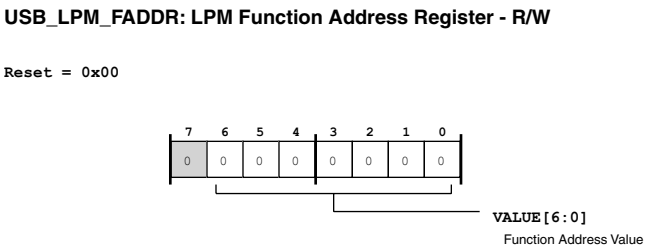


Figure 22-97: USB_LPM_FADDR Register Diagram

Table 22-72: USB_LPM_FADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Function Address Value. The USB_LPM_FADDR.VALUE bits hold the LPM function address value that the USB controller places in the LPM payload.

VBUS Control Register

The USB_VBUS_CTL controls USB controller VBUS related features.

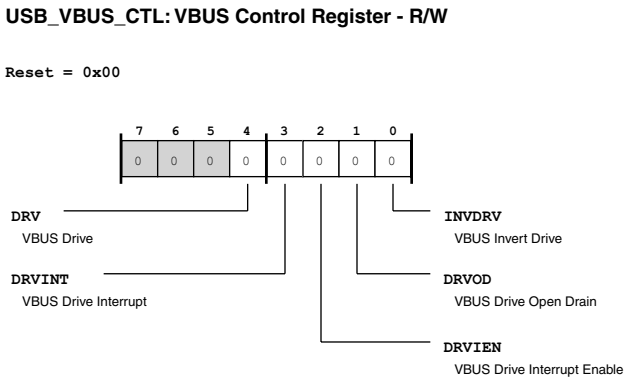


Figure 22-98: USB_VBUS_CTL Register Diagram

Table 22-73: USB_VBUS_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/NW)	DRV	VBUS Drive. The USB_VBUS_CTL.DRV bit indicates the state of the UTMI+ DrvVBUS signal from the USB controller.
3 (R/W1C)	DRVINT	VBUS Drive Interrupt. The USB_VBUS_CTL.DRVINT bit indicates the state of the DrvVBUSInt interrupt.
2 (R/W)	DRVIEN	VBUS Drive Interrupt Enable. The USB_VBUS_CTL.DRVIEN bit enables the DrvVBUS interrupt.
1 (R/W)	DRVOD	VBUS Drive Open Drain. The USB_VBUS_CTL.DRVOD selects whether the DrvVBUS output is open drain.
0 (R/W)	INVDRV	VBUS Invert Drive. The USB_VBUS_CTL.INVDRV bit selects whether the DrvVBUS output is inverted.

Battery Charging Control Register

The USB_BAT_CHG controls USB controller battery changing related features.

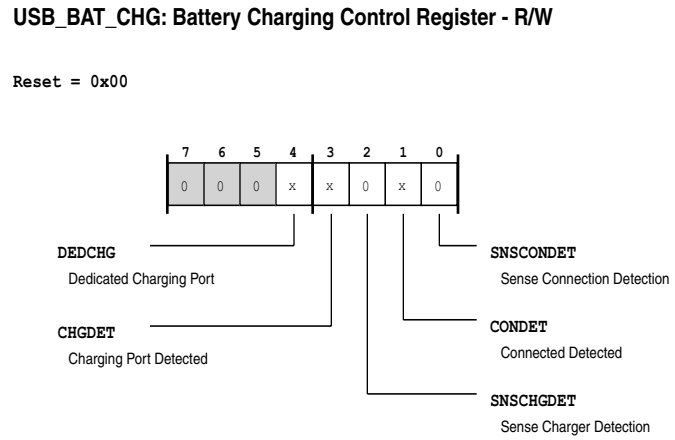


Figure 22-99: USB_BAT_CHG Register Diagram

Table 22-74: USB_BAT_CHG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	DEDCHG	Dedicated Charging Port. The USB_BAT_CHG.DEDCHG bit is asserted if both D+ and D- are high. This can be used to determine if the attached device is a dedicated charging port. This bit is the decode of LineState[1] and LineState[0]. This bit is only valid when a session is initiated, which enables a pullup on D+ when acting as a B-device.
3 (R/NW)	CHGDET	Charging Port Detected. The USB_BAT_CHG.CHGDET bit indicates when a charging port is detected. This bit indicates that D+/- is above V_{DAT_REF} and below V_{LGC}.
2 (R/W)	SNSCHGDET	Sense Charger Detection. The USB_BAT_CHG.SNSCHGDET bit enables charger detection. Setting this bit enables VD_SRC and ID_SINK.
1 (R/NW)	CONDET	Connected Detected. The USB_BAT_CHG.CONDET bit is valid when USB_BAT_CHG.SNSCONDET is enabled. This bit reflects the inverse of D+ (!LineState[0]). If nothing is connected, D+ is pulled high. If a charger or USB port is connected, D+ is pulled low.
0 (R/W)	SNSCONDET	Sense Connection Detection. The USB_BAT_CHG.SNSCONDET bit enables connection detection. Enabling this bit enables I_{DP_SRC} and R_{DM_DWN}.

PHY Control Register

The USB_PHY_CTL register provides access to PHY control features.

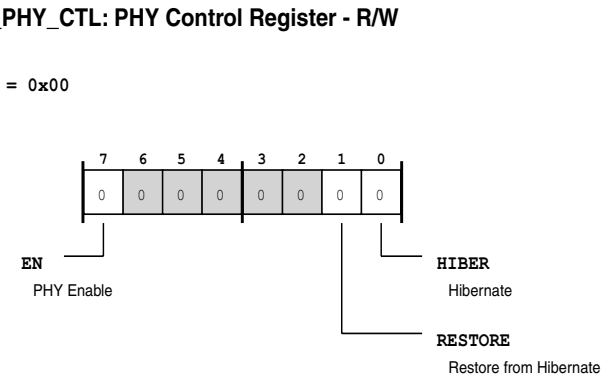


Figure 22-100: USB_PHY_CTL Register Diagram

Table 22-75: USB_PHY_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W)	EN	PHY Enable. The USB_PHY_CTL.EN bit enables the USB controller PHY. This bit enables the schmitt-trigger inputs on D+ and D- to detect session request protocol. The bit also enables the bias circuits and VBUS comparators to detect when a host is connected. This bit should be set for all USB controller operations.
1 (R/W)	RESTORE	Restore from Hibernate. The USB_PHY_CTL.RESTORE bit causes the PHY to come out of hibernate and release its latches.
0 (R/W)	HIBER	Hibernate. The USB_PHY_CTL.HIBER bit causes the PHY to prepare for hibernate. Latches hold the pullup/pulldown state when the core power is removed.

PLL and Oscillator Control Register

The USB_PLL_OSC register provides access to PLL and oscillator related control features.

USB_PLL_OSC: PLL and Oscillator Control Register - R/W

Reset = 0x0014

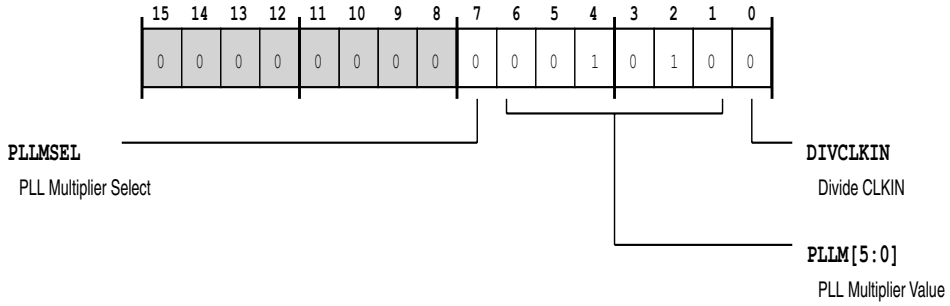


Figure 22-101: USB_PLL_OSC Register Diagram

Table 22-76: USB_PLL_OSC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W)	PLLMSEL	PLL Multiplier Select. The USB_PLL_OSC.PLLMSEL bit directs the PLL to use the PLL multiplier value is stored the USB_PLL_OSC.PLLM bits.
6:1 (R/W)	PLLM	PLL Multiplier Value. PLL multiplier. This field should be set such that CLKIN * (USB_PLL_OSC.PLLM value) = 480MHz.
0 (R/W)	DIVCLKIN	Divide CLKIN. The USB_PLL_OSC.DIVCLKIN bit enables a divide CLKIN by 2 function for the PLL.

.

23 Ethernet Media Access Controller (EMAC)

The EMAC peripheral present in the processor enables network connectivity to applications via a 10/100M bit/s Ethernet interface. The module is fully compliant to the following standards:

- Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications, Standard 802.3-2005, Institute of Electrical and Electronics Engineers (IEEE).
- Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, Standard 1588-2008, Institute of Electrical and Electronics Engineers (IEEE).
- Reduced Media Independent Interface Specification, Revision 1.2, RMII Consortium.

The EMAC interface consists of the hardware for the Media Access Control protocol. This allows applications to support TCP/IP based network communication. At the system end, the module supports direct connection with the System Crossbar bus for Memory/MMR transactions. It supports RMII (Reduced Media Independent Interface) and SMI (Station Management Interface) for interfacing with the external PHY chip.

The MAC also includes a built-in and dedicated DMA controller that performs both data and status transfers between the application and the RMII interface. Internal transmit and receive FIFOs are used to buffer and regulate the frames. A dedicated interrupt line connects the EMAC interrupt sources to the System Event Controller (SEC).

The MAC Management Counters (MMC) block is an extended set of registers that collects various statistics compliant with IEEE 802.3 definitions regarding the operation of the interface. The registers are updated for each new transmitted or received frame when the condition to update the counter is met. The EMAC provides a set of such counters, along with extended usage control.

The EMAC also includes a PTP (Precision Time Protocol) engine that provides hardware assistance for the implementation of the IEEE 1588 Version 1 and Version 2 standards on Blackfin processors, which allows time synchronization between systems.

EMAC Features

The Ethernet MAC's features include the following:

- Supports 10/100 Mbps data transfer rates with external PHY interfaced via RMII.
- Full-duplex and half-duplex support for Ethernet.

- Dedicated DMA controller with independent read write channels.
- Supports dual-buffer (ring) or linked-list (chained) descriptor chaining.
- Direct interface with the System Crossbar bus.
- Provides support for CSMA/CD protocol for half-duplex operation.
- IEEE 802.3x flow control for full-duplex and half-duplex.
- Automatic network monitoring statistics with management counters.
- Flexible address filtering options for uni-cast/multi-cast/broadcast addresses.
- Support for Promiscuous mode in reception.
- Supports IEEE 802.1Q VLAN tag detection.
- Supports programmable Inter-frame Gap (IFG).
- Checksum Offload Engine for checking IPv4 header checksum and TCP/UDP/ICMP checksum encapsulated in IPv4 or IPv6 datagrams.
- Station Management Interface for PHY device configuration and management.
- Includes FIFOs for buffering: 256 bytes for transmit FIFO and 128 bytes for receive FIFO.
- Automatic CRC and pad generation controllable on a per-frame basis.

EMAC Functional Description

This section provides information on the function of Ethernet MAC peripheral and contains the following topics.

ADSP-BF60x EMAC Register List

The ethernet MAC (EMAC) module provides a 10/100M bit/s Ethernet interface, compliant to IEEE Std. 802.3-2005, between an RMII (Reduced Media Independent Interface) and the Blackfin processor. A set of registers govern EMAC operations. For more information on EMAC functionality, see the EMAC register descriptions.

Table 23-1: ADSP-BF60x EMAC Register List

Name	Description
EMAC_MACCFG	MAC Configuration Register
EMAC_MACFRMFILT	MAC Rx Frame Filter Register
EMAC_HASHTBL_HI	Hash Table High Register
EMAC_HASHTBL_LO	Hash Table Low Register
EMAC_SMI_ADDR	SMI Address Register
EMAC_SMI_DATA	SMI Data Register
EMAC_FLOWCTL	FLow Control Register
EMAC_VLANTAG	VLAN Tag Register
EMAC_DBG	Debug Register
EMAC_ISTAT	Interrupt Status Register
EMAC_IMSK	Interrupt Mask Register
EMAC_ADDR0_HI	MAC Address 0 High Register
EMAC_ADDR0_LO	MAC Address 0 Low Register
EMAC_MMC_CTL	MMC Control Register
EMAC_MMC_RXINT	MMC Rx Interrupt Register
EMAC_MMC_TXINT	MMC Tx Interrupt Register
EMAC_MMC_RXIMSK	MMC Rx Interrupt Mask Register
EMAC_MMC_TXIMSK	MMC TX Interrupt Mask Register
EMAC_TXOCTCNT_GB	Tx OCT Count (Good/Bad) Register
EMAC_TXFRMCNT_GB	Tx Frame Count (Good/Bad) Register
EMAC_TXBCASTFRM_G	Tx Broadcast Frames (Good) Register
EMAC_TXMCASTFRM_G	Tx Multicast Frames (Good) Register

Table 23-1: ADSP-BF60x EMAC Register List (Continued)

Name	Description
EMAC_TX64_GB	Tx 64-Byte Frames (Good/Bad) Register
EMAC_TX65TO127_GB	Tx 65- to 127-Byte Frames (Good/Bad) Register
EMAC_TX128TO255_GB	Tx 128- to 255-Byte Frames (Good/Bad) Register
EMAC_TX256TO511_GB	Tx 256- to 511-Byte Frames (Good/Bad) Register
EMAC_TX512TO1023_GB	Tx 512- to 1023-Byte Frames (Good/Bad) Register
EMAC_TX1024TOMAX_GB	Tx 1024- to Max-Byte Frames (Good/Bad) Register
EMAC_TXUCASTFRM_GB	Tx Unicast Frames (Good/Bad) Register
EMAC_TXMCASTFRM_GB	Tx Multicast Frames (Good/Bad) Register
EMAC_TXBCASTFRM_GB	Tx Broadcast Frames (Good/Bad) Register
EMAC_TXUNDR_ERR	Tx Underflow Error Register
EMAC_TXSNGCOL_G	Tx Single Collision (Good) Register
EMAC_TXMULTCOL_G	Tx Multiple Collision (Good) Register
EMAC_TXDEFERRED	Tx Deferred Register
EMAC_TXLATECOL	Tx Late Collision Register
EMAC_TXEXCESSCOL	Tx Excess Collision Register
EMAC_TXCARR_ERR	Tx Carrier Error Register
EMAC_TXOCTCNT_G	Tx Octet Count (Good) Register
EMAC_TXFRMCNT_G	Tx Frame Count (Good) Register
EMAC_TXEXCESSDEF	Tx Excess Deferral Register
EMAC_TXPAUSEFRM	Tx Pause Frame Register
EMAC_TXVLANFRM_G	Tx VLAN Frames (Good) Register
EMAC_RXFRMCNT_GB	Rx Frame Count (Good/Bad) Register

Table 23-1: ADSP-BF60x EMAC Register List (Continued)

Name	Description
EMAC_RXOCTCNT_GB	Rx Octet Count (Good/Bad) Register
EMAC_RXOCTCNT_G	Rx Octet Count (Good) Register
EMAC_RXBCASTFRM_G	Rx Broadcast Frames (Good) Register
EMAC_RXMCASTFRM_G	Rx Multicast Frames (Good) Register
EMAC_RXCRC_ERR	Rx CRC Error Register
EMAC_RXALIGN_ERR	Rx alignment Error Register
EMAC_RXRUNT_ERR	Rx Runt Error Register
EMAC_RXJAB_ERR	Rx Jab Error Register
EMAC_RXUSIZE_G	Rx Undersize (Good) Register
EMAC_RXOSIZE_G	Rx Oversize (Good) Register
EMAC_RX64_GB	Rx 64-Byte Frames (Good/Bad) Register
EMAC_RX65TO127_GB	Rx 65- to 127-Byte Frames (Good/Bad) Register
EMAC_RX128TO255_GB	Rx 128- to 255-Byte Frames (Good/Bad) Register
EMAC_RX256TO511_GB	Rx 256- to 511-Byte Frames (Good/Bad) Register
EMAC_RX512TO1023_GB	Rx 512- to 1023-Byte Frames (Good/Bad) Register
EMAC_RX1024TOMAX_GB	Rx 1024- to Max-Byte Frames (Good/Bad) Register
EMAC_RXUCASTFRM_G	Rx Unicast Frames (Good) Register
EMAC_RXLEN_ERR	Rx Length Error Register
EMAC_RXOORTYPE	Rx Out Of Range Type Register
EMAC_RXPAUSEFRM	Rx Pause Frames Register
EMAC_RXFIFO_OVF	Rx FIFO Overflow Register
EMAC_RXVLANFRM_GB	Rx VLAN Frames (Good/Bad) Register

Table 23-1: ADSP-BF60x EMAC Register List (Continued)

Name	Description
EMAC_RXWDOG_ERR	Rx Watch Dog Error Register
EMAC_IPC_RXIMSK	MMC IPC Rx Interrupt Mask Register
EMAC_IPC_RXINT	MMC IPC Rx Interrupt Register
EMAC_RXIPV4_GD_FRM	Rx IPv4 Datagrams (Good) Register
EMAC_RXIPV4_HDR_ERR_FRM	Rx IPv4 Datagrams Header Errors Register
EMAC_RXIPV4_NOPAY_FRM	Rx IPv4 Datagrams No Payload Frame Register
EMAC_RXIPV4_FRAG_FRM	Rx IPv4 Datagrams Fragmented Frames Register
EMAC_RXIPV4_UDSBL_FRM	Rx IPv4 UDP Disabled Frames Register
EMAC_RXIPV6_GD_FRM	Rx IPv6 Datagrams Good Frames Register
EMAC_RXIPV6_HDR_ERR_FRM	Rx IPv6 Datagrams Header Error Frames Register
EMAC_RXIPV6_NOPAY_FRM	Rx IPv6 Datagrams No Payload Frames Register
EMAC_RXUDP_GD_FRM	Rx UDP Good Frames Register
EMAC_RXUDP_ERR_FRM	Rx UDP Error Frames Register
EMAC_RXTCP_GD_FRM	Rx TCP Good Frames Register
EMAC_RXTCP_ERR_FRM	Rx TCP Error Frames Register
EMAC_RXICMP_GD_FRM	Rx ICMP Good Frames Register
EMAC_RXICMP_ERR_FRM	Rx ICMP Error Frames Register
EMAC_RXIPV4_GD_OCT	Rx IPv4 Datagrams Good Octets Register
EMAC_RXIPV4_HDR_ERR_OCT	Rx IPv4 Datagrams Header Errors Register
EMAC_RXIPV4_NOPAY_OCT	Rx IPv4 Datagrams No Payload Octets Register
EMAC_RXIPV4_FRAG_OCT	Rx IPv4 Datagrams Fragmented Octets Register
EMAC_RXIPV4_UDSBL_OCT	Rx IPv4 UDP Disabled Octets Register

Table 23-1: ADSP-BF60x EMAC Register List (Continued)

Name	Description
EMAC_RXIPV6_GD_OCT	Rx IPv6 Good Octets Register
EMAC_RXIPV6_HDR_ERR_OCT	Rx IPv6 Header Errors Register
EMAC_RXIPV6_NOPAY_OCT	Rx IPv6 No Payload Octets Register
EMAC_RXUDP_GD_OCT	Rx UDP Good Octets Register
EMAC_RXUDP_ERR_OCT	Rx UDP Error Octets Register
EMAC_RXTCP_GD_OCT	Rx TCP Good Octets Register
EMAC_RXTCP_ERR_OCT	Rx TCP Error Octets Register
EMAC_RXICMP_GD_OCT	Rx ICMP Good Octets Register
EMAC_RXICMP_ERR_OCT	Rx ICMP Error Octets Register
EMAC_TM_CTL	Time Stamp Control Register
EMAC_TM_SUBSEC	Time Stamp Sub Second Increment Register
EMAC_TM_SEC	Time Stamp Low Seconds Register
EMAC_TM_NSEC	Time Stamp Nanoseconds Register
EMAC_TM_SECUPDT	Time Stamp Seconds Update Register
EMAC_TM_NSECUPDT	Time Stamp Nanoseconds Update Register
EMAC_TM_ADDEND	Time Stamp Addend Register
EMAC_TM_TGTM	Time Stamp Target Time Seconds Register
EMAC_TM_NTGTM	Time Stamp Target Time Nanoseconds Register
EMAC_TM_HISEC	Time Stamp High Second Register
EMAC_TM_STMPSTAT	Time Stamp Status Register
EMAC_TM_PPSCTL	PPS Control Register
EMAC_TM_AUXSTMP_NSEC	Time Stamp Auxiliary TS Nano Seconds Register

Table 23-1: ADSP-BF60x EMAC Register List (Continued)

Name	Description
EMAC_TM_AUXSTMP_SEC	Time Stamp Auxiliary TM Seconds Register
EMAC_TM_PPSINTVL	Time Stamp PPS Interval Register
EMAC_TM_PPSWIDTH	PPS Width Register
EMAC_DMA_BUSMODE	DMA Bus Mode Register
EMAC_DMA_TXPOLL	DMA Tx Poll Demand Register
EMAC_DMA_RXPOLL	DMA Rx Poll Demand register
EMAC_DMA_RXDSC_ADDR	DMA Rx Descriptor List Address Register
EMAC_DMA_TXDSC_ADDR	DMA Tx Descriptor List Address Register
EMAC_DMA_STAT	DMA Status Register
EMAC_DMA_OPMODE	DMA Operation Mode Register
EMAC_DMA_IEN	DMA Interrupt Enable Register
EMAC_DMA_MISS_FRM	DMA Missed Frame Register
EMAC_DMA_RXIWDOG	DMA Rx Interrupt Watch Dog Register
EMAC_DMA_BMMODE	DMA SCB Bus Mode Register
EMAC_DMA_BMSTAT	DMA SCB Status Register
EMAC_DMA_TXDSC_CUR	DMA Tx Descriptor Current Register
EMAC_DMA_RXDSC_CUR	DMA Rx Descriptor Current Register
EMAC_DMA_TXBUF_CUR	DMA Tx Buffer Current Register
EMAC_DMA_RXBUF_CUR	DMA Rx Buffer Current Register

ADSP-BF60x EMAC Interrupt List

Table 23-2: ADSP-BF60x EMAC Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
EMAC0 Status	68		LEVEL
EMAC1 Status	70		LEVEL

ADSP-BF60x EMAC Trigger List

Table 23-3: ADSP-BF60x EMAC Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
EMAC0 Status	33	LEVEL
EMAC1 Status	34	LEVEL

Table 23-4: ADSP-BF60x EMAC Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
None		

EMAC Definitions

The following definitions aid with using the EMAC.

EMAC SCB

System Crossbar interface of EMAC

EMAC DMA

DMA Controller of EMAC

EMAC MFL

MAC FIFO Layer inside EMAC

EMAC CORE

CORE Layer inside EMAC which performs the actual Ethernet operations, including interface with PHY via RMII.

MMC

MAC Management Counter

SMI

Station Management Interface that controls PHY via MDIO/MDC signals.

MII

Reduced Media Independent Interface

MAC

Media Access Control

PTP

Precision Time Protocol

EMAC Block Diagram and Interfaces

The following figure illustrates the overall functional architecture of the Ethernet MAC peripheral. The EMAC module is comprised of four major layers, EMAC SCB, EMAC DMA, EMAC MFL and EMAC CORE. Each of these layers (sub-blocks) are explained in depth in their respective sections in this chapter.

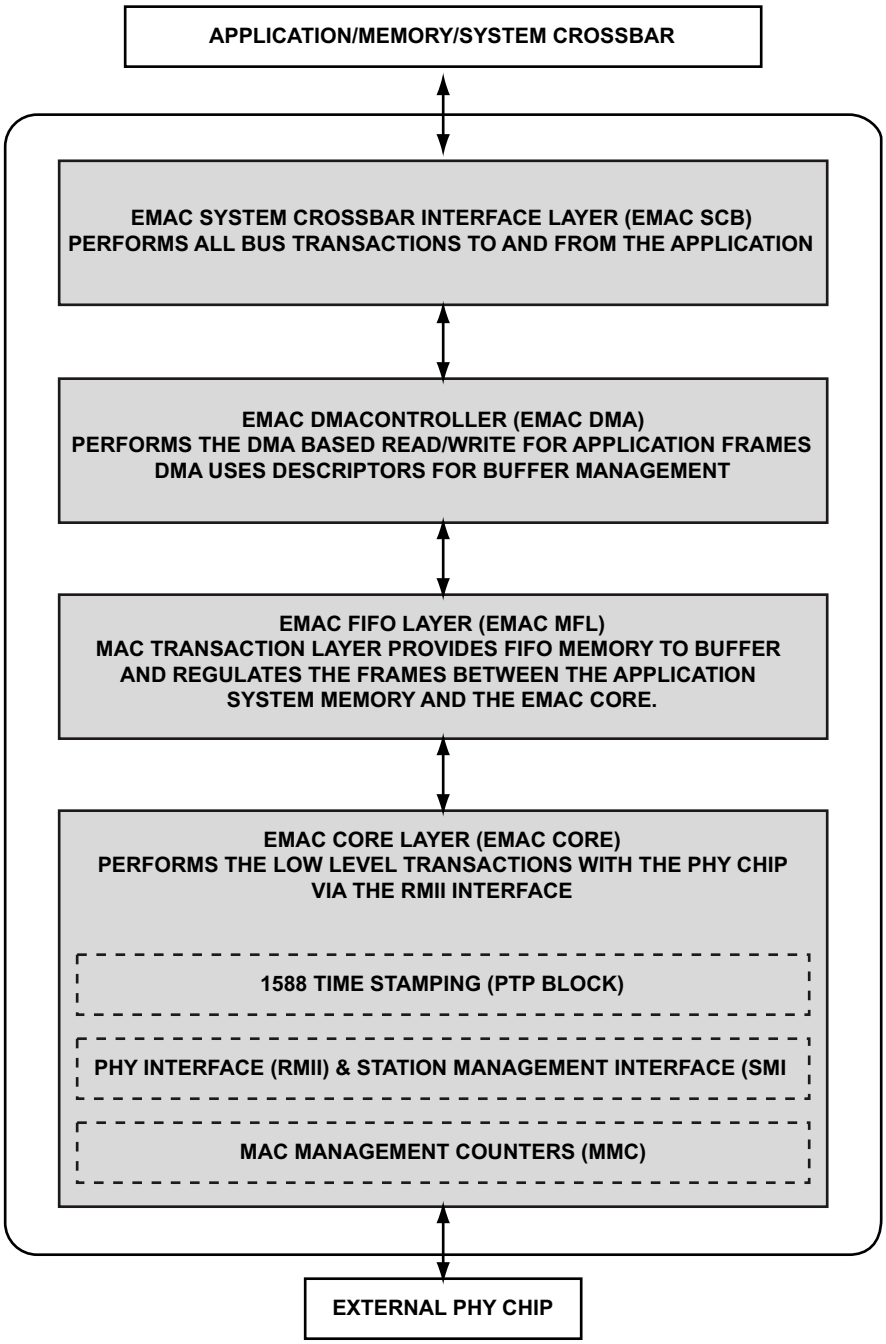


Figure 23-1: EMAC Simplified Block Diagram

A more comprehensive block diagram is shown below. It includes most of the important blocks inside the EMAC. The EMAC is connected to processor memory and the system crossbar via the System Crossbar Bus Interface (SCB) and System Peripheral Bus Interface (SPB), which are part of the SCB Layer. The SPB interface is connected to all modules that require MMR programming.

The DMA controller performs application data transfer frame by frame, through well defined descriptor structures. A FIFO layer acts as a buffer between the DMA controller and the EMAC core. The EMAC core is the most important block as it contains sub-blocks to support IEEE802.3 based communication with external network interfaces of 10/100 Mbps speeds. It includes the PTP sub-block which assists applications requiring time synchronization and a MMC sub-block to generate packet transfer statistics. The MAC is connected to the external PHY via the Reduced Media Independent Interface (RMII) and the Station Management Interface (Serial Management IO).

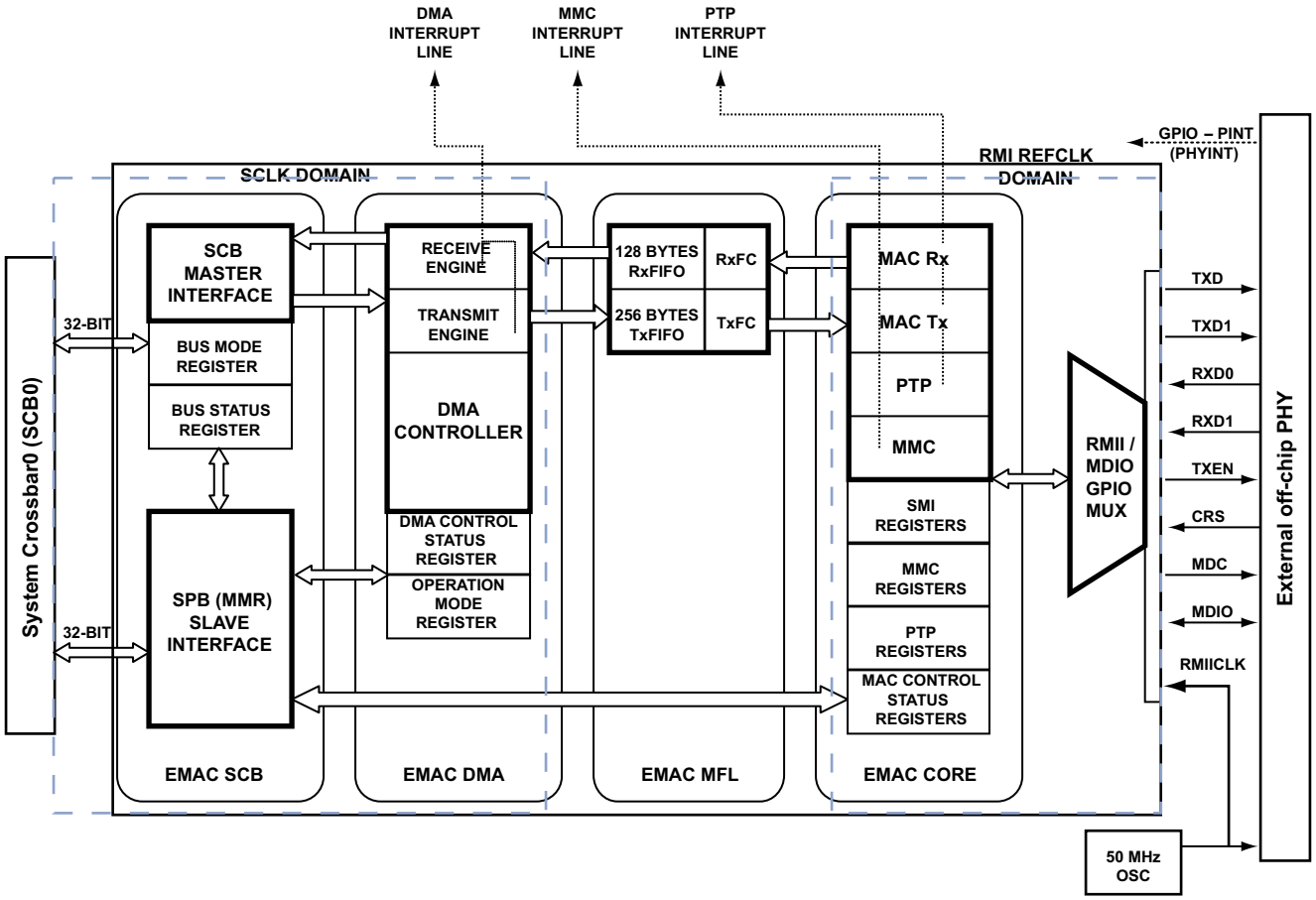


Figure 23-2: EMAC Complete Block Diagram

EMAC CORE Sub-Blocks

The core transmit engine sub-blocks and their function are summarized in the below table. Please refer to the EMAC core section for further explanation of each of these sub-blocks.

Table 23-5: Core Transmit Engine sub-blocks

CORE Transmit Engine Sub Block	Function
Transmit Bus Interface	Interface to the FIFO.

Table 23-5: Core Transmit Engine sub-blocks (Continued)

CORE Transmit Engine Sub Block	Function
Transmit Frame Controller	–Appends Zero-PAD data if required, for short frames. –Appends CRC for Frame Check-Sum from the CRC Generator.
Transmit Protocol Engine	–Generates preamble and SFD, as per 802.3 protocol. –Generates jam pattern in Half-Duplex mode, for collisions. –Jabber timeout, for excessively large frames. –Flow control for Half-Duplex mode (back pressure). –Generates transmit frame status.
Transmit Scheduler	–Maintains the inter-frame gap between two transmitted frames. –Follows the Truncated Binary Exponential Back-off algorithm for Half-Duplex mode.
Transmit CRC Generator	Generate CRC for the Frame Check-Sum field of the Ethernet frame.
Transmit Flow Control	Receives the Pause frame, appends the calculated CRC, and sends the frame to the Protocol Engine module.
Transmit Checksum Offload Engine	Supports checksum calculation and insertion in the transmit path, for IPV4/TCP/UDP/ICMP packets.

The core receive engine sub-blocks and their function are summarized in the following table. Please refer to the EMAC core section for more information on each of these sub-blocks.

Table 23-6: Core Receive Engine Sub-Blocks

CORE Receive Engine sub block	Functionality overview
Receive Protocol Engine	–Strips the incoming preamble and SFD. –Checks for correct Length/Type field. –Performs internal loopback if required. –Generates receive status. –Supports watchdog of received frames. –Supports Jumbo Frames.
Receive CRC Module	Checks for CRC error, by comparing with FCS.

Table 23-6: Core Receive Engine Sub-Blocks (Continued)

CORE Receive Engine sub block	Functionality overview
Receive Frame Controller Module	–Packs incoming 8-bit input stream to 32-bit data internally. –Performs Frame filtering, for uni-cast/multi-cast/ broadcast frames. –Attaches the calculated IP Checksum input from Checksum Offload Engine. –Updates the Receive Status to Bus Interface.
Receive Flow Control Module	–Detects the receiving Pause frame and pauses the frame transmission for the delay specified within the received Pause frame. –Works in Full Duplex mode.
Receive IP Checksum Offload Engine	–Calculates IPv4 header checksums and verify against the received IPv4 header checksums. –Identifies a TCP, UDP or ICMP payload in the received IP datagrams.
Receive Bus Interface Unit Module	Interface to the FIFO.
Address Filtering Module	Performs Destination Address Filtering based on Unicast/ Multi-cast/Broadcast frames. –Provides CRC hash filtering.

EMAC PHY Interface

The EMAC can interface to the PHY via the RMII interface standard. The tables below indicate the RMII pins available in the EMAC in terms of their generic names. Please refer to the data sheet for exact pin names.

Table 23-7: RMII Pins

Sl. No.	Generic Signal Name (IEEE Standards)	RMII Pin functionality.
1.	TXD0	RMII transmit data pin D0 (di-bit lower)
2.	TXD1	RMII transmit data pin D1 (di-bit higher)
3.	RXD0	RMII receive data pin D0 (di-bit lower)
4.	RXD1	RMII receive data pin D1 (di-bit higher)
5.	RMII CLK	RMII common clock (for Tx and Rx), also called reference clock
6.	TXEN	RMII transmit enable pin (Tx valid)
7.	CRS	RMII Carrier Sense / receive data valid
8.	MDC	Serial management clock driven by EMAC

Table 23-7: RMII Pins (Continued)

Sl. No.	Generic Signal Name (IEEE Standards)	RMII Pin functionality.
9.	MDIO	Serial management bi-directional data

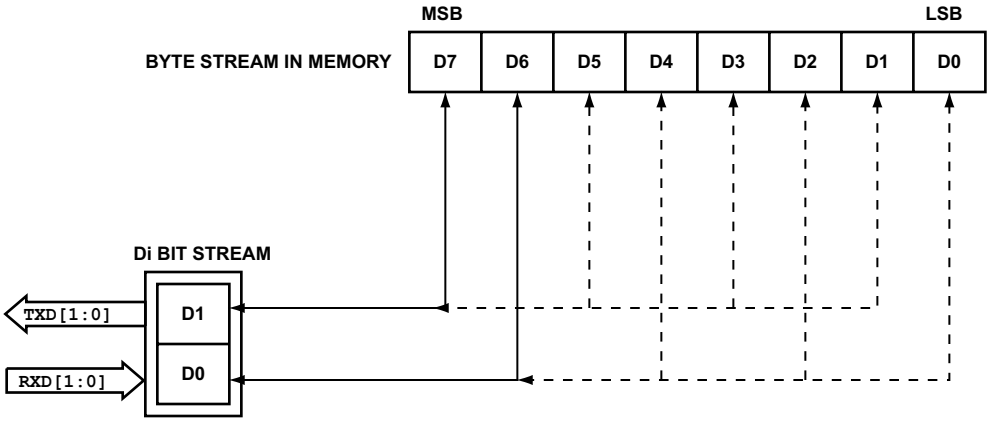


Figure 23-3: RMII Di-bit Data Transfer

Clock Sources

The Ethernet MAC is clocked internally from *SCLK*. Check the processor data sheet for the valid frequency range of the appropriate *SCLK*signal for Ethernet operation.

A 50 MHz clock should be sourced externally to operate the EMAC in RMII mode. This clock is the same for both transmit and receive. The MDC Station Management Clock is derived from the *SCLK*and driven from the MAC to the PHY, when accessing any PHY registers.

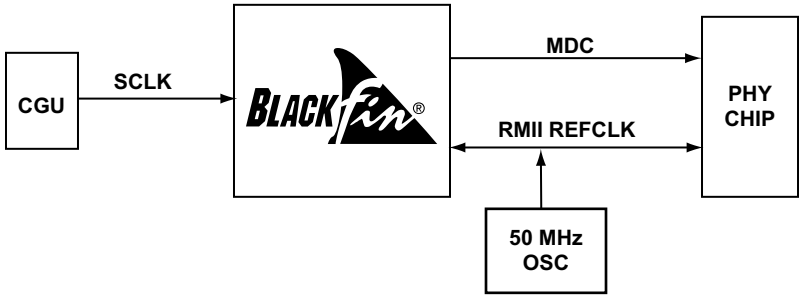


Figure 23-4: EMAC Clock Sources

EMAC Architectural Concepts

This section explains different architectural concepts relevant to EMAC peripheral, such as EMAC SCB, EMAC DMA, EMAC MFL, EMAC CORE and others.

EMAC System Crossbar Interface (EMAC SCB)

The EMAC SCB bus interface provides the bus connectivity to support highly effective data traffic throughput. System bus use is maximized by allowing simultaneous read and write transfers initiated from different DMA channels. The EMAC controller is directly connected to the SCB0 crossbar. The following interfaces are available with the design.

- A 32-bit SCB master interface for reading/writing from/to application memory.
- A 32-bit SPB slave interface for register programming.

Please refer to the "System Crossbars (SCB)" chapter for more information on how the crossbar operates. Only the EMAC specific information is detailed in this chapter.

Table 23-8: EMAC-SCB Interface Data Transfer Specifications with Crossbar

Specification Term	Comments
1 beat in SCB	SINGLE burst
BLEN4 bursts	4 beats in SCB
BLEN8 bursts	8 beats in SCB
BLEN16 bursts	16 beats in SCB
Bus size	32-bit fixed bus size; equals 1 beat
INCR bursts	Incrementing Bursts
INCR ALIGNED bursts	Incrementing aligned bursts
UNDEF bursts	Undefined burst length
PBL	Programmable Burst Length for DMA

The DMA write channel and read channel data paths and their connection to the system crossbar is shown below.

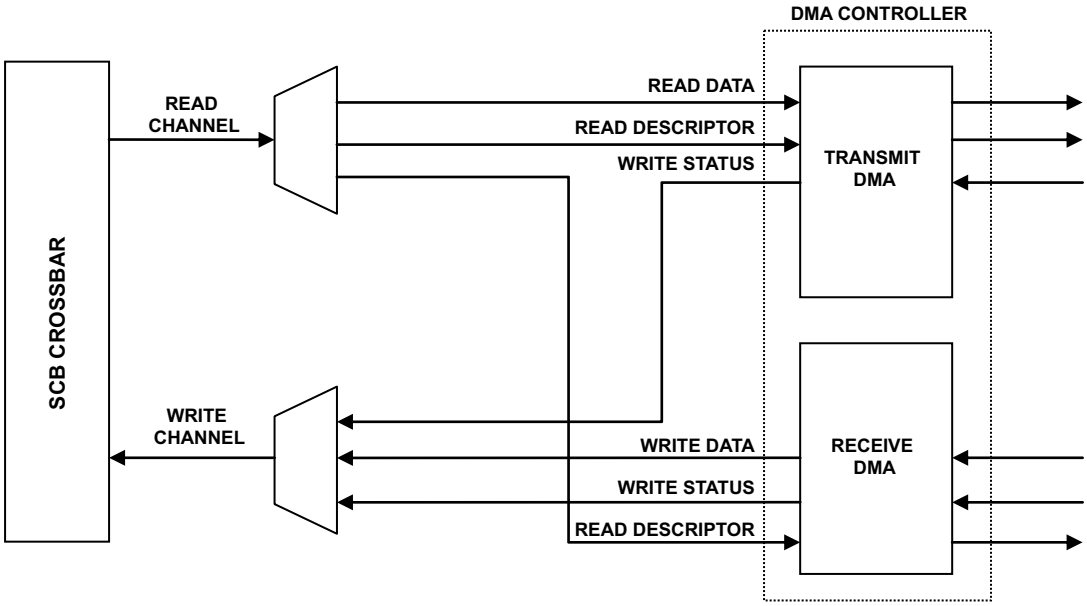


Figure 23-5: EMAC DMA Read/Write channels with System Crossbar

NOTE: Transmit descriptor read and receive descriptor write-back (status update) operations can occur simultaneously. However transmit descriptor read and transmit descriptor write-back operations cannot occur simultaneously because transmit DMA (or receive DMA) does not initiate the next transfer unless the previous one is complete.

Priority of SCB Requests

The descriptor transfers have higher priority than the data transfers. For example, if there are two bus requests—a receive descriptor read and a transmit data read—the receive descriptor read has a higher priority so that the next receive data write (subsequent to the receive descriptor read) need not wait for the transmit data read transfer to complete.

If there are descriptor read requests from both DMA channels, they are serviced based on a first-come first-serve. Receive DMA has higher priority if descriptor read requests are generated from both the DMA channels in the same clock cycle. Similarly, in the write channel, descriptor writes from DMA have higher priority than the data-write transfers for the receive DMA.

SCB Interface Programming Options

The SCB bus interface supports the following programmable options for the EMAC module. These options are available using the `EMAC_DMA_BMMODE` register with the `EMAC_DMA_BUSMODE` register.

- **Outstanding transactions.** The EMAC-SCB supports up to four outstanding read/write requests on the SCB bus. This can also be controlled through software by programming the `EMAC_DMA_BMMODE`.

WROSRLMT and EMAC_DMA_BMMODE.RDOSRLMT bits. Maximum outstanding requests=EMAC_DMA_BMMODE.WROSRLMT + 1 (or) EMAC_DMA_BMMODE.RDOSRLMT + 1.

- **Allowed burst sizes.** The allowed burst sizes are 4 (EMAC_DMA_BMMODE.BLEN4), 8 (EMAC_DMA_BMMODE.BLEN8), 16 (EMAC_DMA_BMMODE.BLEN16) and the SINGLE burst. Only those burst sizes configured by the program (via the EMAC_DMA_BMMODEregister) are used for data transfer through the SCB bus. However, SINGLE burst is available by default, when the EMAC_DMA_BMMODE.UNDEF bit in the is cleared. Data transfers are restricted to the maximum burst size from this list of programmed burst sizes.
- **Burst splitting and burst selection.** The EMAC-SCB splits the DMA requests into multiple bursts on the SCB system bus. Splitting is based on DMA count and software controllable burst enable bits (shown in the Allowed burst sizes) as well as burst types (INCR and INCR_ALIGNED) which are also controllable through the software. SINGLE burst is enabled when the EMAC_DMA_BMMODE.UNDEF bit is not set. Burst length select priority is in the sequence: UNDEF, 16, 8, and 4.
- **INCR burst type**
  - If the EMAC_DMA_BMMODE.UNDEF bit is set, then the EMAC-SCB always chooses the maximum allowed burst length based on the EMAC_DMA_BMMODE.BLEN16, EMAC_DMA_BMMODE.BLEN8, EMAC_DMA_BMMODE.BLEN4 bits. In cases where the DMA requests are not multiples of the maximum allowed burst length, the SCB may also choose a burst-length of any value less than the maximum enabled burst-length (all lesser burst-length enables are redundant). For example, when length bits are enabled and the DMA requests a burst transfer size of 42 beats, then the SCB splits it into three bursts of 16, 16 and 10 beats respectively.
  - If EMAC_DMA_BMMODE.UNDEF is not enabled, then the burst length is based on the priority of the enabled bits in the following order EMAC_DMA_BMMODE.BLEN16, EMAC_DMA_BMMODE.BLEN8, EMAC_DMA_BMMODE.BLEN4. When the DMA requests a burst transfer, the SCB interface splits the requested bursts into multiple transfers using only the enabled burst lengths. This splitting can occur when the requested burst is not a multiple of the maximum enabled burst. If it cannot choose any of the enabled burst lengths then it selects the burst length as 1.

  For example, when EMAC_DMA_BMMODE.BLEN16, EMAC_DMA_BMMODE.BLEN8, EMAC_DMA_BMMODE.BLEN4 are enabled and the DMA requests a burst transfer of 42 beats, then the SCB interface splits it into multiple bursts of size 16, 16, 8, 1 and 1 beats respectively (the sequence is in decreasing burst sizes).
- **INCR_ALIGNED burst type.** When the address-aligned burst-type is enabled (EMAC_DMA_BMMODE.AAL), then in addition to the burst splitting conditions explained in the INCR Burst type, the SCB interface splits the DMA requested bursts such that each burst-size is aligned to the least significant bits of the start address. The SCB interface initially generates smaller bursts so that the remaining transfers can be transferred with the maximum possible (enabled) fixed burst lengths.

  For example, in the same setting as explained earlier for EMAC_DMA_BMMODE.UNDEF set (EMAC_DMA_BMMODE.BLEN16, EMAC_DMA_BMMODE.BLEN8, and EMAC_DMA_BMMODE.BLEN4 are enabled), DMA requests a burst size of 42 beats at the start address of 0x000003A4. The SCB starts the first transfer with

size 3 such that the address of the next burst is aligned (0x000003B0) for a burst of 16. Therefore, the sequence of bursts is 3, 16, 16, and 7, respectively.

When EMAC_DMA_BMMODE.UNDEF is not set, then in the same situation (start address of 0x000003A4 with 42 beats), the sequence of burst transfers is 1, 1, 1, 16, 16, 4, and 3 respectively. The sequence of smaller bursts at the beginning is to align the address to the next higher enabled burst-lengths programmed in the register.

- **Burst operations for DMA transactions.** The EMAC_DMA_BUSMODE.PBL (programmable burst length) field indicates the maximum number of beats to be transferred in one DMA transaction. This is also the maximum value that is used in a single block read/write and is shown in the following table.
  - For example, if EMAC_DMA_BUSMODE.PBL=32 and if EMAC_DMA_BMMODE.BLEN16 is enabled, the DMA automatically splits 32 bursts in to 2 x 16 bursts. If EMAC_DMA_BUSMODE.PBL=8, and if EMAC_DMA_BMMODE.BLEN16 and EMAC_DMA_BMMODE.BLEN8 are enabled, the maximum burst is limited to EMAC_DMA_BMMODE.BLEN8. If the EMAC_DMA_BUSMODE.PBL8 bit is set, the programmed EMAC_DMA_BUSMODE.PBL value is multiplied by 8 times internally. However, the result cannot be more than the maximum limits specified above.
  - The receive DMA burst length configuration can be made independent of transmit DMA configuration, by setting the EMAC_DMA_BUSMODE.USP bit. If this bit is set, the EMAC_DMA_BUSMODE.RPBL bits define the burst length of receive DMA. If the EMAC_DMA_BUSMODE.USP bit is not set, the EMAC_DMA_BUSMODE.RPBL bits are used for both transmit and receive. Programs must ensure that the PBL maximum limit is not violated.
  - The receive and transmit descriptors are always accessed in the maximum possible burst-size (limited by PBL maximum for transmit and receive) for the 16-bytes to be read.

Table 23-9: DMA PBL Max Limits

Burst Limit Max Term	Definition
PBL-max limit	(FIFO size/2)/4 words.
PBL-max limit (transmit)	256 bytes/2 /4 = 32 words.
PBL-max limit (receive)	128 bytes/2 /4 = 16 words.

DMA Bursts Using the SCB Interface

The transmit DMA initiates a data transfer only when sufficient space to accommodate the configured burst is available in the transmit FIFO or the number of bytes until the end of frame (when it is less than the configured burst-length). The DMA indicates the start address and the number of transfers required to the SCB master interface. When the SCB interface is configured for fixed-length burst, then it transfers data using the best combination of INCR4/8/16 and 1 beat transaction.

The receive DMA initiates a data transfer only when sufficient data to accommodate the configured burst is available in the MTL receive FIFO or when the end of frame (when it is less than the configured burst-length) is detected in the receive FIFO. The DMA indicates the start address and the number of transfers required to the SCB master interface. When the SCB interface is configured for fixed-length burst, then it

transfers data using the best combination of INCR 4/8/16 or 1 beat transaction. If the end-of frame is reached before the fixed-burst ends on the SCB interface, then dummy transfers are performed in-order to complete the fixed-burst. Otherwise (if EMAC_DMA_BUSMODE.FBis reset), it transfers data using INCR (undefined length) transactions.

When the SCB interface is configured for address-aligned beats using the EMAC_DMA_BUSMODE.AAL bit, both DMA engines ensure that the first burst transfer the SCB initiates is less than or equal to the size of the configured PBL. Therefore, all subsequent beats start at an address that is aligned to the configured PBL.

SCB Bus Transaction Status

The EMAC_DMA_BMSTAT.BUSRD and EMAC_DMA_BMSTAT.BUSWR bits indicate whether the channel is active or not.

Fatal Bus Error

The EMAC SCB asserts the error interrupt (EMAC_DMA_STAT.FBI) when the corresponding fatal bus error interrupt is enabled in the DMA interrupt enable register. The application has to reset the core to restart the DMA.

DMA Controller (EMAC DMA)

The EMAC has an built-in DMA controller that performs reads and writes of application data and descriptors via the SCB master interface.

The DMA controller has independent transmit and receive engines, and a CSR (control and status register) space. The transmit engine transfers data from system memory to a FIFO, while the receive engine transfers data from the FIFO to the system memory. The controller uses a descriptor chain based transfer mechanism to efficiently move data from source to destination with minimal processor core intervention. The DMA is specially designed for packet-oriented data transfers such as Ethernet frames. The controller can be programmed to interrupt the application for situations such as frame transmit and receive transfer completion, and other normal or abnormal conditions.

The DMA and the application device driver communicate through two internal data structures:

1. DMA control and status registers (CSR).
2. Data buffers and descriptor lists. Descriptor list operate in ring mode and chain mode, as shown in the following figure.

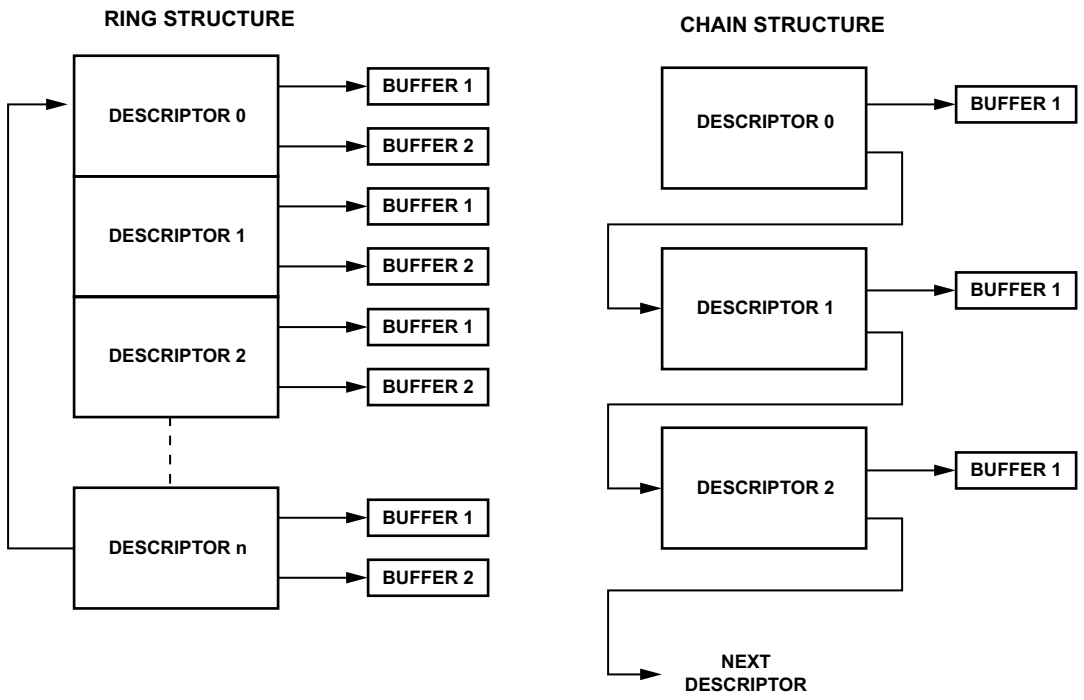


Figure 23-6: EMAC DMA Descriptor Models

Descriptors that reside in the application memory act as pointers to receive and transmit buffers. Descriptors have the following additional attributes.

- There are two descriptor lists, one for receive, and one for transmit. The base address of each list is written into the receive descriptor list address register and transmit descriptor list address register, respectively.
- A descriptor list is forward linked (either implicitly or explicitly). The last descriptor may point back to the first entry to create a ring structure.
- Explicit chaining of descriptors is accomplished by setting the second address chained in both receive and transmit descriptors.
- The descriptor lists reside in the application memory address space.
- Each descriptor can point to a maximum of two buffers. This enables two buffers to be used, physically addressed, rather than contiguous buffers in memory.

A data buffer resides in the application physical memory space and consists of an entire frame or part of a frame, but cannot exceed a single frame. Buffers may contain only data while buffer status is maintained in the descriptor itself. *Data chaining* refers to frames that span multiple data buffers. However, a single descriptor cannot span multiple frames. The DMA skips to the next frame buffer when the end-of-frame is detected. Data chaining can be enabled or disabled.

NOTE: It is possible to define a skip length (in terms of N × 32-bit words) between two subsequent descriptors, when using ring mode. The `EMAC_DMA_BUSMODE.DSL` field must be programmed to enable

this. With this option available, programs are not always restricted to a contiguous memory location in ring mode.

DMA Related Registers

A summary of DMA registers relative to their function is provided in the table below. Please refer to the "Register Descriptions" sections for complete bit descriptions of each of these registers.

Table 23-10: Summary of DMA Related Registers.

Register Name	Description
Bus Mode[1]	Establishes the bus operating modes for the DMA with respect to the SCB master interface.
Transmit Poll Demand	Enables the transmit DMA to check whether or not the current descriptor is owned by DMA. The transmit poll demand command is given to wake up the TxDMA if it is in suspend mode. The TxDMA can go into suspend mode because of an underflow error in a transmitted frame or because of the unavailability of descriptors owned by transmit DMA. This command can be issued anytime and the TxDMA resets this command once it starts re-fetching the current descriptor from host memory.
Receive Poll Demand	Enables the receive DMA to check for new descriptors. This command is given to wake-up the RxDMA from the SUSPEND state. The RxDMA can go into SUSPEND state only because of the unavailability of descriptors owned by it.
Receive Descriptor List Address	Points to the start of the receive descriptor list. The descriptor lists reside in the application memory space and must be word-aligned (32-bit data bus). The DMA internally converts the descriptor list to a bus width aligned address by making the corresponding LSBs low.
Transmit Descriptor List Address	Points to the start of the transmit descriptor list. The descriptor lists reside in the application memory space and must be word-aligned (for 32-bit data bus). The DMA internally converts it to bus width aligned address by making the corresponding LSB to low.
DMA Status	Contains all the status bits that the DMA reports to the application. The software driver reads this register during an interrupt service routine or during polling. Most of the fields in this register cause the host to be interrupted.
Operation Mode	Establishes the transmit and receive operating modes and commands. The operation mode register should be the last control register to be written as part of DMA initialization.
Interrupt Enable	Enables the interrupts reported by DMA status register. After a hardware or software reset, all interrupts are disabled.

Table 23-10: Summary of DMA Related Registers. (Continued)

Register Name	Description
Missed Frame and Buffer Overflow Counter	The DMA maintains two counters to track the number of missed frames during reception. This register reports the current value of the counter, which is used for diagnostic purposes.
Receive Interrupt Watchdog Timer	When written with non-zero value, enables the watchdog timer for receive interrupt (RI) in the DMA status register.
SCB Bus Mode	Controls the SCB interface master behavior. It is mainly used to control the burst splitting and the number of outstanding requests.
SCB Status	Provides the active status of the SCB interface read and write channels.
Current Host Transmit Descriptor	Points to the start address of the current transmit descriptor read by the DMA.
Current Host Receive Descriptor	Points to the start address of the current receive descriptor read by the DMA.
Current Host Transmit Buffer Address	Points to the current transmit buffer address being read by the DMA.
Current Host Receive Buffer Address	Points to the current receive buffer address being read by the DMA.

1.There should not be any further writes to the EMAC_DMA_BUSMODE registers until the first write is updated. Otherwise, the second write operation will not get updated properly. For correct operation, the delay between two writes to the same register location should be at least 8 cycles of 50 MHz RMII REFCLK.

Table 23-11: DMA Registers with Consecutive Writes

Registers with Implications for Consecutive Writes
DMA Bus Mode

DMA Descriptors

The DMA module in the Ethernet subsystem transfers data based on a linked list of descriptors. The descriptor addresses must be aligned to the 32-bit bus width. The descriptors can be either 4 x 32-bit words (16 bytes) or 8 x 32-bit words (32 bytes). The controller needs to be configured for the appropriate word length using the EMAC_DMA_BUSMODE register. Descriptor words are numbered from 0 to 7 for both the transmit and receive engine.

Typical factors for deciding the descriptor word size are as follows.

- When the time-stamping or receive checksum engines are not enabled, the extended descriptors are not required and the software can use descriptors with the default size of 16 bytes (4 words).
- When the time-stamping feature is enabled (to be used with the IEEE 1588 PTP engine), the software needs to allocate 32-bytes (8 words) of memory for every descriptor.

- When only the receive checksum off-load is enabled (time-stamping disabled) the software needs to allocate 32-bytes (8 words) of memory for every descriptor, although in reality only word 4 of the extended words (descriptors 4–7) contain the required status information. The rest of extended words may be treated as reserved or dummy.

Transmit Descriptor

The transmit descriptor structure in memory is shown in the following figure. The application software must program the TDES0 control bits during descriptor initialization. When the DMA updates the descriptor, it writes back all the control bits except the OWN bit (which it clears) and updates the status bits. The contents of the transmitter descriptor word 0 (TDES0) through word 7 (TDES7) are given in the following tables.

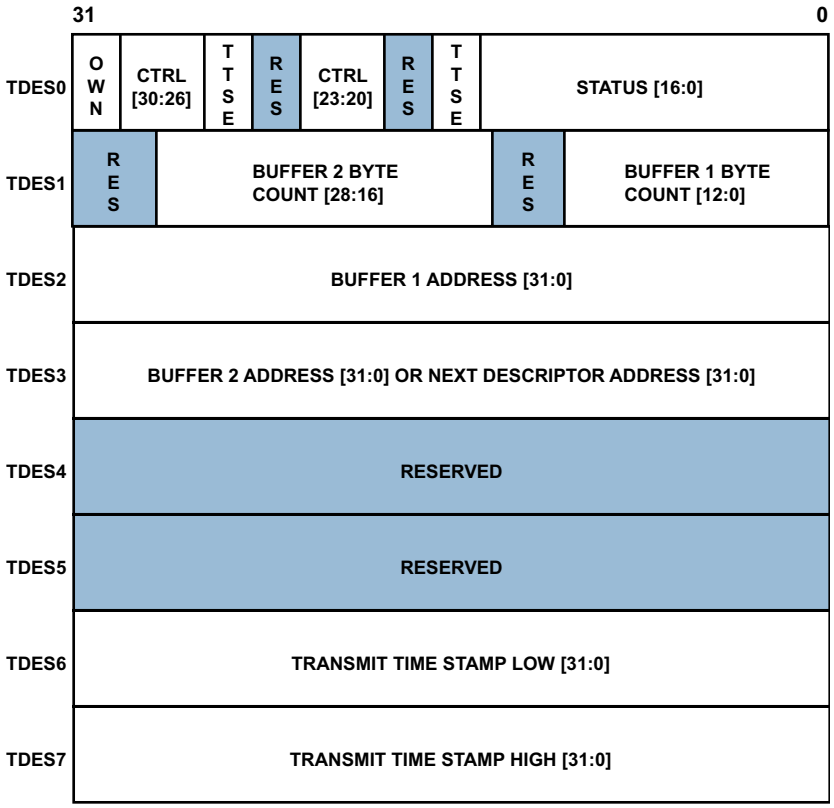


Figure 23-7: Transmit Descriptor Words

Table 23-12: Transmit Descriptor Fields (TDES0)

Bit	Name	Description
31	OWN	When set, this bit indicates that the descriptor is owned by the DMA. When this bit is reset, it indicates that the descriptor is owned by the application. The DMA clears this bit either when it completes the frame transmission or when the buffers allocated in the descriptor are read completely. The ownership bit of the frame's first descriptor must be set after all subsequent descriptors belonging to the same frame have been set. This avoids a possible race condition between fetching a descriptor and the driver setting an ownership bit.
30	IC	Interrupt on Completion. When set, this bit sets the transmit interrupt (DMA status register [0]) after the present frame has been transmitted.
29	LS	Last Segment. When set, this bit indicates that the buffer contains the last segment of the frame
28	FS	First Segment. When set, this bit indicates that the buffer contains the first segment of a frame.
27	DC	Disable CRC. When this bit is set, the EMAC does not append a cyclic redundancy check (CRC) to the end of the transmitted frame. This is valid only when the first segment (TDES0[28]) is set.
26	DP	Disable Pad. When set, the EMAC does not automatically add padding to a frame shorter than 64 bytes. When this bit is reset, the DMA automatically adds padding and CRC to a frame shorter than 64 bytes, and the CRC field is added despite the state of the DC (TDES0[27]) bit. This is valid only when the first segment (TDES0[28]) is set.
25	TTSE	Transmit Time-Stamp Enable. When set, this bit enables IEEE1588 hardware time-stamping for the transmit frame referenced by the descriptor. This field is valid only when the first segment control bit (TDES0[28]) is set.
24		Reserved
23:22	CIC	Checksum Insertion Control. These bits control the checksum calculation and insertion. Bit encodings are shown below. 00 = Checksum Insertion Disabled. 01 = Only IP header checksum calculation and insertion are enabled. 10 = IP header checksum and payload checksum calculation and insertion are enabled, but pseudo-header checksum is not calculated in hardware. 11 = IP Header checksum and payload checksum calculation and insertion are enabled, and pseudo-header checksum is calculated in hardware.
21	TER	Transmit End of Ring. When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a descriptor ring.

Table 23-12: Transmit Descriptor Fields (TDES0) (Continued)

Bit	Name	Description
20	TCH	Second Address Chained. When set, this bit indicates that the second address in the descriptor is the next descriptor address rather than the second buffer address. When TDES0[20] is set, TBS2 (TDES1[28:16]) is a *don't care* value. TDES0[21] takes precedence over TDES0[20].
19:18	Reserved	
17	TTSS	Transmit Time Stamp Status. This bit is used as a status bit to indicate that a time-stamp was captured for the described transmit frame. When this bit is set, TDES2 and TDES3 have a time-stamp value captured for the transmit frame. This field is only valid when the descriptor's Last Segment control bit (TDES0[29]) is set.
16	IHE	IP Header Error. When set, this bit indicates that the EMAC transmitter detected an error in the IP datagram header. The transmitter checks the header length in the IPv4 packet against the number of header bytes received from the application and indicates an error status if there is a mismatch. For IPv6 frames, a header error is reported if the main header length is not 40 bytes. Furthermore, the Ethernet Length/Type field value for an IPv4 or IPv6 frame must match the IP header version received with the packet. For IPv4 frames, an error status is also indicated if the Header Length field has a value less than 0x5.
15	ES	Error Summary. Indicates the logical OR of the following bits. TDES0[14] = Jabber Timeout TDES0[13] = Frame Flush TDES0[11] = Loss of Carrier TDES0[10] = No Carrier TDES0[9] = Late Collision TDES0[8] = Excessive Collision TDES0[2] = Excessive Deferral TDES0[1] = Underflow Error TDES0[16] = IP Header Error TDES0[12] = IP Payload Error
14	JT	Jabber Timeout. When set, this bit indicates the EMAC transmitter has experienced a jabber time-out. This bit is only set when the EMAC configuration register's JD bit is not set.
13	FF	Frame Flushed. When set, this bit indicates that the DMA/MFL flushed the frame due to a software Flush command given by the CPU.
12	IPE	IP Payload Error. When set, this bit indicates that EMAC transmitter detected an error in the TCP, UDP, or ICMP IP datagram payload.The transmitter checks the payload length received in the IPv4 or IPv6 header against the actual number of TCP, UDP, or ICMP packet bytes received from the application and issues an error status in case of a mismatch.

Table 23-12: Transmit Descriptor Fields (TDES0) (Continued)

Bit	Name	Description
11	LC	Loss of Carrier. When set, this bit indicates that a loss of carrier occurred during frame transmission. This is valid only for the frames transmitted without collision when the EMAC operates in Half-Duplex mode.
10	NC	No Carrier. When set, this bit indicates that the Carrier Sense signal form the PHY was not asserted during transmission.
9	LC	Late Collision. When set, this bit indicates that frame transmission was aborted due to a collision occurring after the collision window (64 byte-times, including preamble). This bit is not valid if the Underflow Error bit is set.
8	EC	Excessive Collision. When set, this bit indicates that the transmission was aborted after 16 successive collisions while attempting to transmit the current frame. If the DR (Disable Retry) bit in the EMAC configuration register is set, this bit is set after the first collision, and the transmission of the frame is aborted.
7	VF	VLAN Frame. When set, this bit indicates that the transmitted frame was a VLAN-type frame.
6:3	CC	Collision Count. This 4-bit counter value indicates the number of collisions occurring before the frame was transmitted. The count is not valid when the Excessive Collisions bit (TDES0[8]) is set.
2	ED	Excessive Deferral. When set, this bit indicates that the transmission has ended because of excessive deferral of over 24,288 bit times (155,680 bits times in 1,000-Mbps mode or if Jumbo Frame is enabled) if the Deferral Check (DC) bit in the EMAC control register is set high.
1	UF	Underflow Error. When set, this bit indicates that the EMAC aborted the frame because data arrived late from the application memory. Underflow Error indicates that the DMA encountered an empty transmit buffer while transmitting the frame. The transmission process enters the suspended state and sets both Transmit Underflow (register 5[5]) and Transmit Interrupt (register 5[0]).
0	DB: Deferred Bit	When set, this bit indicates that the EMAC defers before transmission because of the presence of carrier. This bit is valid only in half-duplex mode.

Table 23-13: Transmit Descriptor Word 1 (TDES1)

Bit	Name	Description
31–29	Reserved	
28–16	TBS2	Transmit Buffer 2 Size. These bits indicate the second data buffer size in bytes. This field is not valid if TDES0[20] is set.
15–13	Reserved	

Table 23-13: Transmit Descriptor Word 1 (TDES1) (Continued)

Bit	Name	Description
12–0	TBS1	Transmit Buffer 1 Size. These bits indicate the first data buffer byte size, in bytes. If this field is 0, the DMA ignores this buffer and uses buffer 2 or the next descriptor, depending on the value of TCH (TDES0[20]).

Table 23-14: Transmit Descriptor 2 (TDES2)

Bit	Name	Description
31–0	Buffer 1 Address Pointer	These bits indicate the physical address of buffer 1. There is no limitation on the buffer address alignment

Table 23-15: Transmit Descriptor 3 (TDES3)

Bit	Name	Description
31–0	Buffer 2 Address Pointer (Next Descriptor Address)	Indicates the physical address of buffer 2 when a descriptor ring structure is used. If the second address chained (TDES1[24]) bit is set, this address contains the pointer to the physical memory where the next descriptor is present. The buffer address pointer must be aligned to the bus width only when TDES1[24] is set. (LSBs are ignored internally.)

Table 23-16: Transmit Descriptor 6 (TDES6)

Bit	Name	Description
31–0	TTSL	Transmit Frame Time Stamp Low. This field is updated by DMA with the least significant 32 bits of the time-stamp captured for the corresponding transmit frame. This field has the time-stamp only if the Last Segment bit (LS) in the descriptor is set and time-stamp status (TTSS) bit is set.

Table 23-17: Transmit Descriptor 7 (TDES7)

Bit	Name	Description
31–0	TTSH	Transmit Frame Time Stamp High. This field is updated by DMA with the most significant 32 bits of the time-stamp captured for the corresponding receive frame. This field has the time-stamp only if the last segment bit (LS) in the descriptor is set and time-stamp status (TTSS) bit is set.

DMA Transmit Process

The following sections describe how the direct memory access transmit process works on the EMAC controller.

- *Default (Non-OSF) Mode*
- *OSF Mode Enabled*
- *Transmit Frame Processing*
- *Transmit Polling Suspended*

Default (Non-OSF) Mode

The default process for DMA transmit works as follows:

1. The application sets up the transmit descriptor (using TDES0- TDES3) and sets the OWN bit (TDES0) after setting up the corresponding data buffer(s) with Ethernet frame data.
2. Once the EMAC_DMA_OPMODE.ST bit is set, the DMA enters the run state.
3. While in the run state, the DMA polls the transmit descriptor list for frames requiring transmission. After polling starts, it continues in either sequential descriptor ring order or chained order. If the DMA detects a descriptor flagged as owned by the application, or if an error condition occurs, transmission is suspended and both the transmit buffer unavailable (EMAC_DMA_STAT.TU) and normal interrupt summary (EMAC_DMA_STAT.NIS) bits are set. The transmit engine proceeds to Step 9.
4. If the acquired descriptor is flagged as owned by DMA (TDES0 [31] = 1#b1), the DMA decodes the transmit data buffer address from the acquired descriptor.
5. The DMA fetches the transmit data from the application memory and transfers the data to the MFL for transmission.
6. If an Ethernet frame is stored over data buffers in multiple descriptors, the DMA closes the intermediate descriptor and fetches the next descriptor. Steps 3, 4, and 5 are repeated until the end-of-Ethernet-frame data is transferred to the MFL.
7. When frame transmission is complete, if IEEE 1588 time-stamping was enabled for the frame (as indicated in the transmit status) the time-stamp value obtained from MFL is written to the transmit descriptor (TDES2 and TDES3) that contains the end-of-frame buffer. The status information is then written to this transmit descriptor (TDES0). Because the OWN bit is cleared during this step, the application now owns this descriptor. If time-stamping was not enabled for this frame, the DMA does not alter the contents of TDES2 and TDES3.
8. Transmit interrupt (EMAC_DMA_STAT.TI) is set after completing transmission of a frame that has interrupt on completion (TDES1 [31]) set in its last descriptor. The DMA engine then returns to Step 3.
9. In the suspend state, the DMA tries to re-acquire the descriptor (and thereby return to Step 3) when it receives a transmit poll demand and the EMAC_DMA_STAT.UNF bit is cleared.

NOTE: If the EMAC_DMA_OPMODE.OSF bit is not set, the actual Inter Frame Gap (IFG) may be seen as more than as programmed in the EMAC_MACCFG register.

OSF Mode Enabled

While in the run state, the transmit process can simultaneously acquire two frames without closing the status descriptor of the first (if the EMAC_DMA_OPMODE.OSF bit is set). As the transmit process finishes transferring the first frame, it immediately polls the transmit descriptor list for the second frame. If the second frame is valid, the transmit process transfers this frame before writing the first frame status information.

In OSF mode, the run state transmit DMA operates in the following sequence.

1. The DMA operates as described in steps 1–6 of *Default (Non-OSF) Mode*.
2. Without closing the previous frame's last descriptor, the DMA fetches the next descriptor.
3. If the DMA owns the acquired descriptor, the DMA decodes the transmit buffer address in this descriptor. If the DMA does not own the descriptor, the DMA goes into suspend mode and skips to Step 7.
4. The DMA fetches the transmit frame from the application memory and transfers the frame to the MFL until the End-of-Frame data is transferred, closing the intermediate descriptors if this frame is split across multiple descriptors.
5. The DMA waits for the previous frame's frame transmission status and time-stamp. Once the status is available, the DMA writes the time-stamp to TDES2 and TDES3, if such time-stamp was captured (as indicated by a status bit). The DMA then writes the status, with a cleared OWN bit, to the corresponding TDES0, thus closing the descriptor. If time-stamping was not enabled for the previous frame, the DMA does not alter the contents of TDES2 and TDES3.
6. If enabled, the transmit interrupt is set; the DMA fetches the next descriptor, and then proceeds to Step 3 (when status is normal). If the previous transmission status shows an underflow error, the DMA goes into suspend mode (Step 7).
7. In suspend mode, if a pending status and time-stamp are received from the MFL, the DMA writes the time-stamp (if enabled for the current frame) to TDES2 and TDES3, then writes the status to the corresponding TDES0. It then sets relevant interrupts and returns to suspend mode.
8. The DMA can exit suspend mode and enter the run state (go to Step 1 or Step 2 depending on pending status) only after receiving a transmit poll demand (EMAC_DMA_TXPOLL).

NOTE: If the EMAC_DMA_OPMODE.OSF bit is set, the DMA fetches the next descriptor in advance before closing the current descriptor. Therefore the descriptor chain should have more than two different descriptors for proper operation.

NOTE: If the EMAC_DMA_OPMODE.OSF bit is set, the DMA starts fetching the second frame immediately after completing the transfer of the first frame to the FIFO. It does not wait for the status to be updated. In the meantime the MFL receives the second frame into the FIFO while transmitting the first frame. The difference in cycles are not seen for the first descriptor, because the time taken for

the complete descriptor processing remains the same whether EMAC_DMA_OPMODE.OSF is set or not. The difference is seen only for the following descriptor as the processing of that descriptor is started earlier.

Transmit Frame Processing

The transmit DMA engine expects that the data buffers contain complete Ethernet frames, excluding preamble, pad bytes, and FCS fields. The destination address, source address, and type/length fields contain valid data. If the transmit descriptor indicates that the EMAC CORE must disable CRC or PAD insertion, the buffer must have complete Ethernet frames (excluding preamble), including the CRC bytes.

Frames can be data-chained and can span several buffers. Frames must be delimited by the first descriptor (TDES1 [29]) and the last descriptor (TDES1 [30]), respectively.

As transmission starts, the first descriptor must have (TDES1 [29]) set. When this occurs, frame data transfers from the application buffer to the transmit FIFO. Concurrently, if the current frame has the Last Descriptor (TDES1 [30]) cleared, the transmit process attempts to acquire the next descriptor. The transmit process expects this descriptor to have TDES1 [29] clear. If TDES1 [30] is clear, it indicates an intermediary buffer. If TDES1 [30] is set, it indicates the last buffer of the frame.

After the last buffer of the frame has been transmitted, the DMA writes back the final status information to the transmit descriptor 0 (TDES0) word of the descriptor that has the last segment set in transmit descriptor 1 (TDES1 [30]). At this time, if interrupt on completion (TDES1 [31]) was set, transmit interrupt (DMA_STAT [0]) is set, the next descriptor is fetched, and the process repeats.

Actual frame transmission begins after the MFL transmit FIFO has reached either a programmable transmit threshold (EMAC_DMA_OPMODE.TTC), or a full frame is contained in the FIFO. There is also an option for store and forward mode (EMAC_DMA_OPMODE.TSF). Descriptors are released (OWN bit TDES0 [31] clears) when the DMA finishes transferring the frame.

Transmit Polling Suspended

Transmit polling may be suspended by either of the following conditions.

1. The DMA detects a descriptor owned by the application (TDES0 [31] = 0).
2. A frame transmission is aborted when a transmit error due to underflow is detected. The appropriate transmit descriptor 0 (TDES0) bit is set.

If the second condition occurs, both abnormal interrupt summary ([15]) and transmit underflow bits ([5]) are set and the information is written to transmit descriptor 0, causing the suspension. If the DMA goes into SUSPEND state due to the first condition then both EMAC_DMA_STAT.NIS and EMAC_DMA_STAT.TU are set.

In both cases, the position in the transmit list is retained. The retained position is that of the descriptor following the last descriptor closed by the DMA.

The driver must explicitly issue a transmit poll demand command after rectifying the suspension cause. If the first condition occurs, the driver must give descriptor ownership to the DMA and then issue a poll demand command, in order to resume the transfer.

Receive Descriptor

The structure of the receive descriptor is shown below. It can have 32 bytes of descriptor data (8 DWORDs) when advanced time-stamping or checksum is enabled.

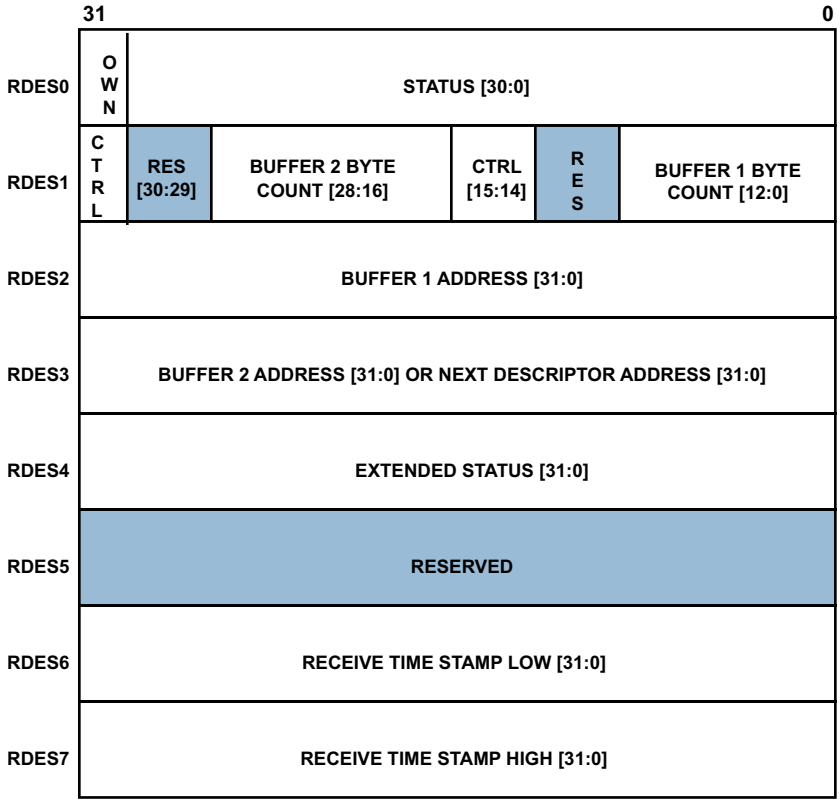


Figure 23-8: Receive Descriptor words

Table 23-18: Receive Descriptor Fields (RDES0)

Bit	Name	Description
31	OWN	Own. When set, this bit indicates that the descriptor is owned by the DMA of the EMAC Subsystem. When this bit is reset, this bit indicates that the descriptor is owned by the application. The DMA clears this bit either when it completes the frame reception or when the buffers that are associated with this descriptor are full.
30	AFM	Destination Address Filter Fail. When set, this bit indicates a frame that failed in the DA Filter in the EMAC CORE.

Table 23-18: Receive Descriptor Fields (RDES0) (Continued)

Bit	Name	Description
29–16	FL	Frame Length. These bits indicate the byte length of the received frame that was transferred to application memory (including CRC). This field is valid when last descriptor (RDES0[8]) is set and either the descriptor error (RDES0[14]) or overflow error bits are reset. This field is valid when Last Descriptor (RDES0[8]) is set. When the last descriptor and error summary bits are not set, this field indicates the accumulated number of bytes that have been transferred for the current frame.
15	ES	Error Summary. Indicates the logical OR of the following bits. RDES0[1] = CRC Error RDES0[4] = Watchdog Timeout RDES0[6] = Late Collision RDES0[7] = time-stamp Available RDES4[4:3] = IP Header/Payload Error RDES0[11] = Overflow Error RDES0[14] = Descriptor Error. This field is valid only when the last descriptor (RDES0[8]) is set.
14	DE	Descriptor Error. When set, this bit indicates a frame truncation caused by a frame that does not fit within the current descriptor buffers, and that the DMA does not own the next descriptor. The frame is truncated. This field is valid only when the last descriptor (RDES0[8]) is set.
13	Reserved	
12	LE	Length Error. When set, this bit indicates that the actual length of the frame received and that the length/type field does not match. This bit is valid only when the frame type (RDES0[5]) bit is reset.
11	OE	Overflow Error. When set, this bit indicates that the received frame was damaged due to buffer overflow in MFL.
10	VLAN	VLAN Tag. When set, this bit indicates that the frame pointed to by this descriptor is a VLAN frame tagged by the EMAC CORE.
9	FS	First Descriptor. When set, this bit indicates that this descriptor contains the first buffer of the frame. If the size of the first buffer is 0, the second buffer contains the beginning of the frame. If the size of the second buffer is also 0, the next descriptor contains the beginning of the frame.
8	LS	Last Descriptor. When set, this bit indicates that the buffers pointed to by this descriptor are the last buffers of the frame
7	time-stamp Available	When set, this bit indicates that a snapshot of the timestamp is written in descriptor words 6 (RDES6) and 7 (RDES7). This is valid only when the last descriptor bit (RDES0[8]) is set
6	LC	Late Collision. When set, this bit indicates that a late collision has occurred while receiving the frame in half-duplex mode.

Table 23-18: Receive Descriptor Fields (RDES0) (Continued)

Bit	Name	Description
5	FT	Frame Type. When set, this bit indicates that the receive frame is an Ethernet-type frame (the LT field is greater than or equal to 16'h0600). When this bit is reset, it indicates that the received frame is an IEEE802.3 frame. This bit is not valid for Runt frames less than 14 bytes.
4	RWT	Receive Watchdog Timeout. When set, this bit indicates that the receive watchdog timer has expired while receiving the current frame and the current frame is truncated after the watchdog timeout.
3	Reserved	
2	DE	Dribble Bit Error. When set, this bit indicates that the received frame has a non-integer multiple of bytes (odd nibbles).
1	CE	CRC Error. When set, this bit indicates that a Cyclic Redundancy Check (CRC) Error occurred on the received frame. This field is valid only when the Last Descriptor (RDES0[8]) is set.
0	Extended Status Available	When set, this bit indicates that the extended status is available in descriptor word 4 (RDES4). This is valid only when the last descriptor bit (RDES0[8]) is set.

Table 23-19: Receive Descriptor Fields 1 (RDES1)

Bit	Name	Description
31	DIC	Disable Interrupt on Completion. When set, this bit prevents setting the status register's `EMAC_DMA_STAT.RI` bit for the received frame ending in the buffer indicated by this descriptor. This, in turn, disables the assertion of the interrupt to the application due to RI for that frame.
30–29	Reserved	
28–16	RBS2	Receive Buffer 2 Size. These bits indicate the second data buffer size, in bytes. The buffer size must be a multiple of 4, (32-bit bus), even if the value of RDES3 (buffer2 address pointer) is not aligned to bus width. If the buffer size is not an appropriate multiple of 4, 8, or 16, the resulting behavior is undefined. This field is not valid if RDES1[14] is set.
15	RER	Receive End of Ring. When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a descriptor ring.
14	RCH	Second Address Chained. When set, this bit indicates that the second address in the descriptor is the next descriptor address rather than the second buffer address. When this bit is set, RBS2 (RDES1[28:16]) is a *don't care* value. RDES1[15] takes precedence over RDES1[14].
13	Reserved	

Table 23-19: Receive Descriptor Fields 1 (RDES1) (Continued)

Bit	Name	Description
12–0	RBS1	Receive Buffer 1 Size. Indicates the first data buffer size in bytes. The buffer size must be a multiple of 4 (32-bit bus), even if the value of RDES2 (buffer1 address pointer) is not aligned. When the buffer size is not a multiple of 4 the resulting behavior is undefined. If this field is 0, the DMA ignores this buffer and uses buffer 2 or next descriptor depending on the value of RCH (Bit 14).

Table 23-20: Receive Descriptor Fields 2 (RDES2)

Bit	Name	Description
31–0	Buffer 1 Address Pointer	These bits indicate the physical address of buffer 1. There are no limitations on the buffer address alignment except for the following condition: The DMA uses the configured value for its address generation when the RDES2 value is used to store the start of frame. Note that the DMA performs a write operation with the RDES2[1:0] bits as 0 during the transfer of the start of frame but the frame data is shifted as per the actual buffer address pointer. The DMA ignores RDES2[1:0] (corresponding to bus width of 32) if the address pointer is to a buffer where the middle or last part of the frame is stored.

Table 23-21: Receive Descriptor Fields 3 (RDES3)

Bit	Name	Description
31–0	Buffer 2 Address Pointer (Next Descriptor Address)	These bits indicate the physical address of buffer 2 when a descriptor ring structure is used. If the second address chained (RDES1[24]) bit is set, this address contains the pointer to the physical memory where the next descriptor is present. If RDES1[24] is set, the buffer (next descriptor) address pointer must be bus width-aligned (RDES3[1:0] = 0, corresponding to a bus width of 32. LSBs are ignored internally.) However, when RDES1[24] is reset, there are no limitations on the RDES3 value, except for the following condition: The DMA uses the configured value for its buffer address generation when the RDES3 value is used to store the start of frame. The DMA ignores RDES3[1:0] (corresponding to a bus width of 32) if the address pointer is to a buffer where the middle or last part of the frame is stored.

Table 23-22: Receive Descriptor Fields 4 (RDES4)

Bit	Name	Description
31–14	Reserved	
13	PTP Version	When set, indicates that the received PTP message is having the IEEE 1588 version 2 format. When reset, it has the version 1 format. This is valid only if the message type is non-zero.

Table 23-22: Receive Descriptor Fields 4 (RDES4) (Continued)

Bit	Name	Description
12	PTP Frame Type	When set, this bit indicates that the PTP message is sent directly over Ethernet. When this bit is not set and the message type is non-zero, it indicates that the PTP message is sent over UDP-IPv4 or UDP-IPv6. The information on IPv4 or IPv6 can be obtained from bits 6 and 7.
11–8	Message Type	These bits are encoded to give the type of the message received. 0000 = No PTP message received 0001 = SYNC (all clock types) 0010 = Follow_Up (all clock types) 0011 = Delay_Req (all clock types) 0100 = Delay_Resp (all clock types) 0101 = Pdelay_Req (in peer-to-peer transparent clock) or Announce (in ordinary or boundary clock) 0110 = Pdelay_Resp (in peer-to-peer transparent clock) or Management (in ordinary or boundary clock) 0111 = Pdelay_Resp_Follow_Up (in peer-to-peer transparent clock) or Signaling (for ordinary or boundary clock) 1xxx - Reserved
7	IPv6 Packet Received	When set, this bit indicates that the received packet is an IPv6 packet.
6	IPv4 Packet Received	When set, this bit indicates that the received packet is an IPv4 packet.
5	IP Checksum Bypassed	When set, this bit indicates that the checksum off-load engine is bypassed.
4	IP Payload Error	When set, this bit indicates that the 16-bit IP payload checksum (that is, the TCP, UDP, or ICMP checksum) that the core calculated does not match the corresponding checksum field in the received segment. It is also set when the TCP, UDP, or ICMP segment length does not match the payload length value in the IP Header field.
3	IP Header Error	When set, this bit indicates either that the 16-bit IPv4 header checksum calculated by the core does not match the received checksum bytes, or that the IP datagram version is not consistent with the Ethernet Type value.
2–0	IP Payload Type	These bits indicate the type of payload encapsulated in the IP datagram processed by the receive checksum off-load engine (COE). The COE also sets these bits to 2'b00 if it does not process the IP datagram's payload due to an IP header error or fragmented IP. 000 = Unknown or did not process IP payload 001 = UDP 010 = TCP 011 = ICMP 1xx = Reserved

Table 23-23: Receive Descriptor Fields 6 (RDES6)

Bit	Name	Description
31–0	RTSL	Receive Frame Time-Stamp Low. This field is updated by DMA with the least significant 32 bits of the time-stamp captured for the corresponding receive frame. This field is updated by DMA only for the last descriptor of the receive frame which is indicated by last descriptor status bit (RDES0[8]).

Table 23-24: Receive Descriptor Fields 7 (RDES7)

Bit	Name	Description
31–0	RTSH	Receive Frame Time-Stamp High. This field is updated by DMA with the most significant 32 bits of the time-stamp captured for the corresponding receive frame. This field is updated by DMA only for the last descriptor of the receive frame which is indicated by last descriptor status bit (RDES0[8]).

EMAC DMA Receive Process

The following sections describe how the direct memory access receive process works on the EMAC controller.

- *Receive Frame Processing*
- *Receive Descriptor Acquisition*
- *Receive Process Suspended*

The Receive DMA engine's reception proceeds as follows:

1. The application sets up receive descriptors (RDES0–RDES3) and sets the `OWN` bit (RDES0 [31]).
2. Once the `EMAC_DMA_OPMODE.SR` bit is set, the DMA enters the run state. While in the run state, the DMA attempts to acquire free descriptors by polling the receive descriptor list. If the fetched descriptor is not free (is owned by the application), the DMA enters the suspend state and jumps to Step 9.
3. The DMA decodes the receive data buffer address from the acquired descriptors.
4. Incoming frames are processed and placed in the acquired descriptor's data buffers.
5. When the buffer is full or the frame transfer is complete, the receive engine fetches the next descriptor.
6. If the current frame transfer is complete, the DMA proceeds to Step 7. If IEEE 1588 time-stamping is enabled, the DMA writes the time-stamp (if available) to the current descriptor. If the DMA does not own the next fetched descriptor and the frame transfer is not complete, the DMA sets the descriptor error bit in the RDES0 (unless flushing is disabled). The DMA closes the current descriptor (clears the `OWN` bit) and marks it as intermediate by clearing the last segment (`LS`) bit in the RDES0 value (marks it as Last descriptor if flushing is not disabled), then proceeds to Step 8. If the DMA owns the next

descriptor but the current frame transfer is not complete, the DMA closes the current descriptor as intermediate and reverts to Step 4.

7. If IEEE 1588 time-stamping is enabled, the DMA writes the time-stamp (if available) to the current descriptor's RDES2 and RDES3. It then takes the receive frame's status from the MFL and writes the status word to the current descriptor's RDES0, with the OWN bit cleared and the last segment bit set.
8. The receive engine checks the latest descriptor's OWN bit. If the host owns the descriptor (OWN bit is 0) the EMAC_DMA_STAT.RU bit is set and the DMA receive engine enters the suspended state (Step 9). If the DMA owns the descriptor, the engine returns to Step 4 and awaits the next frame.
9. Before the receive engine enters the suspend state, partial frames are flushed from the receive FIFO (programs control flushing using the EMAC_DMA_OPMODE.DFF bit).
10. The receive DMA exits the suspend state when a receive poll demand is given or the start of next frame is available from the MFL's receive FIFO. The engine proceeds to Step 2 and re fetches the next descriptor.

Receive Frame Processing

The EMAC transfers the received frames to the application memory only when the frame passes the address filter sub-block and frame size is greater than or equal to configurable threshold bytes set for the receive FIFO of MFL, or when the complete frame is written to the FIFO in Store-and-Forward mode.

If the frame fails the address filtering, it is dropped in the EMAC block itself (unless the EMAC_MACFRMFILT.RA bit is set). Frames that are shorter than 64 bytes, because of collision or premature termination, can be purged from the receive FIFO.

After 64 bytes (configurable threshold) have been received, the MFL block requests the DMA block to begin transferring the frame data to the receive buffer pointed to by the current descriptor. The DMA sets first descriptor (RDES0 [9]) after the SCB becomes ready to receive the data (if DMA is not fetching transmit data from the application). The descriptors are released when the OWN (RDES [31]) bit is reset to 0, either as the data buffer fills up or as the last segment of the frame is transferred to the receive buffer. If the frame is contained in a single descriptor, both the last descriptor (RDES [8]) and the first descriptor (RDES [9]) are set.

The DMA fetches the next descriptor, sets the last descriptor (RDES [8]) bit, and releases the RDES0 status bits in the previous frame descriptor. Then the DMA sets the EMAC_DMA_STAT.RI bit. The same process repeats unless the DMA encounters a descriptor flagged as being owned by the application. If this occurs, the receive process sets the EMAC_DMA_STAT.RU bit and then enters the suspend state. The position in the receive list is retained.

Receive Descriptor Acquisition

The Receive engine always attempts to acquire an extra descriptor in anticipation of an incoming frame. Descriptor acquisition is attempted if any of the following conditions is satisfied:

- The`EMAC_DMA_OPMODE.SR` bit has been set immediately after being placed in the run state.
- The data buffer of current descriptor is full before the frame ends for the current transfer.
- The controller has completed frame reception, but the current receive descriptor is not yet closed.
- The receive process has been suspended because of an application-owned buffer (RDES0 [31] = 0) and a new frame is received.
- A receive poll demand has been issued.

Receive Process Suspended

If a new receive frame arrives while the receive process is in suspend state, the DMA re-fetches the current descriptor in the application memory. If the descriptor is now owned by the DMA, the receive process re-enters the run state and starts frame reception. If the descriptor is still owned by the application, by default, the DMA discards the current frame at the top of the receive FIFO and increments the missed frame counter. If more than one frame is stored in the receive FIFO, the process repeats.

The discarding or flushing of the frame at the top of the receive FIFO can be avoided by setting the `EMAC_DMA_OPMODE.DFF` bit. In such conditions, the receive process sets the receive buffer unavailable status and returns to the suspend state.

OWN Bit (Ownership) Semaphore

Usage or ownership of the transmit/receive descriptor between application and EMAC is mutually exclusive. While the EMAC is accessing the descriptor, the application cannot modify it. Conversely, while the host is updating the descriptor, the EMAC cannot use the descriptor's contents. This functionality is implemented through the `OWN` bit in the transmit/receive descriptor, acting as a semaphore to prevent multiple, simultaneous access to the descriptors.

The following example is based on a use case of 4 WORDs enabled for descriptors (which means the `EMAC_DMA_BUSMODE.ATDS` bit is not set) and chain structure configuration is assumed. However, the explanation of the `OWN` bit semaphore remains consistent irrespective of any particular mode of operation.

1. Transmit `OWN` Bit:
    - TDES0 – TDES3 words implement the transmit descriptors. TDES0 [31] is defined as the `OWN` bit. When TDES0 [31] is set to 0, this bit indicates that the descriptor is available for the application to update. The application sets up the descriptors, including the buffer addresses, by updating TDES0 through TDES3.
    - To release ownership of the descriptor to the EMAC, the application sets the transmit `OWN` bit, TDES0 [31], to 1. TDES0 [31] = 1 indicates that the descriptor is ready for use by the EMAC. The DMA reads the descriptors, then fetches the data to be transmitted from the buffer loca-

tions pointed to by the transmit descriptors (TDES2 and TDES3). When either the last data buffer is empty or the end-of-frame is reached, DMA clears the TDES0 [31] bit to 0. Now the transmit descriptor is released to the application for updates.

2. Receive OWN Bit:

   - RDES0 – RDES3 words implement the receive descriptors. RDES0 [31] is defined as the OWN bit. When RDES0 [31] is set to 0, this bit indicates that the descriptor is available for the application to update. The application sets up the descriptors, including the buffer locations for writing the received data, by updating RDES0 through RDES3. To give ownership of the descriptor to the EMAC, the host sets the receive OWN bit, RDES0 [31], to 1.
   - RDES0 [31] = 1 indicates that the descriptor is ready for use by the EMAC. The EMAC's DMA reads the descriptors, then writes the received data to the buffers with locations pointed to by the receive descriptors (RDES2 and RDES3). When either the last data buffer is full or the end-of-frame is reached, DMA clears the RDES0 [31] bit to 0. Now the receive descriptor is released to the application for updates

Application Data Buffer Alignment

The transmit and receive data buffers do not have any restrictions on start address alignment; the start address for the buffers can be aligned to any of the four bytes. However, the DMA always initiates transfers with address aligned to the bus width with dummy data for the byte lanes not required. This typically happens during the transfer of the beginning or end of an Ethernet frame.

Example for Buffer Read—If the transmit buffer address is 0xFF800002 and 15 bytes need to be transferred, then the DMA reads 5 full words (5 x 32-bit data) from address 0xFF800000. However, when transferring data to the EMAC transmit FIFO, the extra bytes (the first two bytes) are dropped or ignored. Similarly, the last 3 bytes of the last transfer are also ignored. The DMA always ensures it transfers a full 32-bit data to the transmit FIFO, unless it is the end-of-frame.

Example for Buffer Write—If the receive buffer address is 0xFF800002 and 15 bytes of a received frame need to be transferred, then the DMA writes 5 full words (5 x 32-bit data) to address 0xFF800000. However, the first 2 bytes of first transfer and the last 3 bytes of the third transfer have dummy data.

Buffer Size Calculations

The DMA engines do not update the size fields in the transmit and receive descriptors alone. The DMA updates only the status fields (RDES0 and TDES0) of the descriptors. The driver has to perform the size calculations. The transmit DMA transfers the exact number of bytes (indicated by buffer size field of TDES1) towards the EMAC CORE. If a descriptor is marked as first (FS bit of TDES1 is set), then the DMA marks the first transfer from the buffer as the start of frame. If a descriptor is marked as last (LS bit of TDES1), then the DMA marks the last transfer from that data buffer as the end-of frame to the EMAC.

The receive DMA transfers data to a buffer until the buffer is full or the end-of frame is received from the MFL. If a descriptor is not marked as last (LS bit of RDES0), then the descriptor's corresponding buffer(s) are full and the amount of valid data in a buffer is accurately indicated by its buffer size field minus the data buffer pointer offset when the FS bit of that descriptor is set. The offset is zero when the data buffer pointer

is aligned to the data bus width. If a descriptor is marked as last, then the buffer may not be full (as indicated by the buffer size in RDES1). To compute the amount of valid data in this final buffer, the driver must read the frame length (FL bits of RDES0[29:16]) and subtract the sum of the buffer sizes of the preceding buffers in this frame. The receive DMA always transfers the start of next frame with a new descriptor.

EMAC FIFO Layer (EMAC MFL)

The MAC FIFO layer provides FIFO memory to buffer and regulates the frames between the application system memory and the EMAC CORE. It also allows the data to be transferred between the application clock domain and the EMAC clock domains. The MFL layer has transfer controllers for each direction, called the transmit controller (TxFIFO) and the receive controller (RxFIFO). The data path for both directions is 32-bit wide and each controller has a dedicated FIFO.

FIFO Size

The transmit FIFO size is fixed and is 256 bytes. The receive FIFO sized is fixed and is 128 bytes.

FIFO Layer Transmit Path

The DMA Engine controls all transactions for the transmit path with the application. Ethernet frames read from the system memory are pushed into the FIFO by the DMA. The frame is then popped out and transferred to the EMAC CORE when triggered. When the end-of-frame is transferred, the status of the transmission is taken from the EMAC CORE and transferred back to the DMA. FIFO-fill level is indicated to the DMA so that it can initiate a data fetch in required bursts from the system memory, through the SCB interface.

When the EMAC_DMA_OPMODE.OSF bit is enabled, the MFL receives the second frame into the FIFO while transmitting the first frame. As soon as the first frame has been transferred and the status is sent to DMA. If the DMA has already completed sending the second packet to the MFL, it must wait for the status of the first packet before proceeding to the next frame.

The following are the modes of operation for FIFO transactions.

1. Threshold mode – In this mode as soon as the number of bytes in the FIFO crosses the configured threshold level (or when the end-of-frame is written before the threshold is crossed), the data is ready to be popped out and forwarded to the EMAC core. The threshold level is configured using the TTC bits of the DMA bus mode register.

2. Store-and-Forward mode – In this mode, the MFL pops the frame towards the EMAC core only after a complete frame is stored in the FIFO. If the Tx FIFO size is smaller than the Ethernet frame to be transmitted (such as a Jumbo frame), then the frame is forwarded when the Tx FIFO becomes almost full or when the requested FIFO does not have space to accommodate the requested burst-length. Therefore, the FIFO read controller never stalls in Store and Forward mode even if the Ethernet frame length is bigger than the Tx FIFO depth.

Transmit FIFO and Half-Duplex Retransmissions

While a frame is being transferred from the FIFO a collision event can occur on the EMAC line interface in half-duplex mode. The EMAC then indicates a retry attempt to the MFL by giving the status even before the end-of-frame is transferred from MFL. Then the MFL enables the retransmission by popping out the frame again from the FIFO.

After more than 96 bytes are popped out of FIFO, the FIFO controller frees up that space and makes it available to the DMA to push in more data. This means that the retransmission is not possible after this threshold is crossed or when the EMAC CORE indicates a late-collision event.

Transmit FIFO Flush Operation

The EMAC provides a control to the software to flush the transmit FIFO in the MFL layer through the use of the EMAC_DMA_OPMODE.FTF bit. The flush operation is immediate and the MFL clears the transmit FIFO and the corresponding pointers to the initial state even if it is in the middle of transferring a frame to the EMAC CORE. The data which is already accepted by the MAC transmitter is not flushed. It is scheduled for transmission and results in underflow as the transmit FIFO does not complete the transfer of rest of the frame. As in all underflow conditions, a runt frame is transmitted and observed on the line. The status of such a frame is marked with both underflow and frame flush events (TDES0 bits 13 and 1).

The MFL layer also stops accepting any data from the application (DMA) during the flush operation. It generates and transfers transmit status words to the application for the number of frames that is flushed inside the MFL (including partial frames). Frames that are completely flushed in the MFL have the frame flush status bit (TDES0 bit 13) set. The MFL completes the flush operation when the application (DMA) accepts all of the status words for the frames that were flushed, and then clears the transmit FIFO flush control register bit. At this point, the MFL starts accepting new frames from the application (DMA).

FIFO Layer Receive Path

The receive controller operates in the following sequence.

1. When the EMAC core receives a frame, it pushes in data with the frame start and end indicators. The MFL accepts the data and pushes it into the FIFO.
2. The receive controller takes the data out of the FIFO and sends it to the DMA.
    - In the default threshold mode, when 64 bytes (configured using EMAC_DMA_OPMODE.RTC) or a full packet of data are received into the FIFO, the receive controller pops out the data and indicates its availability to the DMA. Some error frames may not be dropped, because the error status is received at the end-of-frame, by which time the start of that frame has already been read out of the FIFO.
    - In Rx FIFO Store-and-Forward mode (configured using EMAC_DMA_OPMODE.RSF), a frame is read out only after being written completely into the receive FIFO. In this mode, all error frames are dropped (if the EMAC core is configured to do so) such that only valid frames are read out and forwarded to the application.

3. After the end-of-frame is transferred, the status word from EMAC core is also the pushed FIFO. When the status of a partial frame due to overflow is given out, the frame length field in the status word is not valid.

Receive FIFO Multi-Frame Handling

Since the status is available immediately following the data, the MFL is capable of storing any number of frames into the FIFO, as long as it is not full.

Receive FIFO Error Handling

If the MFL Rx FIFO is full before it receives the end-of-frame data from the EMAC, an overflow is declared, the whole frame (including the status word) is dropped, and the overflow counter in the DMA (Over Flow Counter register) is incremented. This is true even if the `EMAC_DMA_OPMODE.FEF` bit is set. If the start address of such a frame has already been transferred, the rest of the frame is dropped and a dummy end-of-frame is written to the FIFO along with the status word. The status indicates a partial frame due to overflow. In such frames, the frame length field is invalid.

The MFL receive control logic can filter error and undersized frames using the `EMAC_DMA_OPMODE.FEF` and `EMAC_DMA_OPMODE.FUF` bits. If the start address of such a frame has already been transferred to the Rx FIFO read controller, that frame is not filtered. The start address of the frame is transferred to the read controller after the frame crosses the receive threshold (set by the `EMAC_DMA_OPMODE.RTC` bits).

If the MFL receive FIFO is configured to operate in Store-and-Forward mode, all error frames can be filtered and dropped.

EMAC CORE

The EMAC CORE is the lowest block in the EMAC peripheral and it performs all operations with the external world (PHY chip). It has independent transmit and receive modules that interact with the EMAC FIFO layer at one end and the PHY chip via the RMII interface at the other end. Both the modules have several sub blocks which are discussed in subsequent sections.

Transmission is initiated when the MFL (FIFO Layer) pushes in data with start-of-frame and the CORE subsequently transmitting to the RMII. After the end-of-frame is transferred out, it gives out the status of the transmission back to the MFL to be forwarded to the application via DMA.

A receive operation is initiated when the EMAC detects a SFD on the RMII. The CORE strips the preamble and SFD before proceeding to process the frame. The header fields are checked for the filtering and the FCS field used to verify the CRC for the frame. The frame is dropped in the core if it fails the address filter.

NOTE: The term *CORE* (written in capitals) is used refer to the internal block of Ethernet peripheral, and should not be confused with the *processor core*.

Table 23-25: EMAC CORE Related Registers

Register Name	Description
MAC Configuration[1]	Establishes receive and transmit operating modes including: • Watchdog/Jabber/Jumbo frame sizes • Inter Frame Gap • Speed Control – 10/100 Mbps • Full/Half Duplex • Loopback Mode • Checksum Offload • Enabling TX/RX Engines
MAC Frame Filter	Contains the filter controls for receiving frames. Some of the controls from this register go to the address check block of the MAC, which performs the first level of address filtering. The second level of filtering is performed on the incoming frame, based on other controls such as pass bad frames and pass control frames.
Hash Table High/Low[1]	A 64-bit hash table is used for group address filtering. For hash filtering, the contents of the destination address in the incoming frame is passed through the CRC logic, and the upper 6 bits of the CRC register are used to index the contents of the hash table.
SMI Address[1]	Controls the management cycles to the external PHY through the Station Management interface. The register also includes a field to program the frequency of MDC.
SMI Data[1]	Stores write data to be written to the PHY register located at the address specified in SMI Address register. This register also stores read data from the PHY register located at the address specified by SMI address register.
Flow Control[1]	Controls the generation and reception of the control (pause command) frames by the EMAC's flow control module. The fields of the control frame are selected as specified in the 802.3x specification, and the pause time value from this register is used in the pause time field of the control frame. The host must make sure that the activate bit is cleared before writing to the register.
VLAN Tag[1]	Contains the IEEE 802.1Q VLAN tag to identify the VLAN frames. The MAC compares the 13th and 14th bytes of the receiving frame (length/type) with 16.h8100, and the following 2 bytes are compared with the VLAN tag. If a match occurs, it sets the received VLAN bit in the receive frame status. The legal length of the frame is increased from 1518 bytes to 1522 bytes.

Table 23-25: EMAC CORE Related Registers (Continued)

Register Name	Description
Debug	Provides the status of all main modules of the transmit and receive data-paths and the FIFOs. An all-zero status indicates that the MAC core is in idle state (and FIFOs are empty) and no activity is going on in the data-paths.
Interrupt Status	The contents of this register identify the events in the EMAC-CORE that can generate MMC and PTP related interrupts.
Interrupt Mask	Enables the program to mask the interrupt signal because of the corresponding PTP event in the interrupt status register.
MAC Address0 High/Low[1]	Holds the upper/lower 16 bits of the MAC address of the station. Note that the first DA byte that is received on the RMII interface corresponds to the LS byte (bits [7:0]) of the MAC address low register. For example, if 0x112233445566 is received (0x11 is the first byte) on the RMII as the destination address, then the macaddress0 register [47:0] is compared with 0x665544332211.
Operation Mode[1]	

1.There should not be any further writes to these registers until the first write is updated. Otherwise, the second write operation is not updated properly. For correct operation, the delay between two writes to the same register location should be at least 8 cycles of 50MHz RMII REFCLK.

NOTE: Please refer to the "Register Description" section for the detailed bit-level explanation of the registers.

EMAC CORE Transmission Engine

The following modules constitute the transmission function (transmission engine components) of the EMAC:

- *Transmit Bus Interface Module (TBU)*
- *Transmit Frame Controller Module (TFC)*
- *Transmit Checksum Offload Engine (TCOE)*
- *Transmit Protocol Engine Module (TPE)*
- *Transmit Scheduler Module (STX)*
- *Transmit CRC Generator Module (CTX)*
- *Transmit Flow Control Module (FTX)*

Transmit Bus Interface Module (TBU)

This module interfaces the transmit path of the EMAC CORE with the MAC Layer FIFO interface. This module outputs the transmit status to the application at the end of normal transmission or collision.

Transmit Frame Controller Module (TFC)

The transmit frame controller regulates frames as well as converts the 32-bit input data into an 8-bit stream.

When the number of bytes received from the application falls below 60 (DA+SA+LT+DATA), the state machine automatically appends zeros to the transmitting frame to make the data length exactly 46 bytes to meet the minimum data field requirement of IEEE 802.3. The EMAC can also be programmed not to append any padding.

The frame controller receives the computed CRC and appends it as the FCS field to the data being transmitted out. When the EMAC is programmed to not append the CRC value to the end of Ethernet frames, the TFC module ignores the computed CRC. An exception to this rule is that when the EMAC is programmed to append pads for frames (DA+SA+LT+DATA) less than 60 bytes, then the CRC is always appended at the end of padded frame.

Transmit Checksum Offload Engine (TCOE)

Communication protocols such as TCP and UDP implement checksum fields, which help determine the integrity of data transmitted over a network. Because the most widespread use of Ethernet is to encapsulate TCP and UDP over IP datagrams, the EMAC has a checksum offload engine (COE) to support checksum calculation and insertion in the transmit path, and error detection in the receive path.

NOTE: The checksum for TCP, UDP, or ICMP is calculated over a complete frame, and then inserted into its corresponding header field. Because of this requirement, this function is enabled only when the transmit FIFO is configured for store-and-forward mode (that is, when the `EMAC_DMA_OPMODE.TSF` bit is set. If the MAC is configured for threshold (cut-through) mode, the transmit COE is bypassed.

NOTE: Programs must make sure that the transmit FIFO is deep enough to store a complete frame before that frame is transferred to the EMAC CORE transmitter. The program must enable the checksum insertion only in the frames that are less than the following number of bytes in size (even in the store-and-forward mode): FIFO depth (256 bytes) – PBL – 3 FIFO locations, where PBL is the programmed burst-length in the DMA bus mode register.

IP Header Checksum

In IPv4 datagrams, the integrity of the header fields is indicated by the 16-bit header checksum field (the eleventh and twelfth bytes of the IPv4 datagram). The COE detects an IPv4 datagram when the Ethernet frame's type field has the value 0x0800 and the IP datagram's version field has the value 0x4. The input frame's checksum field is ignored during calculation and replaced with the calculated value.

The result of this IP header checksum calculation is indicated by the IP header error status bit in transmit descriptor word TDES0. This status bit is set whenever the values of the Ethernet type field and the IP header's version field are not consistent, or when the Ethernet frame does not have enough data, as indicated by the IP header length field. In other words, this bit is set when an IP header error is asserted under the following circumstances.

For IPv4 datagrams:

- The received Ethernet type is 0x0800, but the IP header's version field is not equal to 0x4.
- The IPv4 header length field indicates a value less than 0x5 (20 bytes).
- The total frame length is less than the value given in the IPv4 header length field.

For IPv6 datagrams:

- The Ethernet type is 0x86dd but the IP header version field is not equal to 0x6.
- The frame ends before the IPv6 header (40 bytes) or extension header (as given in the corresponding Header Length field in an extension header) is completely received.

If the COE detects an IP header error, it still inserts an IPv4 header checksum if the Ethernet type field indicates an IPv4 payload.

NOTE: IPv6 headers do not have a checksum field. Therefore, the COE does not modify the IPv6 header fields.

TCP/UDP/ICMP Checksum

The TCP/UDP/ICMP checksum engine processes the IPv4 or IPv6 header (including extension headers) and determines whether the encapsulated payload is TCP, UDP, or ICMP.

NOTE: See IETF specifications RFC 791, RFC 793, RFC 768, RFC 792, RFC 2460, and RFC 4443 for IPv4, TCP, UDP, ICMP, IPv6, and ICMPv6 packet header specifications, respectively.

NOTE: For non-TCP/UDP/ICMP/ICMPv6 payloads, this checksum engine is bypassed and nothing further is modified in the frame.

NOTE: For ICMP-over-IPv4 packets, the Checksum field in the ICMP packet must always be 0x0000 in both modes, because pseudo-headers are not defined for such packets. If it does not equal 0x0000, an incorrect checksum may be inserted into the packet.

NOTE: Fragmented IP frames (IPv4 or IPv6), IP frames with security features (such as an encapsulated security payload), and IPv6 frames with routing headers are not processed by this engine. The checksum engine bypasses the checksum insertion for such frames even if the checksum insertion is enabled.

The checksum is calculated for the TCP, UDP, or ICMP payload and inserted into its corresponding field in the header. This engine can work in the following two ways.

- The TCP, UDP, or ICMPv6 pseudo-header is not included in the checksum calculation and is assumed to be present in the checksum field of the input frame. This engine includes the checksum field in the checksum calculation, and then replaces the checksum field with the final calculated checksum.
- The engine ignores the checksum field, includes the TCP, UDP, or ICMPv6 pseudo-header data into the checksum calculation, and overwrites the checksum field with the final calculated value.

The result of this operation is indicated by the payload checksum error status bit in the transmit descriptor word TDES0. The checksum engine sets the payload checksum error status bit when it detects that the frame has been forwarded to the MAC transmitter engine in the store-and-forward mode without the end-of-frame (EOF) being written to the FIFO, or when the packet ends before the number of bytes indicated by the payload length field in the IP header is received. When the packet is longer than the indicated payload length, the COE ignores them as stuff bytes, and no error is reported. When the engine detects the first type of error, it does not modify the TCP, UDP, or ICMP header. For the second error type, it still inserts the calculated checksum into the corresponding header field.

Transmit checksum offloading is enabled by setting the CIC bits [23:22] of TDES0 word in the transmit descriptor.

Transmit Protocol Engine Module (TPE)

The transmit protocol engine consists of a state-machine that controls the protocol level operation of Ethernet frame transmission. The module performs the following functions to meet the IEEE 802.3 specifications.

- Generates preamble and SFD
- Generates jam pattern in half-duplex mode
- Jabber timeout
- Flow control for half-duplex mode (back pressure)
- Generates transmit frame status

When a new frame transmission is requested, the protocol engine sends out the preamble and SFD, followed by the data received. The preamble is defined as 7 bytes of 10101010 pattern and the SFD is defined as 1 byte of 10101011 pattern.

The collision window is defined as 1 slot time (512 bit times for 10/100 Mbps). The jam pattern generation is applicable only to half-duplex mode, not to full-duplex mode. If a collision occurs any time from the beginning of the frame to the end of the CRC field, the state machine sends a 32-bit jam pattern of 0x55555555 on the RMII to inform all other stations that a collision has occurred. If the collision is seen during the preamble transmission phase, it completes the transmission of preamble and SFD and then sends the jam pattern. If the collision occurs after the collision window and before the end of the FCS field, it sends a 32-bit jam pattern and sets the late collision bit in the transmit frame status.

The module maintains a jabber timer (in 10/100-Mbps) to cut off the transmission of Ethernet frames if the TFC module transfers more than 2,048 (default) bytes. The time-out is changed to 10,240 bytes when the jumbo frame is enabled.

The transmit state machine uses the deferral mechanism for the flow control (back pressure) in half-duplex mode. When the application asks to stop receiving frames, the module sends a JAM pattern of 32 bytes whenever it senses a reception of a frame, provided the transmit flow control is enabled. This results in a collision and the remote station backs off.

The application can request a flow control signal by setting the EMAC_FLOWCTL.FCBBPA bit. If the application requests a frame to be transmitted, then it is scheduled and transmitted even when the back pressure is activated. Note that if the back pressure is kept activated for a long time (and more than 16 consecutive collision events occur) then the remote stations abort their transmissions due to excessive collisions.

Transmit Scheduler Module (STX)

The Transmit Scheduler is responsible for scheduling the frame transmission on the RMII. The two major functions of this module are:

- Maintain the inter-frame gap between two transmitted frames.
- Follow the truncated binary exponential back-off algorithm for half-duplex mode.

The scheduler maintains an idle period of the configured inter-frame gap (EMAC_MACCFG.IFG bits) between any two transmitted frames. The scheduler starts its IFG counter as soon as the carrier signal of the RMII goes inactive. In half-duplex mode and when IFG is configured for 96 bit times, the scheduler follows the rule of deference specified in Section 4.2.3.2.1 of the IEEE 802.3 specification. The module resets its IFG counter if a carrier is detected during the first two-thirds (64-bit times for all IFG values) of the IFG interval. If the carrier is detected during the final one third of the IFG interval, the scheduler continues the IFG count and enables the transmitter after the IFG interval.

Transmit CRC Generator Module (CTX)

The transmit CRC generator module generates CRC for the FCS field of the Ethernet frame (DA + SA + LT + DATA + PAD).

This module calculates the 32-bit CRC for the FCS field of the Ethernet frame. The encoding is defined by the following polynomial:

$G(x) = x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1.$

Transmit Flow Control Module (FTX)

The transmit flow control module generates pause frames and transmits them to the frame controller as necessary, in full-duplex mode. The application can request the flow control module to send a pause frame by setting the EMAC_FLOWCTL.FCBBPA bit.

If the application has requested for flow control, the flow control module generate and transmit a single pause frame. The value of the pause time in the generated frame contains the programmed pause time value configured using the EMAC_FLOWCTL.PT bit. To extend the pause or end the pause prior to the time specified in the previously transmitted pause frame, the application must request another pause frame transmission after programming the EMAC_FLOWCTL.PT bit with an appropriate value.

If the flow control signal goes inactive prior to the sampling time, the flow control module transmits a pause frame with zero pause time to indicate to the remote end that the receive buffer is ready to receive new data frames.

EMAC CORE Reception Engine

The following are the functional blocks (reception engine components) in the receive path of the EMAC core.

- *Receive Protocol Engine Module (RPE)*
- *Receive CRC Module (CRX)*
- *Receive Frame Controller Module (RFC)*
- *Receive Flow Control Module (FRX)*
- *Receive Checksum Offload Engine (RCOE)*
- *Receive Bus Interface Unit Module (RBU)*
- *Address Filtering Module (AFM)*

Receive Protocol Engine Module (RPE)

The receive protocol engine is a state-machine that strips the incoming preamble and SFD. Once the receive data valid signal (ETH_CRS) signal of the RMII becomes active, the protocol engine begins hunting for the SFD field from the receive modifier logic. Until then, the state machine drops the receiving preambles. Once the SFD is detected, it begins sending the data of the Ethernet frame to the frame controller, beginning with the first byte following the SFD (destination address).

NOTE: According to the IEEE 802.3 Ethernet specifications, the EMAC receiver need not look or check for the preamble pattern. It has to wait only for the SFD pattern to identify the start of a frame. Then the EMAC receiver accepts a frame even when no preamble is received before the SFD pattern.

The protocol engine also decodes the length/type field of the receiving Ethernet frame. If the length/type field is less than 0x600 and if the MAC is programmed for the auto CRC/PAD stripping option, the state machine sends the data of the frame up to the count specified in the length/type field, then starts dropping bytes (including the FCS field).

If the length/type field is greater than or equal to 0x600, the protocol engine sends all received Ethernet frame data to the frame controller, irrespective of the value on the programmed auto-CRC strip option.

The EMAC is programmed with the watchdog timer enabled (default setting). In this configuration frames above 2,048 (10,240 if jumbo frame is enabled) bytes (DA + SA + LT + DATA + PAD + FCS) are cut off at the protocol engine. This feature can be disabled by setting the EMAC_MACCFG.WD bit. However even if the watchdog timer is disabled, frames greater than 16 KB in size are cut off and a watchdog time-out status is issued.

The EMAC supports loopback of transmitted frames onto its receiver. By default, the EMAC loopback function is disabled, but can be enabled by setting the EMAC_MACCFG.LM bit.

At the end of every received frame, the protocol engine generates received frame status and sends it to the frame controller. Control, missed frame, and filter fail status are added to the receive status in the frame controller.

Receive CRC Module (CRX)

The receive CRC module checks for any CRC errors in the receiving frame.

This module calculates the 32-bit CRC for the received frame that includes the destination address field through the FCS field (DA+SA+LT+DATA+PAD+FCS). The encoding is defined by the following generating polynomial.

G (x) = x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1

Irrespective of the auto pad/CRC strip, the CRC module receives the entire frame to compute the CRC check for received frame.

Receive Frame Controller Module (RFC)

The main functions of the frame controller are:

- Converting the 8-bit stream data to 32-bit data.
- Frame filtering.
- Attaching the calculated IP Checksum.
- Update the receive status.

If the `EMAC_MACFRMFILT.RA` bit is set, the RFC module initiates the data transfer as soon as possible. At the end of the data transfer, the frame controller sends out the received frame status that includes the address filtering pass/fail status.

If the `EMAC_MACFRMFILT.RA` bit is reset, the frame controller performs frame filtering based on the destination/source address (the application still needs to perform another level of filtering if it decides not to receive any bad frames like runt, CRC error frames, for example). After receiving the destination/source address bytes, the frame controller checks the filter-fail signal from the AFM module for an address match. On detecting a filter-fail from AFB, the frame is dropped and not transferred to the application.

Receive Flow Control Module (FRX)

The receive flow controller detects the receiving pause frame and pauses the frame transmission for the delay specified within the received pause frame. The flow controller is enabled only in full-duplex mode. The pause frame detection function can be enabled or disabled with the `EMAC_FLOWCTL.RFE` bit.

Once the receive flow control is enabled, the flow controller begins monitoring the received frame destination address for any match with the multicast address of the control frame (0x0180C2000001). If a match is detected, it indicates to the frame controller, that the destination address of the received frame matches the reserved control frame destination address. The RFC module then decides whether or not to transfer the received control frame to the application, based on the `EMAC_MACFRMFILT.PCF` bit setting.

The receive flow controller also decodes the type, op-code, and pause timer field of the receiving control frame. If the byte count of the frame status indicates 64 bytes, and if there is no CRC error, the flow controller requests the MAC transmitter to pause the transmission of any data frame for the duration of

the decoded pause time value, multiplied by the slot time (64 byte times). Meanwhile, if another pause frame is detected with a zero pause time value, the module resets the pause time and gives another pause request to the transmitter. If the received control frame matches neither the type field (0x8808), opcode (0x00001), nor byte length (64 bytes), or if there is a CRC error, the module does not generate a pause request to the transmitter.

In the case of a pause frame with a multicast destination address, the frame controller filters the frame based on the address match from the flow controller. For a pause frame with a unicast destination address, the filtering in the FRX module depends on whether the destination address matched the contents of the MAC address register 0 (EMAC_ADDR0_HI/EMAC_ADDR0_LO) and the EMAC_FLOWCTL.UP bit is set (detecting a pause frame even with a unicast destination address). The EMAC_MACFRMFILT.PCF bits control the filtering for control frames in addition to the address filter module.

Receive Checksum Offload Engine (RCOE)

When checksum offloading is enabled, both IPv4 and IPv6 frames in the received Ethernet frames are detected and processed for data integrity. Programs can enable this module by setting the EMAC_MACCFG.IPC bit. The EMAC receiver identifies IPv4 or IPv6 frames by checking for value 0x0800 or 0x86DD, respectively, in the received Ethernet frames' type field. This identification applies to VLAN-tagged frames as well. *Extended descriptor mode (8 x32-bit words) must be enabled to get the IPC checksum engine status in RDES4.* Status can be checked by polling the bit 0 of RDES0 word of receive descriptor and then if this bit is set, further parsing bits [7:0] of RDES4 word.

The receive checksum offload engine calculates IPv4 header checksums and checks if they match the received IPv4 header checksums. The IP header error bit is set for any mismatch between the indicated payload type (Ethernet type field) and the IP header version, or when the received frame does not have enough bytes, as indicated by the IPv4 header's length field (or when fewer than 20 bytes are available in an IPv4 or IPv6 header).

This engine also identifies a TCP, UDP or ICMP payload in the received IP datagrams (IPv4 or IPv6) and calculates the checksum of such payloads properly, as defined in the TCP, UDP, or ICMP specifications. This engine includes the TCP/UDP/ICMPv6 pseudo-header bytes for checksum calculation and checks whether the received checksum field matches the calculated value. The result of this operation is given as a payload checksum error bit in the receive status word. This status bit is also set if the length of the TCP, UDP, or ICMP payload does not tally to the expected payload length given in the IP header.

NOTE: The COE engine bypasses the payload of fragmented IP datagrams, IP datagrams with security features, IPv6 routing headers, and payloads other than TCP, UDP or ICMP. This information (whether the checksum engine is bypassed or not) is given in the receive status.

The meaning of checksum related errors can be understood using the table below which shows bit combination in receive descriptors (frame status with full checksum offload engine enabled and advanced timestamps not enabled).

Table 23-26: Checksum Error Status

IEEE802.3 Frame: bit 5 of RDES0	Header Checksum Error: bit 3 of RDES4	Payload Checksum Error: bit 4 of RDES4	Frame Status
0	0	0	The frame is an IEEE 802.3 frame (length field value is less than 0x0600).
1	0	0	IPv4/IPv6 type frame in which no checksum error is detected.
1	0	1	IPv4/IPv6 type frame in which a payload checksum error (as described for PCE) is detected
1	1	0	IPv4/IPv6 type frame in which IP header checksum error (as described for IPC HCE) is detected.
1	1	1	IPv4/IPv6 type frame in which both PCE and IPC HCE is detected.
0	0	1	IPv4/IPv6 type frame in which there is no IP HCE and the payload check is bypassed due to unsupported payload.
0	1	1	Type frame which is neither IPv4 or IPv6 (COE bypasses the checksum check completely)
0	1	0	Reserved

Receive Bus Interface Unit Module (RBU)

The receive bus interface unit (RBU) constructs the 32-bit data received from the frame controller into a 32-bit FIFO based protocol.

Address Filtering Module (AFM)

The address filtering (AFM) module performs the destination checking function on all received frames and reports the address filtering status to the frame controller. The address checking is based on different parameters (frame filter register, `EMAC_MACFRMFILT`) chosen by the application. These parameters are inputs to the AFM module as control signals, and the AFM module reports the status of the address filtering based on the combination of these inputs. The AFM module also reports whether the receiving frame is a multicast frame or a broadcast frame, as well as the address filter status. The AFM module uses the station's physical (MAC) address and the multicast hash table for address checking.

- **Hash or Perfect Address Filter.** The destination address filter can be configured to pass a frame when its destination address matches either the hash filter or the perfect filter by setting the `EMAC_MACFRMFILT.HPF` bit and setting the corresponding `EMAC_MACFRMFILT.HUC` or `EMAC_MACFRMFILT.HMC`

bits. This configuration applies to both unicast and multicast frames. If the EMAC_MACFRMFILT.HPF bit is reset, only one of the filters (hash or perfect) is applied to the received frame.

NOTE: Hash filtering is not perfect filtering because a 48-bit MAC address is reduced to a 6-bit hash value. Consequently, there may be instances where more than one address has the same hash value.

- **Unicast Destination Address Filter.**
  - The AFM supports 1 MAC address for unicast perfect filtering. If perfect filtering is selected (EMAC_MACFRMFILT.HUC bit is reset), the AFM compares all 48 bits of the received unicast address with the programmed MAC address for any match.
  - In hash filtering mode (When EMAC_MACFRMFILT.HUC bit is set), the AFM performs imperfect filtering for unicast addresses using a 64-bit hash table. For hash filtering, the AFM uses the upper 6 bit CRC of the received destination address to index the content of the hash table. A value of 000000 selects bit 0 of the selected register, and a value of 111111 selects bit 63 of the hash table register. If the corresponding bit (indicated by the 6-bit CRC) is set to 1, the unicast frame is said to have passed the hash filter; otherwise, the frame has failed the hash filter.
- **Multicast Destination Address Filter.**
  - The EMAC can be programmed to pass all multicast frames by setting the EMAC_MACFRMFILT.PM bit. If the EMAC_MACFRMFILT.PM bit is reset, the AFM performs the filtering for multicast addresses based on the EMAC_MACFRMFILT.HMC bit. In perfect filtering mode, the multicast address is compared with the programmed MAC destination address register. Group address filtering is also supported.
  - In hash filtering mode, the AFM performs imperfect filtering using a 64-bit hash table. For hash filtering, the AFM uses the upper 6 bit CRC of the received multicast address to index the content of the hash table. A value of 000000 selects bit 0 of the selected register and a value of 111111 selects bit 63 of the hash table register. If the corresponding bit is set to 1, then the multicast frame is said to have passed the hash filter; otherwise, the frame has failed the hash filter.
- **Broadcast Address Filter.** The AFM doesn't filter any broadcast frames in the default mode. However, if the EMAC is programmed to reject all broadcast frames by setting the EMAC_MACFRMFILT.DBF bit, the AFM asserts the filter fail signal, whenever a broadcast frame is received.
- **Inverse Filtering Operation.** There is an option to invert the filter-match result at the final output. This is controlled by the EMAC_MACFRMFILT.DAIF bit. The this bit is applicable for both unicast and multicast DA frames. The result of the unicast /multicast destination address filter is inverted in this mode.

Destination Address Filtering

The following table provides various address filtering possibilities using the EMAC AFM module. The bits are located in the MAC receive frame filter register (EMAC_MACFRMFILT).

Table 23-27: Destination Address Filtering

Frame Type	Bit Setting (0 = Cleared, 1 = Set, X = Don't Care)							DA Filter Operation
	PR	HPF	HUC	HMC	DAIF	PM	DBF	
Broadcast	1	X	X	X	X	X	X	Pass
	0	X	X	X	X	X	0	Pass
	0	X	X	X	X	X	1	Fail
Unicast	1	X	X	X	X	X	X	Pass all frames
	0	X	0	0	X	X	X	Pass on Perfect/Group filter match
	0	X	0	1	X	X	X	Fail on Perfect/Group filter match
	0	0	1	0	X	X	X	Pass on Hash filter match
	0	0	1	1	X	X	X	Fail on Hash filter match
	0	1	1	0	X	X	X	Pass on Hash or Perfect/Group filter match
	0	1	1	1	X	X	X	Fail on Hash or Perfect/Group filter match

Table 23-27: Destination Address Filtering (Continued)

Frame Type	Bit Setting (0 = Cleared, 1 = Set, X = Don't Care)							DA Filter Operation
	PR	HPF	HUC	HMC	DAIF	PM	DBF	
Multicast	1	X	X	X	X	X	X	Pass all frames
	X	X	X	X	X	1	X	Pass all frames
	0	X	X	0	0	0	X	Pass on Perfect/Group filter match and drop PAUSE control frames if PCF = 0x
	0	0	X	0	1	0	X	Pass on Hash filter match and drop PAUSE control frames if PCF = 0x
	0	1	X	0	1	0	X	Pass on Hash or Perfect/Group filter match and drop PAUSE control frames if PCF = 0x
	0	X	X	1	0	0	X	Fail on Perfect/Group filter match and drop PAUSE control frames if PCF = 0x
	0	0	X	1	1	0	X	Fail on Hash filter match and drop PAUSE control frames if PCF = 0x
	0	1	X	1	1	0	X	Fail on Hash or Perfect/Group filter match and drop PAUSE control frames if PCF = 0x

EMAC Station Management Interface (SMI)

The IEEE 802.3 MII station management interface (applicable for RMII as well), also known as the MDIO management interface, allows the Blackfin processor to monitor and control one or more external Ethernet physical-layer transceivers (commonly called PHYs). The management interface physically consists of a 2-wire serial connection composed of the MDC (management data clock) output signal and the MDIO (management data input/output) bidirectional data signal.

The application can select one of the 32 PHYs and one of the 32 registers within any PHY and send control data or receive status information. Only one register in one PHY can be addressed at any given time. All the transfers are initiated by the EMAC CORE, and the PHY chip only acts as a slave device.

Standard PHY control and status registers typically provide device capability status bits (for example, auto-negotiation, duplex modes, 10/100 speeds and protocols), device status bits (for example, auto-negotiation complete, link status, remote fault), and device control bits (for example, reset, speed selection, loopback, and auto-negotiation start). The features supported by the PHY may be determined at power-up by an MDIO read access (at default rates) of device capabilities in PHY status registers.

The MII management logical interface specifies:

- A set of 16-bit device control/status registers within the PHYs, including both required registers with standardized bit definitions as well as optional vendor-specified registers.
- A 5-bit device addressing scheme which allows the MAC to select one of up to 32 externally-connected PHY devices.
- A 5-bit register addressing scheme for selecting the target register within the addressed device.
- A transfer frame protocol for 16-bit read and write accesses to PHY registers via the MDC and MDIO signals under control of the MAC.

Table 23-28: Station Management Interface pins

Station Management Interface Pins	Pin Description
MDIO – Management Data IO	A periodic clock that runs at a maximum period of 400 ns. Always driven by the EMAC to PHY.
MDC – Management Data Clock	Data signal driven by EMAC or PHY, depending on write or read access with respect to EMAC; synchronous to MDC.

MDC Clock Frequency

The frequency of MDC is determined by the EMAC_SMI_ADDR.CR bit field as shown in the table below. The clock range selection determines the frequency of the clock relative to the SCLK frequency. The suggested range of SCLK frequency applicable for each value of the EMAC_SMI_ADDR.CR field is shown in the table below. The programmability based on SCLK frequency range ensures that the MDC clock frequency range is within the IEEE specifications of 1.0 MHz to 2.4 MHz. However, the EMAC MDC can also support higher frequencies for PHY devices that support the frequencies.

Table 23-29: MDC Clock Frequency Selection

EMAC_SMI_ADDR.CR Selection	Programmed SCLK Frequency Range	Frequency of MDC	Min and Max MDC Freq (Per Specifications)
0000	60–100 MHz	SCLK/42	MIN = 1.43 MHz and MAX = 2.39 MHz

Table 23-29: MDC Clock Frequency Selection (Continued)

EMAC_SMI_ADDR.CR Selection	Programmed SCLK Frequency Range	Frequency of MDC	Min and Max MDC Freq (Per Specifications)
0001	100–125 MHz	SCLK/62	MIN = 1.61 MHz and MAX = 2.01 MHz
0010	20–35 MHz	SCLK/16	MIN = 1.25 MHz and MAX = 2.19MHz
0011	35–60 MHz	SCLK/26	MIN = 1.35 MHz and MAX = 2.31 MHz

MDIO data transfer parameters are provided in the table below. The write and read sequences provided in the tables, **MDIO Write Data Sequence** and **MDIO Read Data Sequence**, are based on these parameters.

Table 23-30: MDIO Frame Parameters

Parameter	Description
IDLE	The MDIO line is three-state (noted as Z in sequence); there is no clock on MDC.
PREAMBLE	32 continuous bits, each of value 1.
START	Start-of-frame is 01.
OPCODE	10 for read and 01 for write.
PHY ADDR	5-bit address select for one of 32 PHYs (noted as AAAAA in sequence).
REG ADDR	Register address in the selected PHY (noted as RRRRR in sequence).
TA	Turnaround is Z0 for read and 10 for write (Z = high impedance).
DATA	Any 16-bit value. Driven by MAC or PHY based on direction (noted as DDD...DDD).

Table 23-31: MDIO Write Data Sequence

IDLE	PREAMBLE	START	OPCODE	PHY ADDR	REG ADDR	TA	DATA	IDLE
Z	111...111	01	01	AAAAA	RRRRR	10	DDD... DDD	Z

Table 23-32: MDIO Read Data Sequence

IDLE	PREAMBLE	START	OPCODE	PHY ADDR	REG ADDR	TA	DATA	IDLE
Z	111...111	01	10	AAAAA	RRRRR	Z0	DDD... DDD	Z

SMI Write Operation

When programs set the EMAC_SMI_ADDR.SMIW (write) and EMAC_SMI_ADDR.SMIB (busy) bits, the Station Management Interface initiates a write operation into the PHY registers with the management frame format (the PHY address, the register address in PHY, and the write data) specified in the IEEE specifications (Section 22.2.4.5 of IEEE standard). The application should not change the EMAC_SMI_ADDR register contents or the EMAC_SMI_DATA register while the transaction is ongoing.

Write operations to the EMAC_SMI_ADDR register or the EMAC_SMI_DATA register during the transfer period are ignored (while the EMAC_SMI_ADDR.SMIB bit is high), and the transaction is completed without any error. After the write operation has completed, the SMI indicates the same by resetting the EMAC_SMI_ADDR.SMIB bit. The EMAC drives the MDIO line for the complete duration of the frame as shown in the following figure.

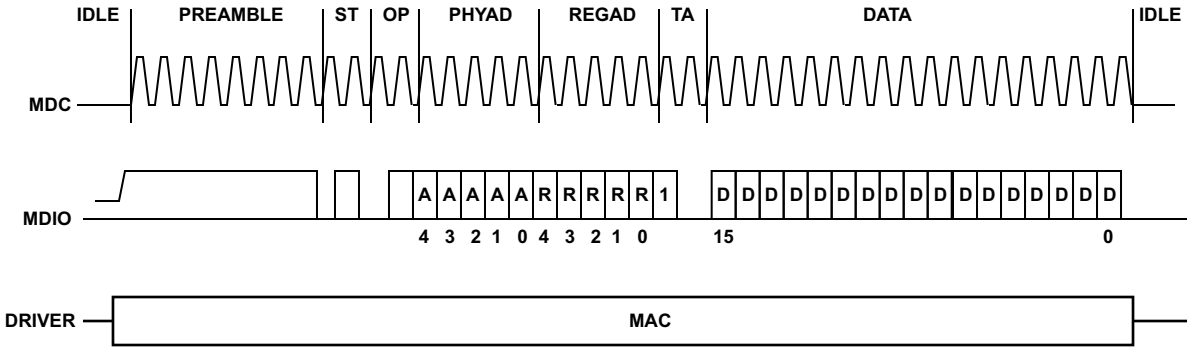


Figure 23-9: SMI Write Operation via MDIO/MDC Pins

SMI Read Operation

When programs set the EMAC_SMI_ADDR.SMIB bit with the EMAC_SMI_ADDR.SMIW bit cleared (=0), the Station Management Interface transfers the PHY address and the register address in the PHY to the SMI to initiate a read operation in the PHY registers. The application should not change the EMAC_SMI_ADDR register contents or the EMAC_SMI_DATA register while the transaction is ongoing.

Write operations to the EMAC_SMI_ADDR register or the EMAC_SMI_DATA register during the transfer period are ignored (while the EMAC_SMI_ADDR.SMIB bit is high) and the transaction is completed without any error. After the read operation has completed, the SMI indicates this by resetting the EMAC_SMI_ADDR.SMIB bit and updates the EMAC_SMI_DATA register with the data read from the PHY. The EMAC drives the MDIO line for the complete duration of the frame except during the data fields when the PHY is driving the MDIO line as shown in the following figure.

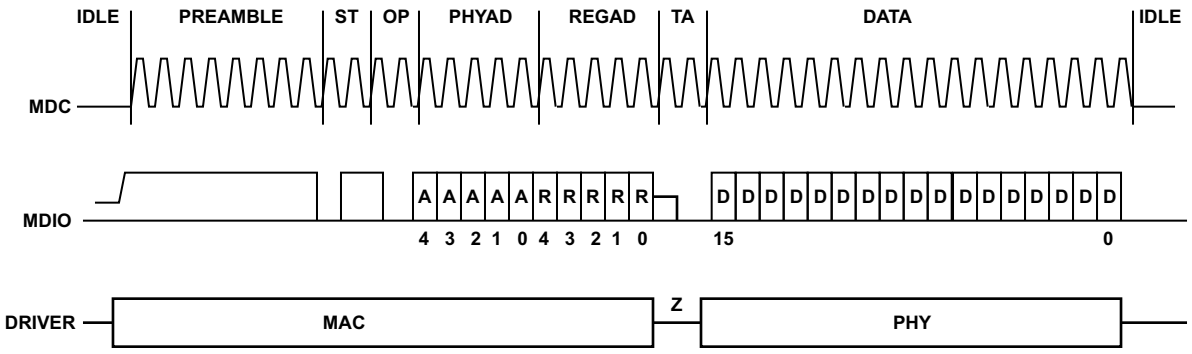


Figure 23-10: SMI Read Operation via MDIO/MDC Pins

EMAC Management Counters (MMC)

The EMAC provides a comprehensive set of 32-bit MAC management counters. These counters are used for gathering statistics on the received and transmitted frames. The MMC sub-block also includes a control register (`EMAC_MMC_CTL`) for controlling the behavior of the counters, two 32-bit registers containing interrupts generated (`EMAC_MMC_RXINT` and `EMAC_MMC_TXINT`), and two 32-bit registers containing masks for the interrupt register (`EMAC_MMC_RXIMSK` and `EMAC_MMC_TXIMSK`).

The MMC receive counters are updated for frames that are passed by the address filtering sub-block in the EMAC CORE. Statistics of frames that are dropped by the AFM module are not updated unless they are runt frames of less than 6 bytes (destination address bytes are not received fully). The module is also capable of gathering statistics on encapsulated IPv4, IPv6, and TCP, UDP, or ICMP payloads in received Ethernet frames.

Please refer to the “Register Descriptions” section for all the statistical counters available in EMAC. The MMC register naming conventions are as follows:

- TX as a prefix or suffix indicates counters associated with transmission.
- RX as a prefix or suffix indicates counters associated with reception.
- _G as a suffix indicates registers that count good frames only.
- _GB as a suffix indicates registers that count frames regardless of whether they are good or bad.

Transmitted frames are considered *good* if transmitted successfully. In other words, a transmitted frame is good if the frame transmission is not aborted due to any of the following errors:

- Jabber Timeout
- No Carrier/Loss of Carrier
- Late Collision
- Frame Underflow
- Excessive Deferral

- Excessive Collision

Received frames are considered good if none of the following errors exists:

- CRC error
- Runt Frame (shorter than 64 bytes)
- Alignment error
- Length error (non-Type frames only)
- Out of Range (non-Type frames only, longer than maximum size)

The maximum frame size depends on the frame type, as follows:

- Untagged frame maxsize = 1518
- VLAN Frame maxsize = 1522
- Jumbo Frame maxsize = 9018
- Jumbo VLAN Frame maxsize = 9022

The EMAC_MMC_CTL register also contains bits that control preset, freeze and roll-over of counters. Additional configuration include EMAC_MMC_CTL.RDRST bit that enables an auto-reset feature whenever the counters are read and the EMAC_MMC_CTL.RST bit for resetting all the counters.

The MMC can also trigger an interrupt when the corresponding bits are enabled in the transmit, receive and IPC mask registers, and when the particular counter reaches half/full. The status is also updated in the corresponding interrupt register.

MMC Receive Interrupt Register

The EMAC_MMC_RXINT register maintains the interrupts that are generated when receive statistic counters reach half their maximum values (0x80000000), and when they cross their maximum values (0xFFFFFFFF). When EMAC_MMC_CTL.NOROLL is set, then interrupts are set but the counter remains at all-ones. The EMAC_MMC_RXINT register is a 32-bit wide register. An interrupt bit is cleared when the respective MMC counter that caused the interrupt is read. The least significant byte lane (bits 7–0) of the respective counter must be read in order to clear the interrupt bit.

MMC Transmit Interrupt Register

The EMAC_MMC_TXINT register maintains the interrupts generated when transmit statistic counters reach half their maximum values (0x80000000), and when they cross their maximum values (0xFFFFFFFF). When EMAC_MMC_CTL.NOROLL is set, then interrupts are set but the counter remains at all-ones. The EMAC_MMC_TXINT register is a 32-bit wide register. An interrupt bit is cleared when the respective MMC counter that caused the interrupt is read. The least significant byte lane (bits 7–0) of the respective counter must be read in order to clear the interrupt bit.

MMC Receive Checksum Offload Interrupt Register

The EMAC_MMC_RXINT.CRCERR register maintains the interrupts generated when receive IPC statistic counters reach half their maximum values (0x80000000), and when they cross their maximum values (0xFFFFFFFF). When EMAC_MMC_CTL.NOROLL is set, then interrupts are set but the counter remains at all-ones. The EMAC_MMC_RXINT.CRCERR register is 32-bits wide. When the MMC IPC counter that caused the interrupt is read, its corresponding interrupt bit is cleared. The counter's least-significant byte lane (bits 7–0) must be read to clear the interrupt bit.

EMAC Precision Time Protocol (PTP) Engine

The following sections describe the Precision Time Protocol engine.

IEEE1588 and the PTP Engine

The Ethernet MAC peripheral includes a PTP Engine to assist applications requiring time synchronization. The PTP module is tightly integrated with the EMAC CORE to aid hardware time stamping defined in the IEEE1588 2002/2008 standards. Applications can make use of accurate hardware time stamps via TCP/IP stacks (if using Network layer communication) or via Ethernet device drivers (if using MAC layer communication), to exchange time information across devices connected over network.

PTP Engine

For calculation of drift in time between two Ethernet devices, the device should note down its system time whenever a timing message is sent or received (IEEE 1588 protocol). Due to the indeterministic delay of a node's software system, the software is unable to capture an accurate time when the message is sent or received. However, the hardware is capable of monitoring the signal on the communication media and get accurate message arrival/departure time.

The PTP (Precision Time Protocol) module is closely integrated with the EMAC module and provides hardware assistance to implement both the IEEE 1588-2002 and IEEE 1588-2008 standards on Ethernet (IEEE 802.3). It takes one input clock signal as its PTP clock and maintains the timing information (called *system time*) at the nanosecond level.

The PTP module includes hardware for clock and system time adjustment. The system time is physically represented by Pulse -Per-Second (PPS) signal. PPS can be programmed to a fixed frequency or provide flexibility to the signal in terms of pulse width, interval, start and stop time of the signal. The PTP module can be programmed to trigger an alarm interrupt when system time reaches specified time.

The PTP module can be programmed to detect different types of received frames, capture the system time and timestamp those frames with the captured system time. Programs can configure any frame so that the PTP module capture the system time when it is transmitted. The PTP module can also capture the system time when an event is detected on the Auxiliary Snapshot Trigger input pin (ETH_PTPAUXIn).

IEEE1588 Standard

Many systems require two independent devices to operate in a time synchronized fashion. If each system were to rely solely on its own oscillator, differences between the specific characteristics and operating conditions of the individual oscillators would limit the ability of the clocks to operate synchronously. To serve applications requiring synchronized clocks, a periodic correction mechanism is employed.

A simple way to synchronize multiple systems is to choose one system (with the best clock) as a master. The system master broadcasts the clock and timing information to other systems (slaves) and then the slaves adjust their clocks and timing according to that of master. However, this method has limitations such as the master cannot broadcast the time at infinitesimal intervals, path delay (propagation delay) exists between a master and a slave, and the delay varies between each slave and master.

IEEE 1588 (also known as Precision Time Protocol or PTP) standard specifies a protocol used to synchronize the time and clock of multiple devices, dispersed but interconnected by any communication, for example, Ethernet (IEEE 802.3). According to the protocol, timing messages are exchanged between two devices (both devices should have the same representation of their system time), and then one of the device calculates its drift from other device and corrects its system time. The protocol resolves path delay between devices and also helps synchronize the clocks of multiple devices and all of the limitations mentioned above are resolved.

IEEE 1588 was published in 2002 where four types of timing messages were defined: Sync, Follow_Up, Delay_Req, and Delay_Resp. Here the protocol synchronizes two or more devices where one is a master and others are slaves. The Sync, Follow_Up, and Delay_Resp messages are sent from the master device to the slave device in the system, while the Delay_Req message is sent from a slave device to master device. More information on IEEE 1588-2002 is provided in a following section.

In 2008 a newer version of IEEE 1588 was introduced which provides further mechanisms to measure the peer-to-peer delay. Three additional timing messages (PdelayReq, PdelayResp, and PdelayRespFollowup) were added to implement peer-to-peer synchronization. More information on IEEE 1588-2008 is provided in a following section.

IEEE 1588-2002

The IEEE 1588-2002 standard defines the Precision Time Protocol (PTP) that allows precise synchronization of clocks in measurement and control systems that use network communication, local computing, and distributed objects. The protocol applies to systems that communicate by local area networks that support multicast messaging, including (but not limited to) Ethernet. This protocol also allows heterogeneous systems that include clocks of varying inherent precision, resolution, and stability to synchronize. The protocol supports system-wide synchronization accuracy in the sub-microsecond range with minimal network and local clock computing resources.

The PTP is transported over UDP/IP. The system or network is classified into master and slave nodes for distributing the timing/clock information. The following figure shows the process that PTP uses for synchronizing a slave node to a master node by exchanging PTP messages.

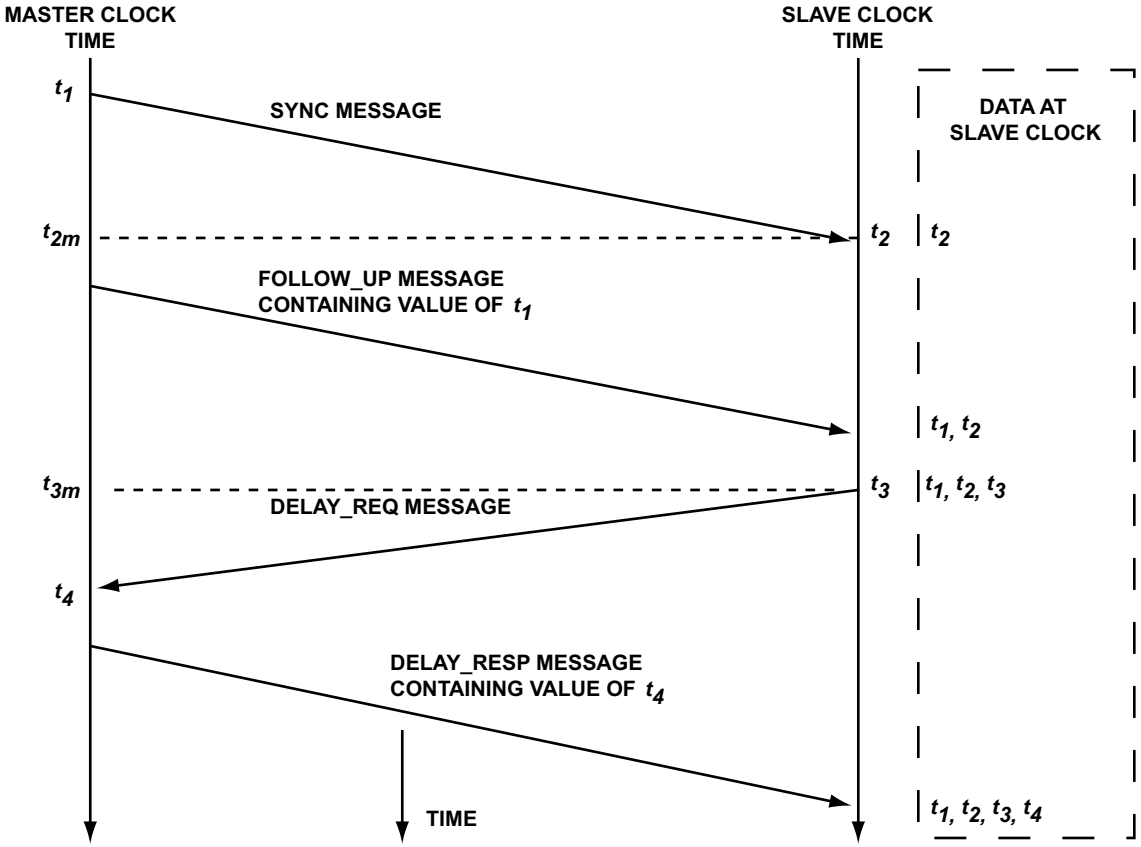


Figure 23-11: IEEE 1588-2002 PTP Process

As shown in the figure, the PTP uses the following process:

1. The master broadcasts the PTP Sync messages to all its nodes. The Sync message contains the master's reference time information. The time at which this message leaves the master's system is t_1. This time must be captured by the Master, for Ethernet ports, at RMII.
2. The slave receives the Sync message and also captures the exact time, t_2, using its timing reference.
3. The master sends a Follow_up message to the slave, which contains t1 information for later use.
4. The slave sends a Delay_Req message to the master, noting the exact time, t_3, at which this frame leaves the RMII.
5. The master receives the message, capturing the exact time, t_4, at which it enters its system.
6. The master sends the t_4 information to the slave in the Delay_Resp message.
7. The slave uses the four values of t_1, t_2, t_3, and t_4 to synchronize its local timing reference to the master's timing reference.

Most of the PTP implementation is done in the software above the UDP layer. However, the hardware support is required to capture the exact time when specific PTP packets enter or leave the Ethernet port at

the RMII. This timing information must be captured and returned to the software for the proper implementation of PTP with high accuracy.

IEEE 1588-2008 Advanced Timestamps

In addition to the basic timestamp features mentioned in IEEE 1588-2002 Timestamps, the EMAC supports the following advanced timestamp features defined in the IEEE 1588-2008 standard.

- Support for the IEEE 1588-2008 (Version 2) timestamp format.
- Provides an option to take snapshot of all frames or only PTP type frames.
- Provides an option to take snapshot of only event messages.
- Provides an option to select the node to be a master or slave.
- Identifies the PTP message type, version, and PTP payload in frames sent directly over Ethernet and sends the status.
- Provides an option to run nanoseconds time in digital or binary format.

Peer-to-Peer (P2P) PTP Message Support

The IEEE 1588-2008 version supports Peer-to-Peer PTP (Pdelay) message in addition to SYNC, Delay Request, Follow-up, and Delay Response messages. Figure below shows the method to calculate the propagation delay between nodes supporting peer-to-peer path correction.

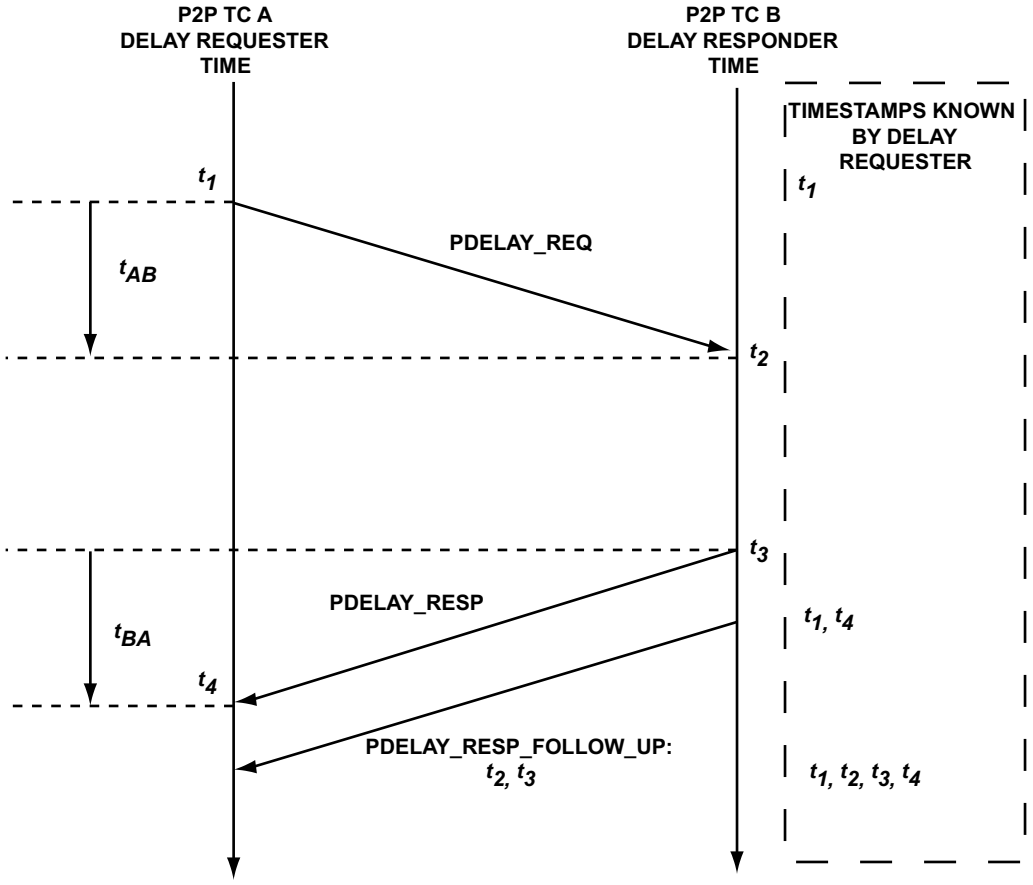


Figure 23-12: Propagation Delay Calculation between Nodes Supporting Peer-to-Peer Path Correction

As shown in Figure above, the propagation delay is calculated in the following way:

1. Port-1 issues a Pdelay_Req message and generates a timestamp, t1, for the Pdelay_Req message.
2. Port-2 receives the Pdelay_Req message and generates a timestamp, t2, for this message.
3. Port-2 returns a Pdelay_Resp message and generates a timestamp, t3, for this message. To minimize errors because of any frequency offset between the two ports, Port-2 returns the Pdelay_Resp message as quickly as possible after the receipt of the Pdelay_Req message. The Port-2 returns any one of the following:
    - The difference between the timestamps t2 and t3 in the Pdelay_Resp message.
    - The difference between the timestamps t2 and t3 in the Pdelay_Resp_Follow_Up message.
    - The timestamps t2 and t3 in the Pdelay_Resp and Pdelay_Resp_Follow_Up messages respectively.
4. Port-1 generates a timestamp, t4, on receiving the Pdelay_Resp message.

Port-1 uses all four timestamps to compute the mean link delay.

Block Diagram

The following figure shows the functional block diagram of PTP module.

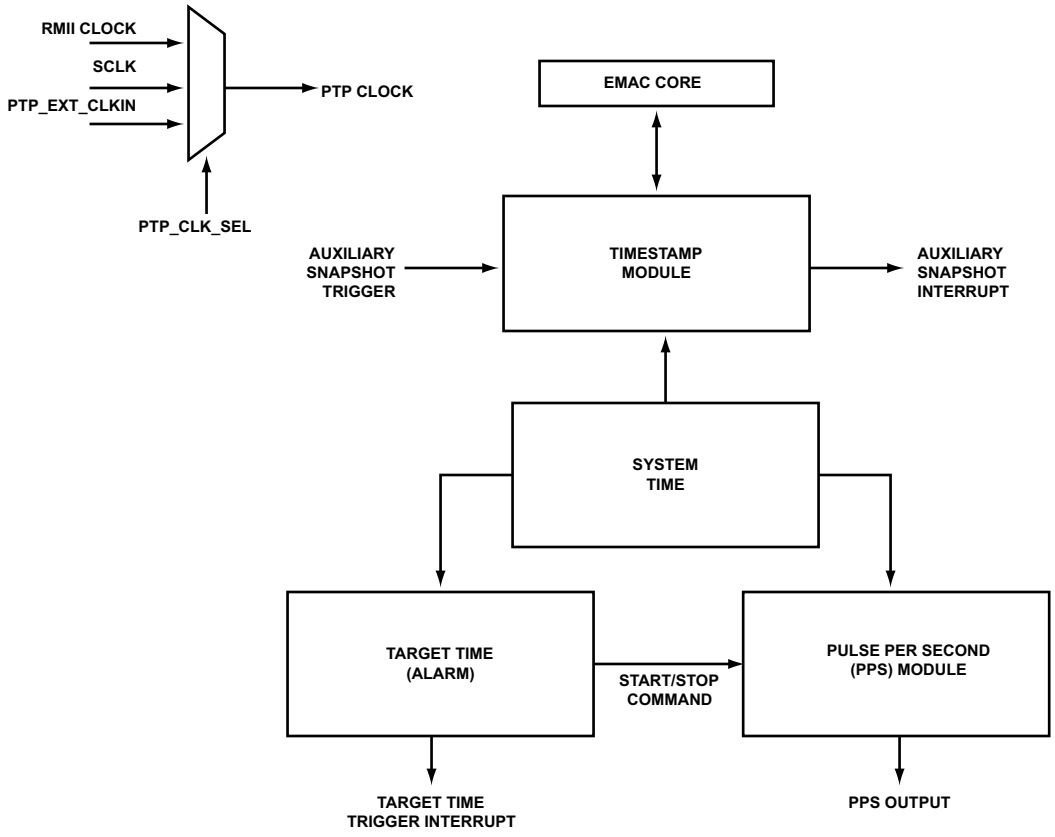


Figure 23-13: PTP Block Diagram

A system time module is present which keeps the time of PTP module. It consists of hardware which can be programmed for time initialization, time correction and clock correction.

The timestamp module is capable of capturing the time (provided by the system time module) at various conditions such as when a frame is sent or received by the EMAC, or the rising edge of the auxiliary snapshot trigger (`ETH_PTPAUXIn`) pin. When system time is captured after detection of a frame, the timestamp module automatically includes the time information in the frame descriptor. Time stamping on the detection of a frame can be programmed on a per frame basis.

The PTP module is driven by PTP clock. This clock can be selected from three different clock sources.

The Pulse per Second (PPS) module is used to generate a pulse or train of pulse on the PPS output pin, (`ETH_PTPPPS`) and it is the physical representation of system time. PPS can be fixed (where only frequency can be varied) or flexible (where width, interval, start time and stop time can be programmed).

The Target Time module acts as an alarm for the PTP module. Whenever system time reaches a value equal to programmed target time, the target time trigger interrupt is generated. By appropriate programming, The target time trigger can also be used to start or stop flexible PPS output at specific time.

PTP Module Clock

The PTP module clock features include *Clock Source Selection* and *Clock Frequency Range*.

Clock Source Selection

The PTP module can take one of three clock sources as its input clock — *SCLK*, RMII clock or PTP external clock.

The PTP clock source can be selected using the PADS_EMAC_PTP_CLKSEL.EMAC0 or PADS_EMAC_PTP_CLKSEL.EMAC1 field, as shown in table below.

Table 23-33: PTP Clock Source Selection

PADS_EMAC_PTP_CLKSEL.EMAC0 **or** PADS_EMAC_PTP_CLKSEL.EMAC1 **Field**	**PTP Clock Source**	**Clock Description**
00	EMAC_RMII	RMII reference clock
10	PTP External Clock	Clock available on PTP_EXT_CLK pin
X1	SCLK	Processor System Clock

Clock Frequency Range

The resolution, or granularity, of the reference time source determines the accuracy of the synchronization. Therefore, a higher PTP clock frequency gives better system performance. The maximum PTP clock frequency is limited by the timing constraints achievable for logic operating on the selected PTP clock source.

The minimum PTP clock frequency depends on the time required between two consecutive frames. Because the RMII clock frequency is fixed by the IEEE specification, the minimum PTP clock frequency required for proper operation depends upon the operating mode and operating speed of the MAC as shown in the following table.

A minimum delay required between two consecutive timestamp captures is 8 clock cycles of RMII and 3 clock cycles of PTP clocks. If the delay between two timestamp captures is less than this delay, the EMAC does not take a timestamp snapshot for the second frame.

Table 23-34: Minimum PTP Clock Frequency

Mode	**Minimum Gap Between Two Frames**	**Minimum PTP Frequency**
100-Mbps full-duplex operation	336 RMII clocks (256 clocks for a 64-byte frame + 48 clocks of min IFG + 32 clocks of preamble)	(3 × PTP period) + (8 × RMII period) ≤ (336 × RMII period) Maximum PTP period = (336 – 8) × 20 ns ÷ 3 = 2,186 ns Minimum PTP frequency = 0.46 MHz

Timestamp Module

The timestamp module captures time in seconds and nanoseconds maintained as system time. The timestamp module also captures time when specific events occur. Events include detection of a frame transmitted or received over the EMAC and a rising edge on the ETH_PTPAUXIn pin. The timestamp module does not need to timestamp all of the transmitted or received frames over the EMAC. The PTP module can be programmed to detect specific kinds of frames for timestamping.

This kind of frame detection is discussed in the following sections.

Frame Detection and Timestamping

The PTP module automatically monitors all received and transmitted IEEE 1588 event messages on the Ethernet. If an event message is detected, the module takes a snapshot of the system time and stores its value to the 64-bit fields in Transmit or receive descriptor. The timestamping is done at the EMAC RMII interface when the module sees the start of frame of an event message packet.

Transmit Path Timestamping

The EMAC captures a timestamp when a frame is being sent on RMII. Timestamp capture is controllable on a per-frame basis. In other words, each transmit frame can be marked to indicate whether a timestamp should be captured for that frame or not.

Applications should extend the descriptor word length from 4 words to 8 words by setting the EMAC_DMA_BUSMODE.ATDS bit. In order to enable the timestamp function, the TTSE (transmit timestamp enable) bit in transmit descriptor word TDES0 should be set. When the PTP module captures a timestamp of a transmitted frame, it notifies the application by setting the TTSS (transmit timestamp status) in TDES0.

The EMAC returns the timestamp to the software inside the corresponding transmit descriptor, automatically connecting the timestamp to the specific frame. The 64-bit timestamp information is written to the TDES6 and TDES7 fields. The TDES6 field holds the 32 LSBs of the timestamp (system time nanoseconds), except as described in transmit timestamp field and TDES7 field holds the 32 MSBs (system time seconds). After the PTP module timestamps the frame, the application can get the timestamp along with the transmit status from the EMAC.

NOTE: The PTP module timestamps all the transmitting frames that has TTSE set in its TDES0. It does not distinguish according to the type of transmitting frame.

Auxiliary Timestamp Snapshot

The auxiliary snapshot feature stores snapshots of the system time whenever a rising edge is detected on the ETH_PTPAUXIn pin.

The PTP stores 64-bits of captured timestamp in a 4-deep FIFO. When a snapshot is stored, the PTP indicates this to the EMAC with the auxiliary snapshot interrupt and the EMAC_TM_STMPSTAT.ATSTS bit is set. The value of the snapshot is read through the EMAC_TM_AUXSTMP_SEC and EMAC_TM_AUXSTMP_NSEC registers. If the FIFO becomes full and an external trigger to take the snapshot is asserted, then the snapshot

trigger-missed status is set in the EMAC_TM_STMPSTAT.ATSSTM bit. The latest snapshot is not written to the FIFO when it is full.

When a host reads the 64-bit timestamp from the FIFO through the EMAC_TM_AUXSTMP_SEC and EMAC_TM_AUXSTMP_NSEC registers, the space becomes available to store the next snapshot.

NOTE: A space in the FIFO is created whenever the EMAC_TM_AUXSTMP_SEC register is read. Therefore the EMAC_TM_AUXSTMP_NSEC register should be read before reading the EMAC_TM_AUXSTMP_SEC register.

The program can clear the FIFO by setting the EMAC_TM_CTL.ATSFC bit. When multiple snapshots are present in the FIFO, the count is indicated in the EMAC_TM_STMPSTAT.ATSNS bits.

NOTE: The minimum gap between two events on the ETH_PTPAUXIn pin must be 4 cycles of PTP_CLK + 3 cycles of SCLK). Otherwise, the rising-edge of the trigger is missed by the logic.

Receive Path Timestamping

The PTP module automatically monitors all received and transmitted IEEE 1588 event messages on the Ethernet. If an event message is detected, the module takes a snapshot of the System Time and stores its value to the 64-bit fields in transmit or receive descriptor. The timestamping is done at the EMAC RMII interface when the module sees the start of frame of an event message packet.

PTP module captures the timestamp of received frames on the RMII. Timestamp capture is controllable on a per-frame and per-type basis. In other words each received frame is timestamped according to the frame type.

Applications should extend the descriptor word length from 4 words to 8 words by setting EMAC_DMA_BUSMODE.ATDS to store timestamp and received message status. The PTP notifies the application of receive time stamp availability when it sets bit 7 (timestamp available) in receive descriptor word RDES0.

When bit 0 (extended status available) is set in RDES0, it indicates that the extended status of the PTP frame is provided in the RDES4 word. Extended status include PTP Version, PTP frame type and message type. The EMAC returns the timestamp to the software inside the corresponding receive descriptor. The 64-bit timestamp information is written back to the RDES6 and RDES7 fields in memory. The RDES6 holds the 32 LSBs of the timestamp (system time nanoseconds), except as mentioned in receive timestamp field and RDES7 field holds 32 MSBs (system time seconds).

The timestamp is written only to that receive descriptor for which the last descriptor status field has been set to 1. When the timestamp is not available (for example, because of an RxFIFO overflow), an all-ones pattern is written to the descriptors (RDES6 and RDES7), indicating that timestamp is not correct. RDES0 [7] indicates whether the time-stamp is updated in RDES6/7 or not.

Processing of received frames to identify valid PTP frames is done by the PTP module. The snapshot of the time to be sent to the application can be controlled using the EMAC_TM_CTL register.

The PTP module can be programmed to detect all received frames or only some types of PTP frames, according to bit settings in the EMAC_TM_CTL register. Refer to the following table.

Table 23-35: PTP Frame Type Selections

TSENALL (bit 8)	SNAPTYPSEL (bits [17:16])	TSMSTRENA (bit 15)	TSEVNTENA (bit 14)	Frames
1	X	X	X	All
0	00	X	0	Sync, Follow_Up, Delay_Req, Delay_ Resp
0	00	0	1	Sync
0	00	1	1	Delay_Req
0	01	X	0	Sync, Follow_Up, Delay_Req, Delay_ Resp, Pdelay_Req, Pdelay_Resp, Pdelay_Resp_ Follow_Up
0	01	0	1	Sync, Pdelay_Req, Pdelay_Resp
0	01	1	1	Delay_Req, Pdelay_ Req, Pdelay_Resp
0	10	X	X	Sync, Delay_Req
0	11	X	X	Pdelay_Req, Pdelay_Resp

PTP Processing and Control

When the EMAC receives a frame, frame detection and timestamping by timestamp module of the PTP is done on the basis of some of the PTP fields in the frame. The **PTP Message Format (IEEE 1588-2008)** table shows the common message header for the PTP messages. This format is taken from IEEE standard 1588-2008. When the EMAC needs to send a PTP frame, the frame has to follow this format.

When a frame is received, PTP module compares these fields with standard values and finds out the type of PTP frame and other information such as PTP version, IP version, and others. It then updates the related fields in RDES4. When a frame is transmitted programs should ensure that all the fields are appropriate so that a PTP module on the other end of a communication can correctly detect and decode the frame.

NOTE: (*) – controlField is used in version 1. For version 2, messageType field will be used for detecting different message types.

Table 23-36: PTP Message Format (IEEE 1588-2008)

Bits								Octets	Offset
7	6	5	4	3	2	1	0		

Table 23-36: PTP Message Format (IEEE 1588-2008) (Continued)

Bits		Octets	Offset
transportSpecific	messageType	1	0
Reserved	versionPTP	1	1
messageLength		2	2
domainNumber		1	4
Reserved		1	5
flagField		2	6
correctionField		8	8
Reserved		4	16
sourcePortIdentity		10	20
sequenceId		2	30
controlField (*)		1	32
logMessageInterva		1	33

There are some fields in the Ethernet payload that user can use to detect the PTP packet type and control the snapshot to be taken. These fields are different for the following PTP frames:

- PTP Frames Over IPv4
- PTP Frames Over IPv6
- PTP Frames Over Ethernet

For any of the above PTP frames, EMAC does not consider the PTP version 1 messages as valid PTP messages when frame consists of Peer delay multicast address as destination address (DA).

PTP Frame Over IPv4

The **IPv4-UDP PTP Frame Fields Required for Control and Status** table provides information about the fields that are matched to control snapshot for the PTP packets sent over UDP over IPv4 for IEEE 1588 version 1 and 2. The octet positions for the tagged frames are offset by 4. This is based on IEEE 1588-2008 standards and the message format defined in the **PTP Message Format (IEEE 1588-2008)** table in the *PTP Processing and Control* section.

NOTE: (*) - PTP event messages are SYNC, Delay_Req (IEEE 1588 version 1 and 2) or Pdelay_Req, Pdelay_Resp (IEEE 1588 version 2 only)

Table 23-37: IPv4-UDP PTP Frame Fields Required for Control and Status

Field Matched	Octet Position	Matched Value	Description
MAC Frame type	12, 13	0x0800	IPv4 datagram

Table 23-37: IPv4-UDP PTP Frame Fields Required for Control and Status (Continued)

Field Matched	Octet Position	Matched Value	Description
IP Version and Header Length	14	0x45	IP version is IPv4
Layer 4 Protocol	23	0x11	UDP
IP Multicast Address (IEEE 1588 Version 1)	30, 31, 32, 33	0xE0,0x00, 0x01,0x81 (or 0x82 or 0x83 or 0x84)	Multicast IPv4 addresses allowed. 224.0.1.129 224.0.1.130 224.0.1.131 224.0.1.132
IP Multicast Address (IEEE 1588 Version 2)	30, 31, 32, 33	0xE0, 0x00, 0x01, 0x81 (Hex) 0xE0, 0x00, 0x00, 0x6B (Hex)	PTP-Primary multicast address: 224.0.1.129 PTP-Peer delay multicast address: 224.0.0.107
UDP Destination Port	36, 37	0x013F 0x0140	0x013F - PTP Event Messages (*) 0x0140 - PTP general messages
PTP Control Field (IEEE version 1)	74	0x00/0x01/0x02/ 0x03/0x04	0x00 - SYNC 0x01 - Delay_Req 0x02 - Follow_Up 0x03 - Delay_Resp 0x04 - Management
PTP Message Type Field (IEEE version 2)	42 (nibble)	0x0/0x1/0x2/0x3/0x8/0x9/ 0xA/0xB/ 0xC/0xD	0x0 - SYNC 0x1 - Delay_Req 0x2 - Pdelay_Req 0x3 - Pdelay_Resp 0x8 - Follow_Up 0x9 - Delay_Resp 0xA - Pdelay_Resp_Follow_Up 0xB - Announce 0xC - Signaling 0xD - Management
PTP Version	43 (nibble)	0x1 or 0x2	0x1 - Supports PTP version 1 0x2 - Supports PTP version 2

PTP Frame Over IPv6

The **IPv6-UDP PTP Frame Fields Required for Control And Status** table provides information about the fields that are matched to control the snapshots for the PTP packets sent over UDP over IPv6 for IEEE 1588 version 1 and 2. The octet positions for the tagged frames are offset by 4. This is based on IEEE 1588-2008 standards and the message format defined in PTP Message Format (IEEE 1588-2008).

NOTE: (*) - IPv6 extension header is not defined for PTP packets.

Table 23-38: IPv6-UDP PTP Frame Fields Required for Control and Status

Field Matched	Octet Position	Matched Value	Description
MAC Frame type	12, 13	0x86DD	IP datagram
IP Version	14(bits [7:4])	0x06	IP version is IPv6
Layer 4 Protocol	20 (*)	0x11	UDP
PTP Multicast Address	38–53	FF0x:0:0:0:0:0:0:0:181 (Hex) FF02:0:0:0:0:0:0:0:6B (Hex)	PTP - primary multicast address: FF0x:0:0:0:0:0:0:0:181 (Hex) PTP - Peer delay multicast address: FF02:0:0:0:0:0:0:0:6B (Hex)
UDP Destination Port	56, 57 (*)	0x013F, 0x0140	0x013F - PTP event messages 0x0140 - PTP general messages
PTP Control Field (IEEE 1588 version 1)	93 (*)	0x00/0x01/0x02/ 0x03/0x04	0x00 - SYNC 0x01 - Delay_Req 0x02 - Follow_Up 0x03 - Delay_Resp 0x04 - Management (version1)
PTP Message Type Field (IEEE version 2)	74(*)(nibble)	0x0/0x1/0x2/0x3/0x8/0x9/ 0xA/0xB/ 0xC/0xD	0x0 - SYNC 0x1 - Delay_Req 0x2 - Pdelay_Req 0x3 - Pdelay_Resp 0x8 - Follow_Up 0x9 - Delay_Resp 0xA - Pdelay_Resp_Follow_Up 0xB - Announce 0xC - Signaling 0xD - Management
PTP Version	75(nibble)	0x1 or 0x2	0x1 - Supports PTP version 1 0x2 - Supports PTP version 2

PTP Frame Over Ethernet

The following table provides information about the fields that are matched to control the snapshots for the PTP packets sent over Ethernet for IEEE 1588 version 1 and 2. The octet positions for the tagged frames are offset by 4. This is based on IEEE 1588-2008 standards and the message format defined in the table.

NOTE: (*) - The address match of destination address (DA) programmed in MAC address 0 is used if the `EMAC_TM_CTL.TSENMACADDR` bit is set.

Table 23-39: Ethernet PTP Frame Fields Required for Control and Status

Field Matched	Octet Position	Matched value	Description
MAC Destination Multicast Address(*)	0–5	01-1B-19-00-00-00 01-80-C2-00-00-0E	All PTP messages can use any of the following multicast addresses: 01-1B-19-00-00-00 01-80-C2-00-00-0E
MAC Frame Type	12, 13	0x88F7	PTP Ethernet frame
PTP control field (IEEE Version 1)	45	0x00/0x01/0x02/ 0x03/ 0x04	0x00 - SYNC 0x01 - Delay_Req 0x02 - Follow_Up 0x03 - Delay_Resp 0x04 - Management
PTP Message Type Field (IEEE version 2)	14(nibble)	0x0/0x1/0x2/0x3/0x8/ 0x9/0xA/0xB/ 0xC/0xD	0x0 - SYNC 0x1 - Delay_Req 0x2 - Pdelay_Req 0x3 - Pdelay_Resp 0x8 - Follow_Up 0x9 - Delay_Resp 0xA - Pdelay_Resp_Follow_Up 0xB - Announce 0xC - Signaling 0xD - Management
PTP Version	15(nibble)	0x1 or 0x2	0x1 - Supports PTP version 1 0x2 - Supports PTP version 2

System Time

To get a snapshot of the time, the EMAC requires a reference time in 64-bit format as defined in the IEEE 1588 specification. The PTP module maintains 80-bit time, known as system time and it is updated using the PTP clock.

The 80-bit timing reference is split into the following three registers:

- `EMAC_TM_NSEC` – 32-bit nanoseconds register which provides time in nanoseconds
- `EMAC_TM_SEC` – 32-bit seconds register which provides time in seconds
- `EMAC_TM_HISEC` – 16-bit high seconds register which provides time beyond the seconds register. This register is not included in the IEEE 1588 standard, and its use is application specific.

The 64-bit system time (seconds and nanoseconds) is the source for taking timestamps for Ethernet frames being transmitted or received at the RMII.

Since the PTP clock frequency does not correspond to a 1 ns period, the EMAC_TM_NSEC register should be incremented with a value equal to the PTP clock period for every PTP clock cycle. This is achieved by use of EMAC_TM_SUBSEC register. The EMAC_TM_NSEC value is incremented with value programmed in EMAC_TM_SUBSEC register every PTP clock cycle.

Whenever the EMAC_TM_SEC register overflows from 0xFFFFFFFF to 0x00000000, the seconds overflow interrupt is triggered and indicated by the EMAC_TM_STMPSTAT.TSSOVF bit. After a seconds overflow the EMAC_TM_HISEC register increments by one.

The system time module supports the following two types of rollover modes for the EMAC_TM_NSEC register. digital rollover and binary rollover.

- Digital rollover mode. The maximum value in the nanoseconds field is 0x3B9AC9FF, that is, 10^9 nanoseconds. After it reaches this value, the EMAC_TM_SEC register is incremented and the EMAC_TM_NSEC register restarts counting from zero. Accuracy in digital rollover mode it is 1 ns per bit.
- Binary rollover mode. The nanoseconds field rolls over and increments the seconds field after the value reaches 0x7FFFFFFF. Accuracy in binary rollover mode is ~0.466 ns per bit.

System Time Adjustment

The following sections describe the process for system time adjustment.

System Time Initialization

System time can be initialized with 64-bit time when the PTP module is enabled. The initial value is written to the EMAC_TM_SECUPDT and EMAC_TM_NSECUPDT system time update registers. The system time counter is written with the value in these registers when the EMAC_TM_CTL.TSINIT bit is set.

Coarse Correction Method

If slave system time has an offset with respect to the master's system time, then it can be corrected using the coarse correction method. The time offset value is written to the EMAC_TM_SECUPDT and EMAC_TM_NSECUPDT registers. The offset value is then added to or subtracted from the system time when the EMAC_TM_CTL.TSUPDT bit is set. Addition or subtraction can be chosen by using the EMAC_TM_NSECUPDT.ADDSUB bit. System time correction is done in one clock cycle using the coarse correction method.

NOTE: During subtraction, the EMAC_TM_SECUPDT register value should be less than the value of the EMAC_TM_SEC register. This should be checked prior to performing subtraction through coarse correction.

Fine Correction Method

If a slave PTP clock's frequency has a drift with respect to the master PTP clock (as defined in IEEE 1588), then it can be corrected using the fine correction method. Using this method, system time is corrected over a period of time (unlike coarse correction where it is done in one clock cycle). This helps maintain linear time and does not introduce drastic changes (or a large jitter) in the reference time between PTP Sync message intervals.

Using this method, an accumulator sums the contents of the EMAC_TM_ADDEND register, as shown in the algorithm illustrated in the figure below. The arithmetic carry that the accumulator generates is used as a pulse to increment the system time counter. The accumulator and the addend are 32-bit registers. Here, the accumulator acts as a high-precision frequency divider.

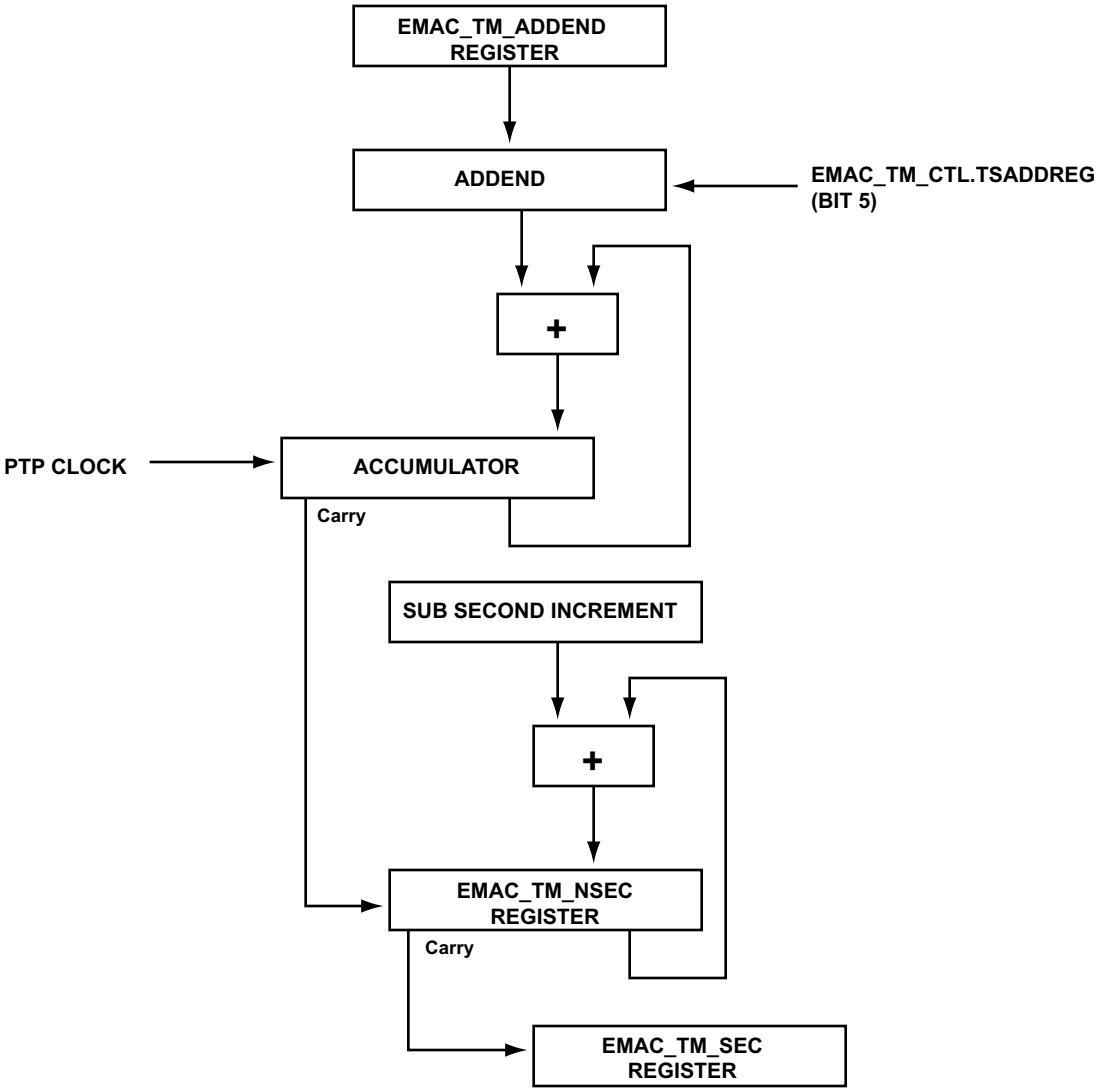


Figure 23-14: System Time Update, Fine Correction Method

Calculating Addend Value

This section describes the process for system time adjustment.

In this example, the master clock runs at 50 MHz and the slave clock has drifted to 66MHz. The goal is to adjust the slave system time to 50 MHz, so that the slave PTP module is synchronized with the master. Using the figure in *Fine Correction Method*, the nanoseconds increment signal should run at 50 MHz. The nanoseconds increment is the carry from accumulator register, which is incremented by the addend value at the rate of the slave clock (66 MHz).

The accumulator overflows and generates a carry every N addend values so N × Addend = 2^{32}.

The accumulator increments at 66 MHz. To bring the carry to 50 MHz N = 66/50 = 1.32.

Here the addend = 2^{32}/1.32 = 0xC1F07C1F.

Therefore, if addend is programmed with 0xC1F07C1F, the slave system time runs at 50 MHz which is synchronized with the master.

In the *Fine Correction Method* figure, the sub second increment is the value programmed in the EMAC_TM_SUBSEC register which increments the EMAC_TM_NSEC register according to the frequency of the nanoseconds increment signal.

In the example, the sub second increment should be 20 (for digital rollover) or 43 (for binary rollover). This increments the EMAC_TM_NSEC register by 20 ns (1/50 MHz).

The software must calculate the drift in frequency and update the EMAC_TM_ADDEND register accordingly.

NOTE: The PTP reference clock is the clock at which the system time is updated.When the EMAC_TM_CTL.TSCFUPDT bit is set to 0, this clock equals the PTP clock. Using fine correction, the PTP reference clock is generated on the nanoseconds increment signal at which the system time is updated.

Target Time Trigger (Alarm)

The PTP module provides an alarm function by triggering an alarm at a preset time. It sets the EMAC_TM_STMPSTAT.TSTARGT bit when the system time matches the value of the EMAC_TM_TGTM and EMAC_TM_NTGTM registers. This trigger can be used to generate an interrupt and/or command the flexible PPS module to start/stop PPS output, depending on value programmed in EMAC_TM_PPSCTL.TRGTMODSEL bits.

The trigger is enabled by setting EMAC_TM_CTL.TSTRIG bit. Once an alarm has occurred, if another alarm is needed, the software must clear the status bit, reprogram the EMAC_TM_TGTM and EMAC_TM_NTGTM registers to a future value, and set the EMAC_TM_CTL.TSTRIG bit.

If the time programmed in the target time registers has elapsed, then a target time programming error is indicated by setting the EMAC_TM_STMPSTAT.TSTRGTERR bit.

The alarm time is represented in absolute units, not relative units. For example, if the software needs to generate an alarm after 10 seconds, it must read the current system time value, add the number corresponding to 10 seconds, and write the result back to the target time registers.

Pulse-Per-Second (PPS)

Pulse-per-second (PPS) is a physical representation of system time. It is composed of a single pulse or train of pulses. PPS can be used for additional synchronization or to monitor the synchronization performance between clocks. With proper configuration, the PTP module can be programmed to generate PPS signals that are output on the ETH_PTPPPS pin. The PTP supports two kinds of PPS output, fixed and flexible.

Fixed Pulse-Per-Second Output

The EMAC supports fixed pulse-per-second (PPS) output that indicates 1 second intervals (default). The frequency of the PPS output can be changed by configuring the EMAC_TM_PPSCTL.PPSCTL bits. The default value for these bits is 0000, which configures a 1 Hz signal with a pulse width equal to the period of the PTP clock.

The following table shows various PPS output frequencies.

Table 23-40: PPS Output Frequencies

PPSCTL Bit Setting	Binary Rollover	Digital Rollover
0001	2 Hz	1 Hz
0010	4 Hz	2 Hz
0011	8 Hz	4 Hz
...	...	...
1111	32.768 kHz	16.384 kHz

In binary rollover mode, the PPS output has a duty cycle of 50% with these frequencies.

In digital rollover mode, the PPS output frequency is an average number. The actual clock is a different frequency that is synchronized every second. PPS output pulses have different periods and duty cycles and this behavior is because of the non-linear toggling of the bits in digital rollover mode. For example:

- When EMAC_TM_PPSCTL.PPSCTL = 0001, the PPS (1 Hz) has a low period of 537 ms and a high period of 463 ms.
- When EMAC_TM_PPSCTL.PPSCTL = 0010, the PPS (2 Hz) is a sequence of:
- One clock of 50 percent duty cycle and 537 ms period
- Second clock of 463 ms period (268 ms low and 195 ms high).
- When EMAC_TM_PPSCTL.PPSCTL = 0011, the PPS (4 Hz) is a sequence of:
- Three clocks of 50 percent duty cycle and 268 ms period
- Fourth clock of 195 ms period (134 ms low and 61 ms high)

Flexible Pulse-Per-Second Output

The EMAC also provides the flexibility to program the start or stop time, width, and interval of the pulse generated on the PPS output. This feature is called Flexible PPS and can be enabled by setting the EMAC_TM_PPSCTL.PPSEN bit.

The Flexible PPS output options are:

- Supports programming the start point of the single pulse and start and stop points of the pulse train in terms of system time. The target time registers are used to program the start and stop time.
- Supports programming the stop time in advance, that is, programs can configure the stop time before the actual start time has elapsed.
- Supports programming the width, between the rising edge and corresponding falling edge of the PPS signal output, in terms of number of units of sub-second increment. This value is configured in the EMAC_TM_SUBSEC register.
- Supports programming the interval, between the rising edges of PPS signal, in terms of number of units of sub-second increment. This value is configured in the EMAC_TM_SUBSEC register.
- Provides the option to cancel the programmed PPS start or stop request.
- Indicates error if the start or stop time being programmed has already elapsed.

PPS Start or Stop Time

Start time can initially be programmed in the target time registers. If required, the start or stop time can be programmed again but it can be done only after the earlier programmed value is synchronized to the PTP clock domain. The EMAC_TM_NTGTM.TSTRBUSY bit indicates the status of synchronization. This enables programs to configure the start or stop time in advance, even before the earlier stop or start time has elapsed.

The start or stop time should be programmed with advanced system time to ensure proper PPS signal output. If the application programs a start or stop time that has already elapsed, then the EMAC sets the EMAC_TM_STMPSTAT.TSTRGTERR bit, indicating the error. If enabled, the EMAC also sets the target time trigger (alarm) interrupt event. The application can cancel the start or stop request only if the corresponding start or stop time has not elapsed. If the time has elapsed, the cancel command has no effect.

PPS Width and Interval

The PPS width and interval are programmed in terms of granularity of system time, that is, the number of the units of sub-second increment value. For example, with the PTP reference clock of 50MHz: for a PPS pulse width of 40 ns and an interval of 100 ns, the width and interval should be programmed to values 2 and 5 respectively.

Smaller granularity can be achieved by using a faster PTP reference clock. Before giving the command to trigger a pulse or pulse train on the PPS output, programs should configure or update the interval and width of the PPS signal output.

PPS Command

When the PPS module is configured for flexible PPS output, the EMAC_TM_PPSCTL.PPSCTL bits can be used to command the PPS module for using any of the flexible PPS features.

Programming these bits with a non-zero value instructs the PPS module to initiate an event. Once the command is transferred or synchronized to the PTP clock domain, these bits are cleared automatically. Software should ensure that these bits are programmed only when they are all-zero.

The following table explains the different commands and their function.

Table 23-41: Flexible PPS Output Commands

PPSCTL (Bits 3–0)	Command	Description
0000	No Command	
0001	Start Single Pulse	Generates single pulse rising at start point defined in target time registers and of duration defined in EMAC_TM_PPSWIDTH register.
0010	Start Pulse train	Generates train of pulses rising at the start time configured in the Target Time registers, of duration configured in the EMAC_TM_PPSWIDTH register and repeated at interval configured in the EMAC_TM_PPSINTVL register. By default, the PPS pulse train is free-running unless stopped by stop pulse train at time or stop pulse train immediately commands.
0011	Cancel Start	Cancels the start single pulse and start pulse train commands if the system time has not crossed the programmed start time.
0100	Stop Pulse train at time	Stops the train of pulses initiated by the start pulse train command after the time programmed in the target time registers elapses.
0101	Stop Pulse train immediately	Immediately stops the train of pulses initiated by the Start Pulse train command.
0110	Cancel Stop Pulse train	Cancels the Stop Pulse train at time command if the programmed stop time has not elapsed. The PPS pulse train becomes free-running on the successful execution of this command.
0111-1111	Reserved	

PTP Interrupts

Interrupts from PTP module can be enabled by setting the EMAC_IMSK.TS bit. The status of the interrupt is indicated on the EMAC_ISTAT.TS bit. The PTP supports the following three types of interrupts.

Auxiliary Snapshot Trigger

This interrupt is triggered when an external event occurs on ETH_PTPAUXIn pin and timestamp snapshot occurs. This is indicated on EMAC_TM_STMPSTAT.ATSTS bit.

Target Time Reached

This interrupt is triggered when the system time becomes equal to the value written in the EMAC_TM_TGTMand EMAC_TM_NTGTM registers. It can be enabled or disabled by using the EMAC_TM_CTL.TSTRIG and

EMAC_TM_PPSCTL.TRGTMODSEL bits. This interrupt can be used as an alarm and is indicated on the EMAC_TM_STMPSTAT.TSTARGT bit.

System Time Seconds Register Overflow

This interrupt is triggered when the EMAC_TM_SEC register overflows from 0xFFFF FFFF to 0x0000 0000. This interrupt is indicated on the EMAC_TM_STMPSTAT.TSSOVF bit. As soon as EMAC_TM_SEC register overflows, the EMAC_TM_HISEC register increments by one.

EMAC Event Control

The EMAC has a dedicated interrupt signal registered with the processor System Event Controller (SEC). Various interrupt sources within EMAC peripheral are shared through this interrupt line. Please refer to the System Event Controller chapter for details on how interrupts work in this product and how to configure them.

EMAC Interrupt Signals

Interrupts from the EMAC can be triggered from the EMAC DMA layer or the EMAC CORE layer. Interrupts are triggered from EMAC DMA if a particular status bit is set in the EMAC_DMA_STAT register. An interrupt line is asserted only when the corresponding bits are enabled in the DMA interrupt enable register. Similarly, interrupts are triggered from the EMAC CORE if a particular MMC status bit or PTP status bit is set in the interrupt status register.

An interrupt line is asserted only when the corresponding bits are enabled in the MMC mask registers in case of MMC counters or the interrupt mask register in the case of PTP. Note that MMC interrupt status is also reflected in the DMA status register. The two groups of interrupts in the DMA status register are listed below.

NIS – Normal Interrupt source summary:

- Transmit Interrupt
- Transmit Buffer Unavailable
- Receive Interrupt
- Early Receive Interrupt

AIS – Abnormal Interrupt source summary:

- Transmit Process Stopped
- Transmit Jabber Timeout
- Receive FIFO Overflow
- Transmit Underflow

- Receive Buffer Unavailable
- Receive Process Stopped
- Receive Watchdog Timeout
- Early Transmit Interrupt
- Fatal Bus Error

An interrupt is generated only once for simultaneous, multiple events. The driver must read the EMAC_DMA_STAT register for the cause of the interrupt. A new interrupt can be generated once the driver has cleared the appropriate bit in DMA status register.

For example, the controller generates a receive interrupt (EMAC_DMA_STAT.RI bit) and the driver begins reading the EMAC_DMA_STAT register. Next, a receive buffer unavailable interrupt (EMAC_DMA_STAT.RU bit) occurs. The driver clears the EMAC_DMA_STAT.RI bit but the he internal interrupt signal is not de-asserted, because of the active or pending rEMAC_DMA_STAT.RU interrupt. Additionally, the driver must scan all of the descriptors, from the last recorded position to the first one owned by the DMA, in order to know which descriptor has asserted the interrupt.

Interrupts are cleared by writing a 1 to the corresponding bit position in the EMAC_DMA_STAT register. When all the enabled interrupts within a group are cleared, the corresponding summary bit is cleared.

An interrupt delay timer is provided (receive interrupt watchdog timer register) for flexible control of the receive interrupt.

When the interrupt timer is programmed with a non-zero value, it is activated as soon as the RxDMA completes a transfer of a received frame to system memory. This is done without asserting the receive interrupt because this interrupt is not enabled in the corresponding receive descriptor (RDES1[31] in the receive DMA descriptors).

When this timer runs out (per the programmed value), the EMAC_DMA_STAT.RI bit is set and the interrupt is asserted if the corresponding EMAC_DMA_STAT.RI bit is enabled in the interrupt enable register. This timer is disabled before it runs out, when a frame is transferred to memory and when the EMAC_DMA_STAT.RI bit is set because it is enabled for that descriptor.

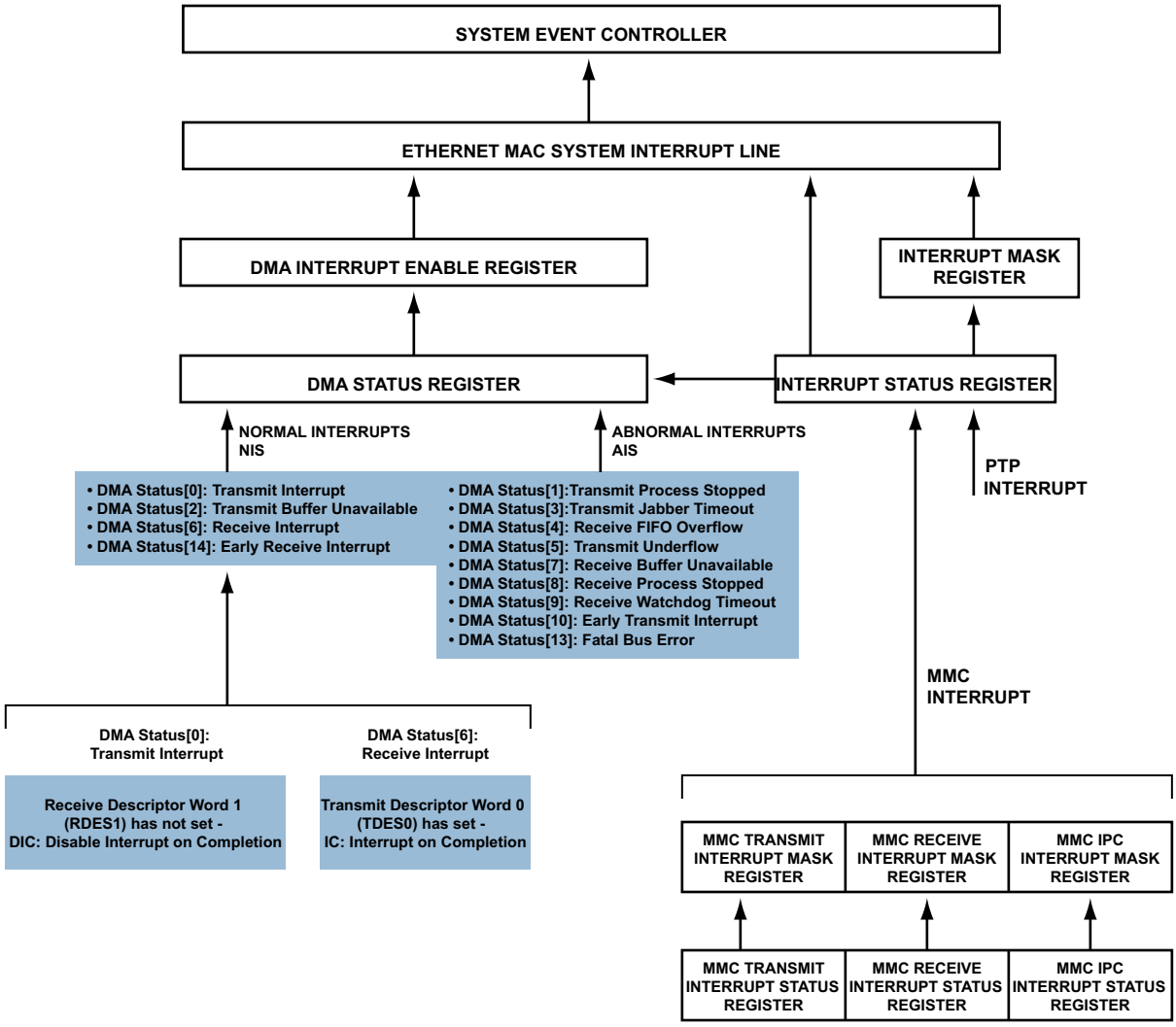


Figure 23-15: EMAC Interrupt Flow Diagram

PHYINT Interrupt Signal

A PHY device can notify the EMAC when it detects changes to the link status, such as auto-negotiation or a duplex-mode change. The external PHY chip typically includes an interrupt generation pin to aid this status change notification to the MAC. This signal is typically called PHYINT and a failing/rising edge on this signal can be used to detect a PHY interrupt at the EMAC.

In the ADSP-BF60x, any of the GPIO pin can be a used as a PHYINT signal. Use the following procedure to configure a GPIO as a PHYINT signal.

1. Program the GPIO to detect a falling/rising edge sensitive interrupt.
2. Program the PHY to generate the interrupt on a signal status change.
3. If PHYINT is asserted, read the PHY status register via the Station Management Interface.

NOTE: The PHYINT is not part of EMAC module, but rather any GPIO pin can be configured to interrupt the processor when a rising edge generated by PHY is detected.

Please refer to GPIO chapter for more info on configuring GPIO pins for input.

EMAC Programming Model

This section provides the programming model of Ethernet MAC peripheral for developers.

EMAC Programming Steps

The following sections provide some general programming information

DMA Initialization

Use the following procedure to initialize DMA.

1. Perform a software reset by setting the EMAC_DMA_BUSMODE.SWR bit. This resets all of the EMAC internal registers and logic.
2. Wait for the completion of the reset process by polling the EMAC_DMA_BUSMODE.SWR bit which is only cleared (automatically) after the reset operation is completed.
3. Poll the EMAC_DMA_BMSTAT.BUSRD and EMAC_DMA_BMSTAT.BUSWR bits to confirm that all previously initiated (before software-reset) or ongoing SCB transactions are complete.
4. Program the required fields in the EMAC_DMA_BMMODE register:
   a. Address aligned bursts.
   b. Fixed burst or undefined burst.
   c. Burst length values and burst mode values.
   d. Descriptor length (only valid if ring mode is used).
5. Program the SCB interface options in the EMAC_DMA_BMMODE register. If fixed burst-length is enabled, then select the maximum burst-length possible on the SCB bus (bits EMAC_DMA_BMMODE.BLEN4, EMAC_DMA_BMMODE.BLEN8, EMAC_DMA_BMMODE.BLEN16).
6. Create a proper descriptor chain for transmit and receive. In addition, ensure that the receive descriptors are owned by DMA (the OWN bit of the descriptor should be set). When OSF mode is used, at least two descriptors are required.
7. Ensure that the software creates three or more different transmit or receive descriptors in the chain before reusing any of the descriptors.

8. Initialize the EMAC_DMA_RXDSC_CUR and EMAC_DMA_TXDSC_CUR registers with the base address of the receive and transmit descriptors respectively.
9. Program the required fields in the EMAC_DMA_OPMODE register to initialize the mode of operation as follows:
    a. Receive and transmit store and forward.
    b. Receive and transmit threshold control.
    c. Error Frame and undersized good frame forwarding enable.
    d. OSF mode.
10. Clear the interrupt requests by writing to those bits of the EMAC_DMA_STAT register (interrupt bits only) that are set. For example, by writing 1 into bit 16, the normal interrupt summary clears this bit.
11. Enable the interrupts by programming the EMAC_DMA_IEN register.
12. Start the receive and transmit DMA by setting the EMAC_DMA_OPMODE.SR and EMAC_DMA_OPMODE.ST bits.

EMAC CORE Initialization

Use the following procedure to initialize the EMAC core.

1. Program the EMAC Management Address Register (EMAC_SMI_ADDR) for controlling the management cycles for external PHY. For example, physical layer address (EMAC_SMI_ADDR.PA). In addition, set the EMAC_SMI_ADDR.SMIB bit for writing into PHY and reading from PHY.
2. Read the 16-bit data of Management Data Register (EMAC_SMI_DATA) from the PHY for link up, speed of operation, and mode of operation, by specifying the appropriate address value in the EMAC_SMI_ADDR.PA bit field.
3. Program the MAC address in the EMAC_ADDR0_HI and EMAC_ADDR0_LO registers.
4. If hash filtering is used, program the hash table high and low registers register (EMAC_HASHTBL_HI, EMAC_HASHTBL_LO).
5. Program the required fields to set the appropriate filters for the incoming frames in the MAC frame filter register (EMAC_MACFRMFILT):
    a. Receive all.
    b. Promiscuous mode.
    c. Hash or perfect filter.
    d. Unicast, multicast, broadcast, and control frames filter settings.

6. Program the required fields for proper flow control in flow control register (EMAC_FLOWCTL):
   a. Pause time and other pause frame control bits.
   b. Receive and transmit flow control bits.
   c. Flow control busy/backpressure activate.
7. Program the EMAC interrupt mask register bits (EMAC_IMSK), as required.
8. Program the appropriate fields in MAC configuration register (EMAC_MACCFG). For example, inter-frame gap while transmission and jabber disable. Based on the Auto-negotiation desired, set the Duplex mode (EMAC_MACCFG.DM bit) or speed select (EMAC_MACCFG.FES bit).
9. Set the transmit enable (EMAC_MACCFG.TE) and receive enable (EMAC_MACCFG.RE) bits.

Performing Normal Transmit and Receive Operations

PREREQUISITE:

For normal transmit and receive interrupts, the program should first read the interrupt status.

1. Poll the descriptors, reading the status of the descriptor owned by the application (either transmit or receive).
2. Set the appropriate values for the descriptors, ensuring that transmit and receive descriptors are owned by the DMA to resume the transmission and reception of data.

   ADDITIONAL INFORMATION: If the descriptors are not owned by the DMA (or no descriptor is available), the DMA goes into SUSPEND state.
3. Write a 0 into the Tx/Rx poll demand registers (EMAC_DMA_TXPOLL and EMAC_DMA_RXPOLL).

   STEP RESULT: This resumes transmit or receive operations by freeing the descriptors and issuing a poll demand.
4. Read (for the debug process), the values of the current host transmitter or receiver descriptor address pointer (EMAC_DMA_TXDSC_CUR, EMAC_DMA_RXDSC_CUR) registers.
5. Read (for the debug process), the values of the current host transmit buffer address pointer and receive buffer address pointer (EMAC_DMA_TXBUF_CUR, EMAC_DMA_TXBUF_CUR) registers.

Stopping and Starting Transfers

Use the following procedure to stop and start EMAC transfers.

1. Disable the transmit DMA (if applicable), by clearing the EMAC_DMA_OPMODE.ST bit.
2. Wait for any previous frame transmissions to complete. Check this by reading the appropriate bits of the debug register (EMAC_DBG).

3. Disable the MAC transmitter and MAC receiver by clearing the EMAC_MACCFG.TE and EMAC_MACCFG.RE bits.
4. Disable the receive DMA (if applicable), after ensuring that the data in the receive FIFO is transferred to the system memory by reading the EMAC_DBG register.
5. Make sure that both the transmit and receive FIFOs are empty.
6. To re-start the operation, first start the DMA, and then enable the MAC transmitter and receiver.

Interrupts and Interrupt Service Routines

Specific steps for enabling interrupts and using their ISRs are described in the following procedure.

PREREQUISITE: This procedure is typically performed with EMAC and DMA initialization and operations.

1. Receive interrupts are enabled for descriptors by default. Transmit interrupts must be enabled for individual descriptors by setting the IC bit (bit 30) in the TDES0 word of the transmit descriptor.
2. Enable the required bits in the DMA interrupt enable register (EMAC_DMA_IEN).

   ADDITIONAL INFORMATION: Setting the EMAC_DMA_IEN.NIS or EMAC_DMA_IEN.AIS bits can turn on the occurrence of all normal/abnormal interrupt conditions. Individual conditions may also be enabled on using individual bits in the EMAC_DMA_IEN register.
3. Enable MMC overflow interrupts by setting appropriate bits in the EMAC_MMC_RXIMSK and EMAC_MMC_TXIMSK registers.
4. Enable PTP interrupts by setting the EMAC_IMSK.TS bit.
5. Once an EMAC interrupt is asserted and the SEC branches execution to the EMAC ISR, the following software program sequence is performed.
   a. Read DMA status from the EMAC_DMA_STAT register.
   b. Clear the interrupt source by writing 1 (W1C) to the bits that are set in the EMAC_DMA_STAT register.
   c. Check for normal/abnormal/mmc/ptp interrupts by parsing the status bits read earlier, and call the appropriate service function.

   ADDITIONAL INFORMATION: Typical normal interrupt assertions include Transmit and Receive Interrupt. Typical abnormal interrupt assertion include Receive Underflow.

6. The MMC handler functions use the following sequence.
   a. Read the EMAC_ISTAT register and parsing for the EMAC_ISTAT.MMCTX and EMAC_ISTAT.MMCRX bits to determine if the interrupt is a transmit counter or receive counter interrupt.
   b. Read the EMAC_MMC_RXINT or EMAC_MMC_TXINT registers to determine which of the counters have triggered the interrupt.
   c. Read the respective MMC counter that caused the interrupt to clear it.
7. PTP handler functions use the following sequence:
   a. Read the EMAC_ISTAT.TS bit to determine if a PTP Interrupt occurred.
   b. Read EMAC_TM_STMPSTAT register to determine the interrupt source by parsing the EMAC_TM_STMPSTAT.ATSTS, EMAC_TM_STMPSTAT.TSTARGT, and EMAC_TM_STMPSTAT.TSSOVF bits.
   c. Clear the interrupt source by reading the EMAC_TM_STMPSTAT register.

Enabling Checksum for Transmit and Receive

Use the following steps to enable transmit and receive checksums.

PREREQUISITE:

Enabling receive and transmit checksums is generally performed in conjunction with EMAC and DMA initialization and operations. Note that transmit and receive checksum features are independent of each other.

1. To enable transmit checksum insertion:
   a. Enable store forward mode in the FIFO by setting the EMAC_DMA_OPMODE.TSF bit.
   b. Ensure that the transmit frame can be contained within the 256 byte Tx FIFO conforming to the size rule: FIFO Depth – PBL – 3 FIFO locations, where PBL is burst length.
   c. Program the following required parameters for transmit checksum, by programming (CIC) checksum insertion control in TDES0: IP header checksum, IP header checksum and payload checksum, IP Header checksum, payload checksum and pseudo header checksum.

   STEP RESULT: A higher layer such as the IP stack sends out the packet to the EMAC which inserts the checksum as configured.

2. To enable receive checksum verification:
   a. Enable receive checksum off-load engine by setting the EMAC_MACCFG.IPC bit.
   b. Enable 8 word descriptor (32 bytes), by setting the EMAC_DMA_BUSMODE.ATDS bit.
   c. Provide a total of 8 x 32-bit word space for the receive descriptor.
   d. Wait for the receive interrupt and check for extended status availability by parsing bit 0 in the RDES0 word.
   e. If extended status available, read RDES4 and pass to a higher layer such as the IP stack.

   STEP RESULT: The higher software layer may check for IPv4/IPv6/payload type and checksum payload/header errors.

Programming the System Time Module

Use the following procedure to configure the PTP module

1. Enable PTP module by setting the EMAC_TM_CTL.TSENA bit 0.
2. System Time Initialization
   a. The time (seconds and nanoseconds) at which System Time should be initialized should be written into EMAC_TM_SECUPDT and EMAC_TM_NSECUPDT registers.
   b. Set EMAC_TM_CTL.TSINIT bit. System time is initialized and this bit clears automatically.
   c. Configure binary or digital rollover of the EMAC_TM_NSEC register using the EMAC_TM_CTL.TSCTRLSSR bit.
3. System Time Coarse Correction
   a. Write the offset time (seconds and nanoseconds) to be added to or subtracted from the system time using the EMAC_TM_SECUPDT and EMAC_TM_NSECUPDT registers.
   b. Choose between add or subtract offset time using the EMAC_TM_NSECUPDT.ADDSUB bit.
   c. Set the EMAC_TM_CTL.TSUPDT bit to correct system time with offset time. This bit clears automatically.
4. System Time Fine Correction
   a. Calculate the required addend value based on the input PTP clock frequency and the required frequency. See *Fine Correction Method*.
   b. Write the calculated addend value in EMAC_TM_ADDEND register and set the EMAC_TM_CTL.TSADDREG bit to update the addend value. This bit is cleared automatically.
   c. Configure the EMAC_TM_SUBSEC register based on new PTP frequency.

5. Target Time Trigger (Alarm)
   a. Set the EMAC_IMSK.TS bit to enable PTP interrupts.
   b. Program the EMAC_TM_PPSCTL.TRGTMODSEL bit with 00 or 10 (for PPS start/stop time programming).
   c. Program the time when interrupt should be triggered using the EMAC_TM_TGTM and EMAC_TM_NTGTM registers. The programmed time should be greater than the current system time.

   ADDITIONAL INFORMATION: If the programmed time is not greater than the target time, a programming error occurs and is indicated by the EMAC_TM_STMPSTAT.TSTRGTERR bit.

   d. Set the EMAC_TM_CTL.TSTRIG bit to enable the target time trigger interrupt.

   STEP RESULT: After the system time reaches the programmed target time (in step 2), the target time trigger interrupt occurs and is indicated by the EMAC_TM_STMPSTAT.TSTARGT and EMAC_ISTAT.TS bits. The EMAC_TM_CTL.TSTRIG bit is cleared automatically.

Programming The PTP for Frame Detection and Timestamping

Use the following procedure to configure the PTP module.

1. For timestamping a transmitting frame, set the TTSE bit in the TDES0 register of the corresponding frame.
2. Extend the descriptor word length from 4 words to 8 words by setting the EMAC_DMA_BUSMODE.ATDS bit.
3. Configure bi 18–10 in the EMAC_TM_CTL register so that the PTP module detects and/or timestamps only specific types of received frames. Refer to the EMAC_TM_CTL register description for more information.
4. Select the PTP clock source by programming the PADS_EMAC_PTP_CLKSEL register.
5. Enable the PTP module by setting the EMAC_TM_CTL.TSENA bit.
6. Initialize the system time.
7. Verify the RDES4 register for the status of the received frame and the RDES6 and RDES7 registers for timestamp nanoseconds and seconds value.

Programming for Auxiliary Timestamps

1. Set the EMAC_IMSK.TS bit to enable PTP interrupts.
2. Set the EMAC_TM_CTL.TSENA bit to enable the PTP module.
3. Initialize system time.

   ADDITIONAL INFORMATION: Whenever a rising edge on auxiliary timestamp trigger pin is detected, system time seconds and nanoseconds are captured and stored into 4-deep auxiliary timestamp FIFO. An

auxiliary timestamp trigger interrupt occurs and is indicated by the EMAC_TM_STMPSTAT.ATSTS and the EMAC_IMSK.TS bit.

4. The contents of the FIFO can be read one by one through EMAC_TM_AUXSTMP_SEC and EMAC_TM_AUXSTMP_NSEC registers. One level of the FIFO is cleared when the EMAC_TM_AUXSTMP_SEC register is read. Therefore read the EMAC_TM_AUXSTMP_NSEC register before the EMAC_TM_AUXSTMP_SEC register.
5. Set the EMAC_TM_CTL.ATSFC bit to clear the FIFO.

Programming Fixed Pulse-Per-Second Output

Use the following procedure to program PPS output fixed pulse-per-second output.

1. Enable the PTP module by setting the EMAC_TM_CTL.TSENA bit.
2. Configure the EMAC_TM_PPSCTL.PPSCTL bits and configure binary or digital rollover by configuring the EMAC_TM_CTL.TSCTRLSSR bit, so as to output the required PPS waveform. See *Fixed Pulse-Per-Second Output*.

Programming Flexible Pulse-Per-Second Output

Use the following procedure to program flexible PPS output.

1. Enable the PTP module by setting the EMAC_TM_CTL.TSENA bit.
2. Set the EMAC_TM_PPSCTL.PPSEN bit to enable flexible PPS output.
3. Program the EMAC_TM_PPSCTL.TRGTMODSEL bits with 11 or 10 (for target time trigger interrupt).
4. Program the start time value when the PPS output should start using the EMAC_TM_TGTM and EMAC_TM_NTGTM registers. Ensure that the EMAC_TM_NTGTM.TSTRBUSY bit is reset before programming the target time registers again.
5. Program the period of the PPS signal output using the EMAC_TM_PPSINTVL register for pulse train output, and the width of the PPS signal output in the EMAC_TM_PPSWIDTH register for single pulse or pulse train output.
6. Ensure that the EMAC_TM_PPSCTL.PPSCTL bits are cleared and then program the bits to 0001 to start single pulse, or to 0010 to start pulse train at programmed start time (Step 4).

   ADDITIONAL INFORMATION: The PPS pulse train is free-running unless stopped by a STOP pulse train at time command (EMAC_TM_PPSCTL.PPSCTL = 0100) or STOP pulse train immediately command EMAC_TM_PPSCTL.PPSCTL = 0101).
7. The start of pulse generation can be cancelled by giving the cancel start command (EMAC_TM_PPSCTL.PPSCTL = 0011) before the programmed start time (Step 4) elapses.

8. Program the stop time value when the PPS output should stop using the EMAC_TM_TGTM and EMAC_TM_NTGTM registers. Ensure that the EMAC_TM_NTGTM.TSTRBUSY bit is reset before programming the target time registers again.

9. Ensure that the EMAC_TM_PPSCTL.PPSCTL bits are cleared and then program them to 0100. This stops the train of pulses on PPS signal output after the programmed stop time (Step 8) elapses.

   ADDITIONAL INFORMATION: The pulse train can be stopped immediately by giving the STOP pulse train immediately command (EMAC_TM_PPSCTL.PPSCTL = 0101). Similarly, the stop pulse train command (given in Step 9) can be cancelled by programming the EMAC_TM_PPSCTL.PPSCTL bits to 0110 before the programmed stop time (Step 8) elapses.

EMAC Programming Concepts

The following sections provide basic information and guidelines to assist in programming the EMAC module.

IEEE 802.3 Ethernet Packet Structure

The typical frame format of an Ethernet packet is provided in the following table. Please refer to the IEEE standards for detailed information on Ethernet packets and their format.

Table 23-42: IEEE 802.3 Frame Structure

Parameter	Description	Position in Ethernet Packet	Total Bytes
PREAMBLE	This is a 56-bit (7-byte) pattern of alternating 1 and 0 bits (#10101010), which allows devices on the network to detect a new incoming frame for synchronization.	1	7
SFD	The SFD (#10101011) is a 1-byte pattern designed to break the preamble pattern, and signal the start of the actual frame.	2	1
DA	48-bit destination address. This can be a unicast, multicast or broadcast address.	3	6
SA	48-bit long source address, typically a unicast, multicast or broadcast address.	4	6
LT	Typically this field is the length, in terms of the number of bytes, and can be anywhere between 0 – 1500. When the value is greater than or equal to 0x0600, this field is also used to indicate the type of special payload carried by the frame. Examples include 0x8808 for flow control and 0x0800 for IPv4.	5	2

Table 23-42: IEEE 802.3 Frame Structure (Continued)

Parameter	Description	Position in Ethernet Packet	Total Bytes
DATA	Actual application data payload, usually between 0 – 1500.	6	0–1500
PAD	This field compensates for data frames that are shorter than 64 bytes long, not including the preamble.	7	0–46
FCS	The frame check sequence is a 32-bit cyclic redundancy check that detects corrupted data within the entire frame. This is generated from a CRC-32 polynomial code (CRC-32-IEEE): G (x) = x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1.	8	4

Frame Size Statistics for Application Software

Table 23-43: Ethernet Frame Size Statistics

Frame size statistics	VLAN specific change Comments	
Information bytes/ Header	4 byte 802.1Q header inserted after Source Address and before Type/LAN in 802.3 packets = 22 bytes.	6 x 2 + 2 + 4 = 18 bytes (DA+SA+LT+FCS)
Minimum Frame Size (typical)	If DATA is NULL, 42 byte padding is done to make 64 bytes (42 +22)	64 bytes. If DATA is NULL, 46 byte padding is done to make 64 bytes (46 +18)
Maximum Frame Size (typical)	1522 bytes	1518 bytes (1500 bytes DATA and 18 bytes header)
Jumbo Frame Size	9022 bytes	Typical industry standard Ethernet jumbo frame size may be treated as 9018 bytes.

Software Visualization of Programmable Packet Size

The following table provides the byte sizes of packets with various configurations.

Table 23-44: Visualization of Programmable Packet Size

Size in Bytes	Comments
16384	Receive watchdog and transmit jabber disabled, jumbo frames enabled.
10240	Receive watchdog and transmit jabber disabled, jumbo frames disabled.
2048	Receive Watchdog and Transmit Jabber enabled.

Table 23-44: Visualization of Programmable Packet Size (Continued)

Size in Bytes	Comments
1518	Typical max size of Ethernet frame. Receive watchdog and transmit jabber enabled.
64	Typical minimum size of Ethernet frame.
< 64	Runt frames requiring Zero-PAD.

Ethernet Packet Structure in C

The following is an example for Ethernet packet structure in the C language.

```
typedef struct ETHER_PACKET
{
    char  dst_addr[6];          //destination address
    char  src_addr[6];          //source address
    char  length[2];            //length of actual data
    char  data[DATA_SIZE];      //application data
    char  fdlimit[DELIMIT_SIZE];//32-bit delimit (if manual appending)
    char  fcs[4];               //crc frame checksum, used by RX buffer.
} ETHER_PACKET;
```

DMA Descriptor Implementation in C

The following code is a simple implementation of descriptors in ring and chain model in C language. Typically 4 WORDs (32-bit) are used for descriptors. Using checksum off load or the PTP engine requires 8 WORDs. Only high-level common functions across transmit and receive descriptors are considered here.

```
/* DMA Ring Descriptor */
typedef struct EMAC_DMADESC_RING
{
    unsigned int                    Status; //TDES0 OR RDES0
    unsigned int                    ControlDesc; //TDES1 OR RDES1
    unsigned int                    StartAddr1;  //TDES2 OR RDES2
    unsigned int                    StartAddr2;  //TDES3 OR RDES3
    #ifdef CHECKSUM_OFFLOAD
    struct EMAC_EXT_STAT  ExtendedStat;
    #endif
} EMAC_DMADESC_RING;
/* DMA Chain Descriptor */
typedef struct EMAC_DMADESC_CHAIN
{
    unsigned int                    Status; //TDES0 OR RDES0
    unsigned int                    ControlDesc; //TDES1 OR RDES1
    unsigned int                    StartAddr; //TDES2 OR RDES2
    struct EMAC_DMADESC_CHAIN *pNextDesc;  //TDES3 OR RDES3
    #ifdef CHECKSUM_OFFLOAD
    struct EMAC_EXT_STAT ExtendedStat;
```

```
    #endif
} EMAC_DMADESC_CHAIN;
/* Extended Status Descriptor with PTP not enabled*/
typedef struct EMAC_EXT_STAT
{
    #ifdef RX_DESC
    unsigned int                    CheckSumStat;//RDES4
    #ifdef TX_DESC
    unsigned int                    Reserved; //TDES4
    #endif
    unsigned int                    Reserved;    //RDES5 OR TDES5
    unsigned int                    Reserved;  //RDES6 OR TDES6
    unsigned int                    Reserved; //RDES7 OR TDES7
} EMAC_EXT_STAT;
```

PTP Header Structure in C

The following code is an example of the PTM message format.

```
/* PTP Message Format (Refer to PTP Frame Over IPv4)*/
    typedef struct EMAC_PTP_HEADER
    {
    unsigned charmessageType:4,    //PTP Version 2 message type
    transportSpecific:4;
    unsigned charversionPTP;       //PTP Version (1 or 2)
    unsigned shortmessageLength;
    unsigned chardomainNumber;
    unsigned charRESERVED1;
    unsigned shortflagField;
    unsigned charcorrectionField[8];
    unsigned charRESERVED2[4];
    unsigned charsourcePortIdentity[10];
    unsigned shortsequenceid;
    unsigned charcontrolField;//PTP Version 1 message type
    unsigned charlogMessageInterval;
    }EMAC_PTP_HEADER;
```

ADSP-BF60x EMAC Register Descriptions

Ethernet MAC (EMAC) contains the following registers.

Table 23-45: ADSP-BF60x EMAC Register List

Name	Description
EMAC_MACCFG	MAC Configuration Register
EMAC_MACFRMFILT	MAC Rx Frame Filter Register

Table 23-45: ADSP-BF60x EMAC Register List (Continued)

Name	Description
EMAC_HASHTBL_HI	Hash Table High Register
EMAC_HASHTBL_LO	Hash Table Low Register
EMAC_SMI_ADDR	SMI Address Register
EMAC_SMI_DATA	SMI Data Register
EMAC_FLOWCTL	FLow Control Register
EMAC_VLANTAG	VLAN Tag Register
EMAC_DBG	Debug Register
EMAC_ISTAT	Interrupt Status Register
EMAC_IMSK	Interrupt Mask Register
EMAC_ADDR0_HI	MAC Address 0 High Register
EMAC_ADDR0_LO	MAC Address 0 Low Register
EMAC_MMC_CTL	MMC Control Register
EMAC_MMC_RXINT	MMC Rx Interrupt Register
EMAC_MMC_TXINT	MMC Tx Interrupt Register
EMAC_MMC_RXIMSK	MMC Rx Interrupt Mask Register
EMAC_MMC_TXIMSK	MMC TX Interrupt Mask Register
EMAC_TXOCTCNT_GB	Tx OCT Count (Good/Bad) Register
EMAC_TXFRMCNT_GB	Tx Frame Count (Good/Bad) Register
EMAC_TXBCASTFRM_G	Tx Broadcast Frames (Good) Register
EMAC_TXMCASTFRM_G	Tx Multicast Frames (Good) Register
EMAC_TX64_GB	Tx 64-Byte Frames (Good/Bad) Register
EMAC_TX65TO127_GB	Tx 65- to 127-Byte Frames (Good/Bad) Register

Table 23-45: ADSP-BF60x EMAC Register List (Continued)

Name	Description
EMAC_TX128TO255_GB	Tx 128- to 255-Byte Frames (Good/Bad) Register
EMAC_TX256TO511_GB	Tx 256- to 511-Byte Frames (Good/Bad) Register
EMAC_TX512TO1023_GB	Tx 512- to 1023-Byte Frames (Good/Bad) Register
EMAC_TX1024TOMAX_GB	Tx 1024- to Max-Byte Frames (Good/Bad) Register
EMAC_TXUCASTFRM_GB	Tx Unicast Frames (Good/Bad) Register
EMAC_TXMCASTFRM_GB	Tx Multicast Frames (Good/Bad) Register
EMAC_TXBCASTFRM_GB	Tx Broadcast Frames (Good/Bad) Register
EMAC_TXUNDR_ERR	Tx Underflow Error Register
EMAC_TXSNGCOL_G	Tx Single Collision (Good) Register
EMAC_TXMULTCOL_G	Tx Multiple Collision (Good) Register
EMAC_TXDEFERRED	Tx Deferred Register
EMAC_TXLATECOL	Tx Late Collision Register
EMAC_TXEXCESSCOL	Tx Excess Collision Register
EMAC_TXCARR_ERR	Tx Carrier Error Register
EMAC_TXOCTCNT_G	Tx Octet Count (Good) Register
EMAC_TXFRMCNT_G	Tx Frame Count (Good) Register
EMAC_TXEXCESSDEF	Tx Excess Deferral Register
EMAC_TXPAUSEFRM	Tx Pause Frame Register
EMAC_TXVLANFRM_G	Tx VLAN Frames (Good) Register
EMAC_RXFRMCNT_GB	Rx Frame Count (Good/Bad) Register
EMAC_RXOCTCNT_GB	Rx Octet Count (Good/Bad) Register
EMAC_RXOCTCNT_G	Rx Octet Count (Good) Register

Table 23-45: ADSP-BF60x EMAC Register List (Continued)

Name	Description
EMAC_RXBCASTFRM_G	Rx Broadcast Frames (Good) Register
EMAC_RXMCASTFRM_G	Rx Multicast Frames (Good) Register
EMAC_RXCRC_ERR	Rx CRC Error Register
EMAC_RXALIGN_ERR	Rx alignment Error Register
EMAC_RXRUNT_ERR	Rx Runt Error Register
EMAC_RXJAB_ERR	Rx Jab Error Register
EMAC_RXUSIZE_G	Rx Undersize (Good) Register
EMAC_RXOSIZE_G	Rx Oversize (Good) Register
EMAC_RX64_GB	Rx 64-Byte Frames (Good/Bad) Register
EMAC_RX65TO127_GB	Rx 65- to 127-Byte Frames (Good/Bad) Register
EMAC_RX128TO255_GB	Rx 128- to 255-Byte Frames (Good/Bad) Register
EMAC_RX256TO511_GB	Rx 256- to 511-Byte Frames (Good/Bad) Register
EMAC_RX512TO1023_GB	Rx 512- to 1023-Byte Frames (Good/Bad) Register
EMAC_RX1024TOMAX_GB	Rx 1024- to Max-Byte Frames (Good/Bad) Register
EMAC_RXUCASTFRM_G	Rx Unicast Frames (Good) Register
EMAC_RXLEN_ERR	Rx Length Error Register
EMAC_RXOORTYPE	Rx Out Of Range Type Register
EMAC_RXPAUSEFRM	Rx Pause Frames Register
EMAC_RXFIFO_OVF	Rx FIFO Overflow Register
EMAC_RXVLANFRM_GB	Rx VLAN Frames (Good/Bad) Register
EMAC_RXWDOG_ERR	Rx Watch Dog Error Register
EMAC_IPC_RXIMSK	MMC IPC Rx Interrupt Mask Register

Table 23-45: ADSP-BF60x EMAC Register List (Continued)

Name	Description
EMAC_IPC_RXINT	MMC IPC Rx Interrupt Register
EMAC_RXIPV4_GD_FRM	Rx IPv4 Datagrams (Good) Register
EMAC_RXIPV4_HDR_ERR_FRM	Rx IPv4 Datagrams Header Errors Register
EMAC_RXIPV4_NOPAY_FRM	Rx IPv4 Datagrams No Payload Frame Register
EMAC_RXIPV4_FRAG_FRM	Rx IPv4 Datagrams Fragmented Frames Register
EMAC_RXIPV4_UDSBL_FRM	Rx IPv4 UDP Disabled Frames Register
EMAC_RXIPV6_GD_FRM	Rx IPv6 Datagrams Good Frames Register
EMAC_RXIPV6_HDR_ERR_FRM	Rx IPv6 Datagrams Header Error Frames Register
EMAC_RXIPV6_NOPAY_FRM	Rx IPv6 Datagrams No Payload Frames Register
EMAC_RXUDP_GD_FRM	Rx UDP Good Frames Register
EMAC_RXUDP_ERR_FRM	Rx UDP Error Frames Register
EMAC_RXTCP_GD_FRM	Rx TCP Good Frames Register
EMAC_RXTCP_ERR_FRM	Rx TCP Error Frames Register
EMAC_RXICMP_GD_FRM	Rx ICMP Good Frames Register
EMAC_RXICMP_ERR_FRM	Rx ICMP Error Frames Register
EMAC_RXIPV4_GD_OCT	Rx IPv4 Datagrams Good Octets Register
EMAC_RXIPV4_HDR_ERR_OCT	Rx IPv4 Datagrams Header Errors Register
EMAC_RXIPV4_NOPAY_OCT	Rx IPv4 Datagrams No Payload Octets Register
EMAC_RXIPV4_FRAG_OCT	Rx IPv4 Datagrams Fragmented Octets Register
EMAC_RXIPV4_UDSBL_OCT	Rx IPv4 UDP Disabled Octets Register
EMAC_RXIPV6_GD_OCT	Rx IPv6 Good Octets Register
EMAC_RXIPV6_HDR_ERR_OCT	Rx IPv6 Header Errors Register

Table 23-45: ADSP-BF60x EMAC Register List (Continued)

Name	Description
EMAC_RXIPV6_NOPAY_OCT	Rx IPv6 No Payload Octets Register
EMAC_RXUDP_GD_OCT	Rx UDP Good Octets Register
EMAC_RXUDP_ERR_OCT	Rx UDP Error Octets Register
EMAC_RXTCP_GD_OCT	Rx TCP Good Octets Register
EMAC_RXTCP_ERR_OCT	Rx TCP Error Octets Register
EMAC_RXICMP_GD_OCT	Rx ICMP Good Octets Register
EMAC_RXICMP_ERR_OCT	Rx ICMP Error Octets Register
EMAC_TM_CTL	Time Stamp Control Register
EMAC_TM_SUBSEC	Time Stamp Sub Second Increment Register
EMAC_TM_SEC	Time Stamp Low Seconds Register
EMAC_TM_NSEC	Time Stamp Nanoseconds Register
EMAC_TM_SECUPDT	Time Stamp Seconds Update Register
EMAC_TM_NSECUPDT	Time Stamp Nanoseconds Update Register
EMAC_TM_ADDEND	Time Stamp Addend Register
EMAC_TM_TGTM	Time Stamp Target Time Seconds Register
EMAC_TM_NTGTM	Time Stamp Target Time Nanoseconds Register
EMAC_TM_HISEC	Time Stamp High Second Register
EMAC_TM_STMPSTAT	Time Stamp Status Register
EMAC_TM_PPSCTL	PPS Control Register
EMAC_TM_AUXSTMP_NSEC	Time Stamp Auxiliary TS Nano Seconds Register
EMAC_TM_AUXSTMP_SEC	Time Stamp Auxiliary TM Seconds Register
EMAC_TM_PPSINTVL	Time Stamp PPS Interval Register

Table 23-45: ADSP-BF60x EMAC Register List (Continued)

Name	Description
EMAC_TM_PPSWIDTH	PPS Width Register
EMAC_DMA_BUSMODE	DMA Bus Mode Register
EMAC_DMA_TXPOLL	DMA Tx Poll Demand Register
EMAC_DMA_RXPOLL	DMA Rx Poll Demand register
EMAC_DMA_RXDSC_ADDR	DMA Rx Descriptor List Address Register
EMAC_DMA_TXDSC_ADDR	DMA Tx Descriptor List Address Register
EMAC_DMA_STAT	DMA Status Register
EMAC_DMA_OPMODE	DMA Operation Mode Register
EMAC_DMA_IEN	DMA Interrupt Enable Register
EMAC_DMA_MISS_FRM	DMA Missed Frame Register
EMAC_DMA_RXIWDOG	DMA Rx Interrupt Watch Dog Register
EMAC_DMA_BMMODE	DMA SCB Bus Mode Register
EMAC_DMA_BMSTAT	DMA SCB Status Register
EMAC_DMA_TXDSC_CUR	DMA Tx Descriptor Current Register
EMAC_DMA_RXDSC_CUR	DMA Rx Descriptor Current Register
EMAC_DMA_TXBUF_CUR	DMA Tx Buffer Current Register
EMAC_DMA_RXBUF_CUR	DMA Rx Buffer Current Register

MAC Configuration Register

The EMAC_MACCFG register configures MAC features.

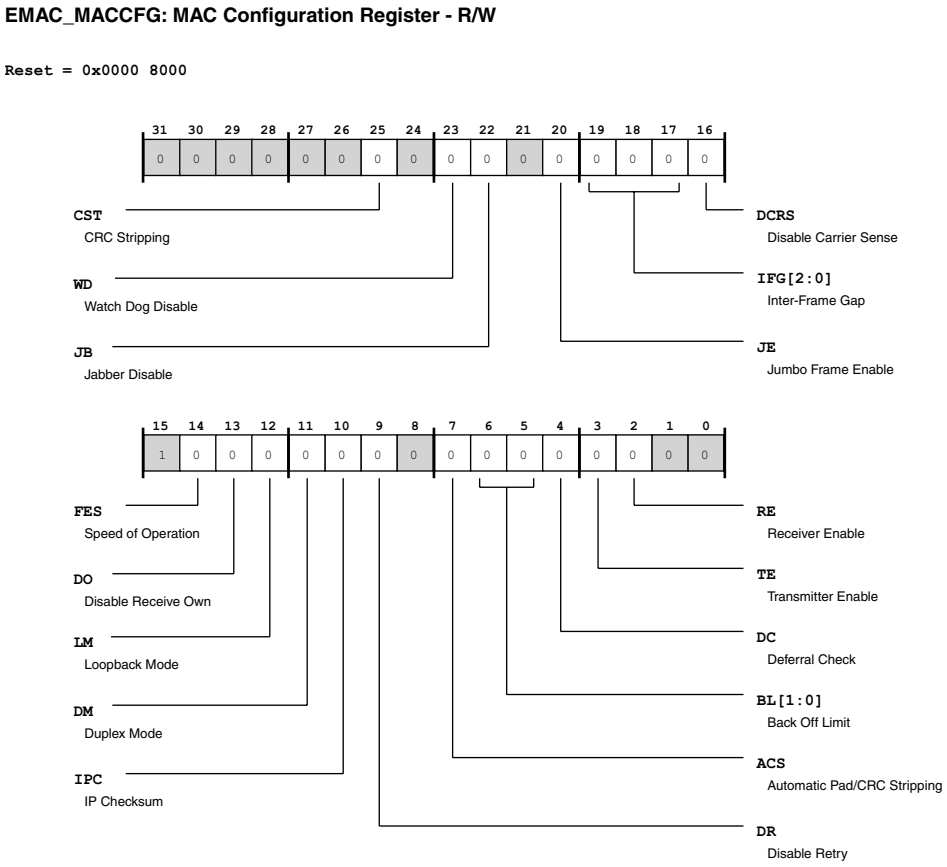


Figure 23-16: EMAC_MACCFG Register Diagram

Table 23-46: EMAC_MACCFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/W)	CST	CRC Stripping. The EMAC_MACCFG.CST bit, when set, directs the MAC to strip the last 4 bytes (FCS) of all frames of Ether type (Type field of frame greater than 0x0600) and drop these bytes before forwarding the frame to the application.
23 (R/W)	WD	Watch Dog Disable. The EMAC_MACCFG.WD bit, when set, disables the watchdog timer on the receiver, and can receive frames of up to 16,384 bytes. When this bit is reset, the MAC allows no more than 2,048 bytes (10,240 if EMAC_MACCFG.JE is set high) of the frame being received and cuts off any bytes received after that.

Table 23-46: EMAC_MACCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/W)	JB	Jabber Disable. The EMAC_MACCFG.JB bit, when set, disables the jabber timer on the transmitter, and can transfer frames of up to 16,384 bytes. When this bit is reset, the MAC cuts off the transmitter if the application sends out more than 2,048 bytes of data (10,240 if EMAC_MACCFG.JE is set high) during transmission.	
20 (R/W)	JE	Jumbo Frame Enable. The EMAC_MACCFG.JE bit, when set, directs the MAC to allow Jumbo frames of 9,018 bytes (9,022 bytes for VLAN tagged frames).	
19:17 (R/W)	IFG	Inter-Frame Gap. The EMAC_MACCFG.IFG bits control the minimum inter-frame gap between frames during transmission. Note that in Half-Duplex mode, the minimum gap can be configured for 64 bit times (EMAC_MACCFG.IFG =100) only. Lower values are not considered.	
		0	96 bit times
		1	88 bit times
		2	80 bit times
		3	72 bit times
		4	64 bit times
		5	56 bit times
		6	48 bit times
		7	40 bit times
16 (R/W)	DCRS	Disable Carrier Sense. The EMAC_MACCFG.DCRS bit, when set, makes the MAC transmitter ignore the CRS signal during frame transmission in Half-Duplex mode. This request results in no errors generated due to Loss of Carrier or No Carrier during such transmission. When this bit is low, the MAC transmitter generates such errors due to Carrier Sense and will even abort the transmissions.	
14 (R/W)	FES	Speed of Operation. The EMAC_MACCFG.FES bit indicates the Ethernet speed as 10 Mbps (bit =0) or 100 Mbps (bit =1).	

Table 23-46: EMAC_MACCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	DO	Disable Receive Own. The EMAC_MACCFG.DO bit, when set, disables MAC reception of frames when MAC is transmitting in Half-Duplex mode. When this bit is reset, the MAC receives all packets that are given by the PHY while transmitting. This bit is not applicable if the MAC is operating in Full-Duplex mode.	
12 (R/W)	LM	Loopback Mode. The EMAC_MACCFG.LM bit, when set, directs the MAC to operate in internal loop back mode. (The media independent interface pins are not driven or sampled.)	
11 (R/W)	DM	Duplex Mode. The EMAC_MACCFG.DM bit, when set, directs the MAC to operate in a Full-Duplex mode where it can transmit and receive simultaneously.	
10 (R/W)	IPC	IP Checksum. The EMAC_MACCFG.IPC bit, when set, directs the MAC to calculate the 16-bit one's complement of the one's complement sum of all received Ethernet frame payloads. It also checks whether the IPv4 Header checksum (assumed to be bytes 25-26 or 29-30 (VLAN-tagged) of the received Ethernet frame) is correct for the received frame and gives the status in the receive status word. The EMAC_MACCFG.IPC bit, when set, enables IPv4 checksum checking for received frame payloads' TCP/UDP/ICMP headers. When this bit is reset, the Checksum Offload Engine function in the receiver is disabled and the corresponding PCE and IP HCE status bits are always cleared.	
9 (R/W)	DR	Disable Retry. The EMAC_MACCFG.DR bit, when set, directs the MAC to attempt only 1 transmission. When a collision occurs on the media independent interface, the MAC ignores the current frame transmission and reports a Frame Abort with excessive collision error in the transmit frame status. When the EMAC_MACCFG.DR bit is reset, the MAC attempts retries based on the settings of BL. This bit is applicable only to Half-Duplex mode.	
		0	Retry enabled
		1	Retry disabled

Table 23-46: EMAC_MACCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	ACS	Automatic Pad/CRC Stripping. The EMAC_MACCFG.ACS bit, when set, directs the MAC to strip the Pad/FCS field on incoming frames only if the length fields value is less than or equal to 1,500 bytes. All received frames with length field greater than or equal to 1,501 bytes are passed to the application without stripping the Pad/FCS field. When the EMAC_MACCFG.ACS bit is reset, the MAC passes all incoming frames to the Host unmodified.	
6:5 (R/W)	BL	Back Off Limit. The EMAC_MACCFG.BL bit selects the back-off limit, determining the random integer number (r) of slot time delays (4,096 bit times for 1000 Mbps and 512 bit times for 10/100 Mbps) the MAC waits before rescheduling a transmission attempt during retries after a collision. This bit is applicable only to Half-Duplex mode.The random integer r takes the value in the range: 0 less-than-equal-to r less-than 2^k Where k is the minimum of n (number of transmission attempts) or a limit value.	
		0	k = min (n, 10)
		1	k = min (n, 8)
		2	k = min (n, 4)
		3	k = min (n, 1)
4 (R/W)	DC	Deferral Check. The EMAC_MACCFG.DC bit, when set, enables the deferral check function in the MAC. The MAC issues a Frame Abort status, along with the excessive deferral error bit set in the transmit frame status when the transmit state machine is deferred for more than 24,288 bit times in 10/100-Mbps mode. If the Jumbo frame mode is enabled in 10/100-Mbps mode, the threshold for deferral is 155,680 bits times. Deferral begins when the transmitter is ready to transmit, but is prevented because of an active CRS (carrier sense) signal. Defer time is not cumulative. If the transmitter defers for 10,000 bit times, then transmits, collides, backs off, and then has to defer again after completion of back-off, the deferral timer resets to 0 and restarts. When the EMAC_MACCFG.DC bit is reset, the deferral check function is disabled and the MAC defers until the CRS signal goes inactive. This bit is applicable only in Half-Duplex mode.	

Table 23-46: EMAC_MACCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	TE	Transmitter Enable. The EMAC_MACCFG.TE bit, when set, enables the transmit state machine of the MAC for transmission. When this bit is reset, the MAC transmit state machine is disabled after the completion of the transmission of the current frame, and will not transmit any further frames.
2 (R/W)	RE	Receiver Enable. The EMAC_MACCFG.RE bit, when set, enables the receiver state machine of the MAC for receiving frames. When this bit is reset, the MAC receive state machine is disabled after the completion of the reception of the current frame, and does not receive any further frames..

MAC Rx Frame Filter Register

The EMAC_MACFRMFILT register controls receive frame filter features.

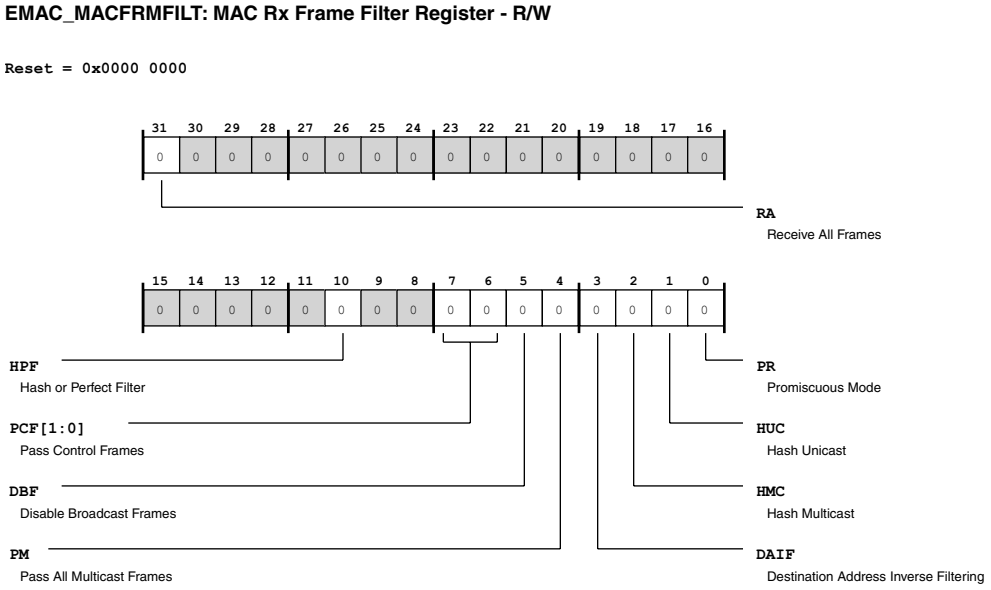


Figure 23-17: EMAC_MACFRMFILT Register Diagram

Table 23-47: EMAC_MACFRMFILT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	RA	Receive All Frames. The EMAC_MACFRMFILT.RA bit, when set, directs the MAC Receiver module to pass to the Application all frames received irrespective of whether they pass the address filter. The result of the DA filtering is updated (pass or fail) in the corresponding bits in the Receive Status Word. When this bit is reset, the Receiver module passes to the Application only those frames that pass the DA address filter.	
10 (R/W)	HPF	Hash or Perfect Filter. The EMAC_MACFRMFILT.HPF bit. when set, configures the address filter to pass a frame if it matches either the perfect filtering or the hash filtering as set by EMAC_MACFRMFILT.HMC or EMAC_MACFRMFILT.HUC bits. When EMAC_MACFRMFILT.HPF is low and either the EMAC_MACFRMFILT.HUC bit or EMAC_MACFRMFILT.HMC bit is set, the frame is passed only if it matches the Hash filter.	
7:6 (R/W)	PCF	Pass Control Frames. The EMAC_MACFRMFILT.PCF bits control the forwarding of all control frames (including unicast and multicast PAUSE frames). Note that the processing of PAUSE control frames depends only on the value of the EMAC_FLOWCTL.RFE bit.	
		0	Pass no control frames All control frames are filtered from reaching the application.
		1	Pass no PAUSE frames All control frames are passed to the application (even if the fail the address filter), except for PAUSE frames.
		2	Pass all control frames All control frames are passed to the application (even if the fail the address filter).
		3	Pass address filtered control frames All control frames that pass the address filter are passed to the application.

Table 23-47: EMAC_MACFRMFILT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	DBF	Disable Broadcast Frames. The EMAC_MACFRMFILT.DBF bit, when set, directs the AFM module to filter all incoming broadcast frames. When this bit is reset, the AFM module passes all received broadcast frames.	
		0	AFM module passes all received broadcast frames
		1	AFM module filters all incoming broadcast frames
4 (R/W)	PM	Pass All Multicast Frames. The EMAC_MACFRMFILT.PM bit, when set, indicates that all received frames with a multicast destination address (first bit in the destination address field is =1) are passed. When this bit is reset, filtering of multicast frame depends on EMAC_MACFRMFILT.HMC bit.	
3 (R/W)	DAIF	Destination Address Inverse Filtering. The EMAC_MACFRMFILT.DAIF bit, when set, directs the Address Check block to operate in inverse filtering mode for the DA address comparison for both unicast and multicast frames. When this bit is reset, normal filtering of frames is performed.	
2 (R/W)	HMC	Hash Multicast. The EMAC_MACFRMFILT.HMC bit, when set, directs the EMAC to perform destination address filtering of received multicast frames according to the hash table. When this bit is reset, the MAC performs a perfect destination address filtering for multicast frames, that is, the MAC compares the DA field with the values programmed in the EMAC_ADDR0_HI and EMAC_ADDR0_LO address registers.	
1 (R/W)	HUC	Hash Unicast. The EMAC_MACFRMFILT.HUC bit, when set, directs the EMAC to perform destination address filtering of unicast frames according to the hash table. When this bit is reset, the MAC performs a perfect destination address filtering for unicast frames, that is, it compares the DA field with the values programmed in the EMAC_ADDR0_HI and EMAC_ADDR0_LO address registers.	
0 (R/W)	PR	Promiscuous Mode. The EMAC_MACFRMFILT.PR bit, when set, directs the Address Filter module to pass all incoming frames regardless of its destination or source address. The DA Filter Fails status bits of the Receive Status Word is always cleared when EMAC_MACFRMFILT.PR is set.	

Hash Table High Register

The EMAC_HASHTBL_HI register contains the upper 32 bits of the hash table.

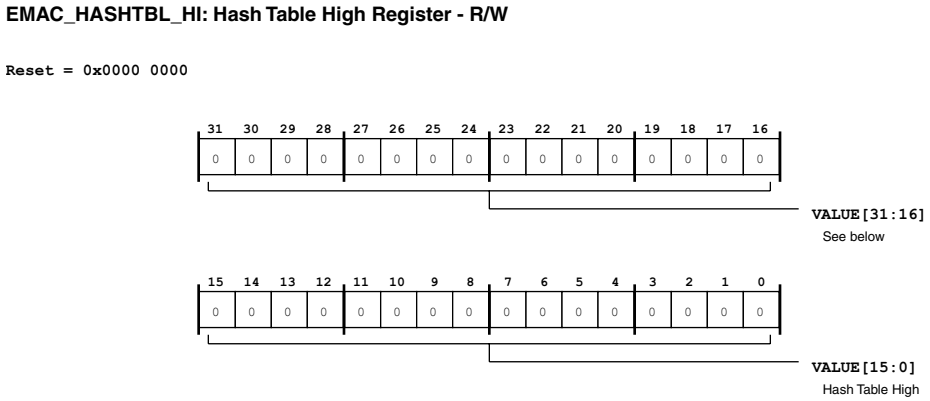


Figure 23-18: EMAC_HASHTBL_HI Register Diagram

Table 23-48: EMAC_HASHTBL_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Hash Table High. The EMAC_HASHTBL_HI.VALUE bits contain the upper 32 bits of Hash table.

Hash Table Low Register

The EMAC_HASHTBL_LO register contains the lower 32 bits of the hash table.

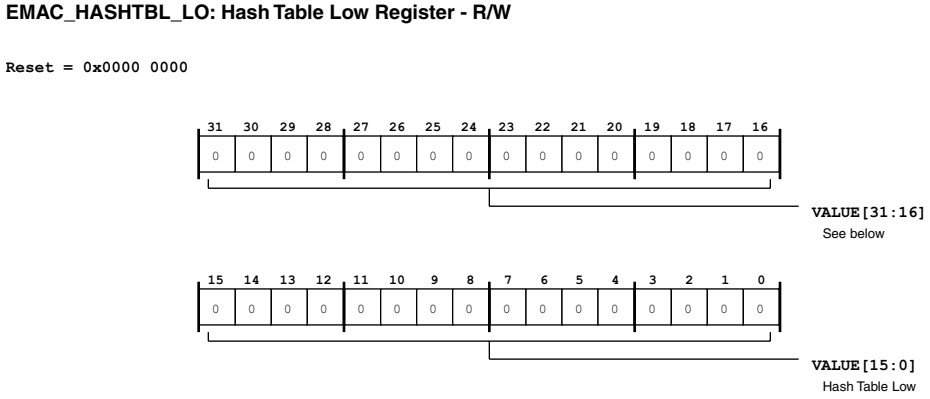


Figure 23-19: EMAC_HASHTBL_LO Register Diagram

Table 23-49: EMAC_HASHTBL_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Hash Table Low. The EMAC_HASHTBL_LO.VALUE bits contain the lower 32 bits of Hash table.

SMI Address Register

The EMAC_SMI_ADDR register contains the station management interface address and feature settings.

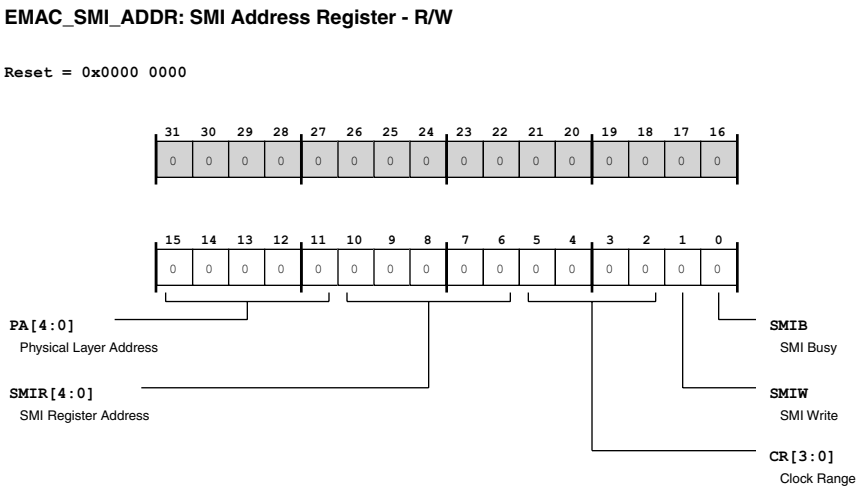


Figure 23-20: EMAC_SMI_ADDR Register Diagram

Table 23-50: EMAC_SMI_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:11 (R/W)	PA	Physical Layer Address. The EMAC_SMI_ADDR.PA bits select the PHY. This field tells which of the 32 possible PHY devices are being accessed.
10:6 (R/W)	SMIR	SMI Register Address. The EMAC_SMI_ADDR.SMIR bits select the desired Station Management Interface register in the selected PHY device.

Table 23-50: EMAC_SMI_ADDR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:2 (R/W)	CR	Clock Range. The EMAC_SMI_ADDR.CR bits select the Clock Range, determining the frequency of the MDC clock as per the SCLK frequency. The suggested range of SCLK frequency applicable for each value below (when Bit[5] =0) ensures that the MDC clock is approximately between the frequency range 1.0 MHz - 2.5 MHz. When the MSB of this field is set, you can achieve MDC clock of frequency higher than the IEEE 802.3 specified frequency limit of 2.5 MHz and program a clock divider of lower value. For example, when SCLK=100 MHz and you program these bits to b#1010, the resulting MDC clock is 12.5 MHz, which is outside the limit of IEEE 802.3 specified range. Use the values shown only if the interface chips support faster MDC clocks.	
		0	MDC Clock=SCLK/42 (for SCLK=60-100MHz)
		1	MDC Clock= SCLK/62 (for SCLK=100-125 MHz)
		2	MDC Clock= SCLK/16 (for SCLK=20-35 MHz)
		3	MDC Clock= SCLK/26 (for SCLK=35-60 MHz)
		8	MDC Clock=SCLK/4
		9	MDC Clock=SCLK/6
		10	MDC Clock=SCLK/8
		11	MDC Clock=SCLK/10
		12	MDC Clock=SCLK/12
		13	MDC Clock=SCLK/14
		14	MDC Clock=SCLK/16
		15	MDC Clock=SCLK/18
1 (R/W)	SMIW	SMI Write. The EMAC_SMI_ADDR.SMIW bit, when set, tells the PHY this is a Write operation using the Station Management Interface Data register. If this bit is not set, this is a Read operation.	

Table 23-50: EMAC_SMI_ADDR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W1S)	SMIB	SMI Busy. The EMAC_SMI_ADDR.SMIB bit should read low (=0) before writing to the EMAC_SMI_ADDR and EMAC_SMI_DATA registers. This bit must also =0 during a Write to EMAC_SMI_ADDR. During a PHY register access, this bit is set (=1) by the Application to indicate that a Read or Write access is in progress. The EMAC_SMI_DATA register should be kept valid until this bit is cleared by the MAC during a PHY Write operation. EMAC_SMI_DATA is invalid until this bit is cleared by the MAC during a PHY Read operation. The EMAC_SMI_ADDR should not be written to until this bit is cleared.

SMI Data Register

The EMAC_SMI_DATA register contains the station management interface data.

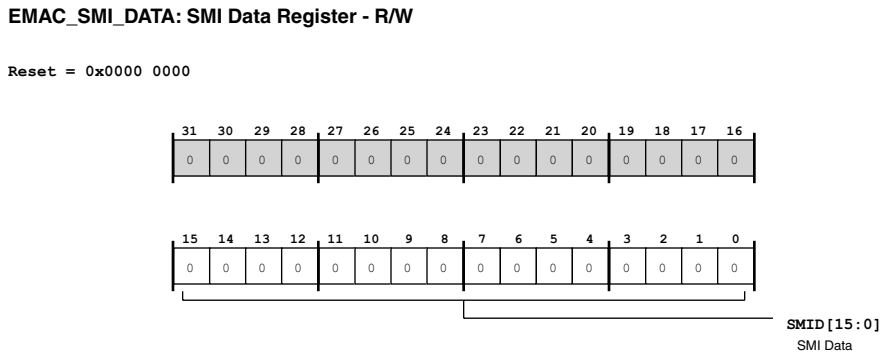


Figure 23-21: EMAC_SMI_DATA Register Diagram

Table 23-51: EMAC_SMI_DATA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	SMID	SMI Data. The EMAC_SMI_DATA.SMID bits contain the 16-bit data value read from the PHY after a Management Read operation or the 16-bit data value to be written to the PHY before a Management Write operation.

FLow Control Register

The EMAC_FLOWCTL register controls EMAC flow control features.

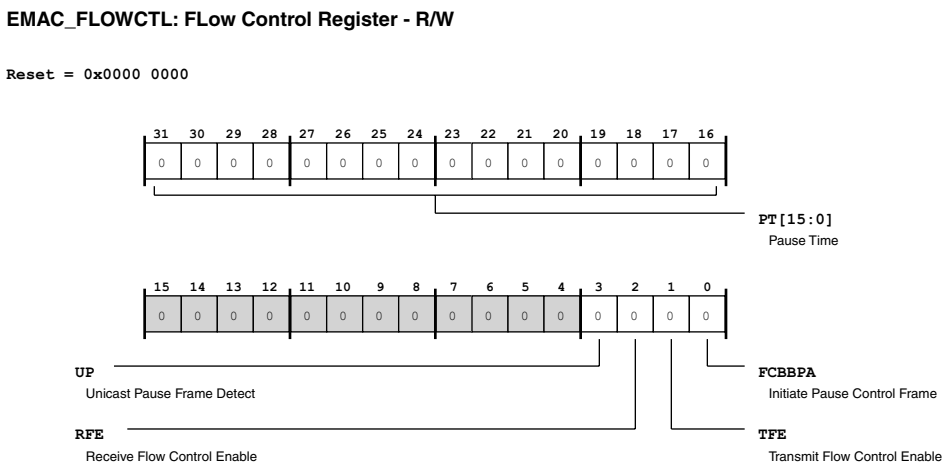


Figure 23-22: EMAC_FLOWCTL Register Diagram

Table 23-52: EMAC_FLOWCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	PT	Pause Time. The EMAC_FLOWCTL.PT bits hold the value to be used in the Pause Time field in the transmit control frame.
3 (R/W)	UP	Unicast Pause Frame Detect. The EMAC_FLOWCTL.UP bit, when set, directs the MAC to detect the Pause frames with the station's unicast address specified in EMAC_ADDR0_HI and EMAC_ADDR0_LO address registers. This bit also directs the MAC to the detect Pause frames with the unique multicast address. When this bit is reset, the MAC will detect only a Pause frame with the unique multicast address specified in the 802.3x standard.
2 (R/W)	RFE	Receive Flow Control Enable. The EMAC_FLOWCTL.RFE bit, when set, directs the MAC to decode the received Pause frame and disable its transmitter for a specified (Pause Time) time. When this bit is reset, the decode function of the Pause frame is disabled.
1 (R/W)	TFE	Transmit Flow Control Enable. In Full-Duplex mode, when the EMAC_FLOWCTL.TFE bit is set, the MAC enables the flow control operation to transmit Pause frames. When this bit is reset, the flow control operation in the MAC is disabled, and the MAC does not transmit any Pause frames. In Half-Duplex mode, when this bit is set, the MAC enables the back pressure operation. When this bit is reset, the back pressure feature is disabled.

Table 23-52: EMAC_FLOWCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W1S)	FCBBPA	Initiate Pause Control Frame. The EMAC_FLOWCTL.FCBBPA bit initiates a Pause Control frame in Full-Duplex mode and activates the back pressure function in Half-Duplex mode if TFE bit is set. In Full-Duplex mode, this bit should be read as =0 before writing to the EMAC_FLOWCTL register. To initiate a Pause control frame, the Application must set this bit to =1. During a transfer of the Control Frame, this bit continues to be set to signify that a frame transmission is in progress. After the completion of Pause control frame transmission, the MAC resets this bit to =0. The EMAC_FLOWCTL register should not be written to until this bit is cleared. In Half-Duplex mode, when this bit is set (and EMAC_FLOWCTL.TFE is set), the back pressure is asserted by the MAC Core. During back pressure, when the MAC receives a new frame, the transmitter starts sending a JAM pattern resulting in a collision. The EMAC_FLOWCTL.FCBBPA bit is logically OR'ed with the flow control input signal for the back pressure function. When the MAC is configured to Full-Duplex mode, the back pressure function is automatically disabled.

VLAN Tag Register

The EMAC_VLANTAG register contains the VLAN tag.

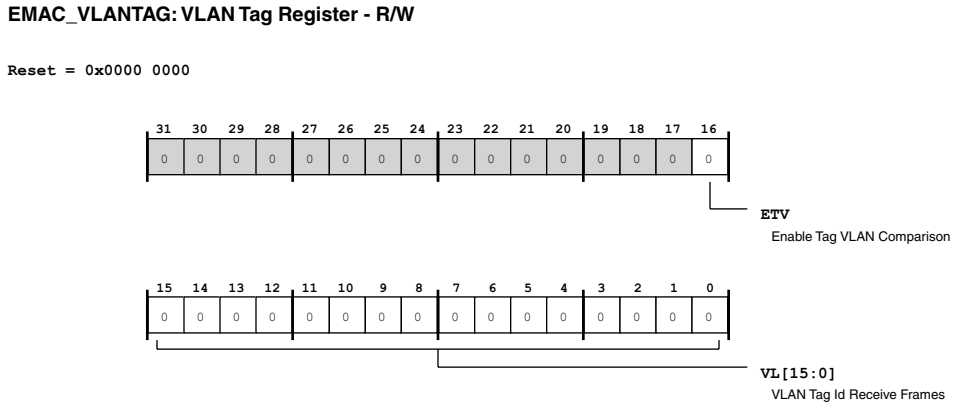


Figure 23-23: EMAC_VLANTAG Register Diagram

Table 23-53: EMAC_VLANTAG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	ETV	Enable Tag VLAN Comparison. The EMAC_VLANTAG.ETV bit, when set, directs the EMAC to use a 12-bit VLAN identifier, rather than the complete 16-bit VLAN tag, for comparison and filtering. Bits[11:0] of the VLAN tag are compared with the corresponding field in the received VLAN-tagged frame. When this bit is reset, all 16 bits of the received VLAN frame's fifteenth and sixteenth bytes are used for comparison.
15:0 (R/W)	VL	VLAN Tag Id Receive Frames. The EMAC_VLANTAG.VL bits contain the 802.1Q VLAN tag to identify VLAN frames, and is compared to the fifteenth and sixteenth bytes of the frames being received for VLAN frames. Bits[15:13] are the User Priority, Bit[12] is the Canonical Format Indicator (CFI) and bits[11:0] are the VLAN tag's VLAN Identifier (VID) field. When the ETV bit is set, only the VID (Bits[11:0]) is used for comparison. If VL (VL[11:0] if ETV is set) is all zeros, the MAC does not check the fifteenth and sixteenth bytes for VLAN tag comparison, and declares all frames with a Type field value of 0x8100 to be VLAN frames.

Debug Register

The EMAC_DBG register contains EMAC debug status information.

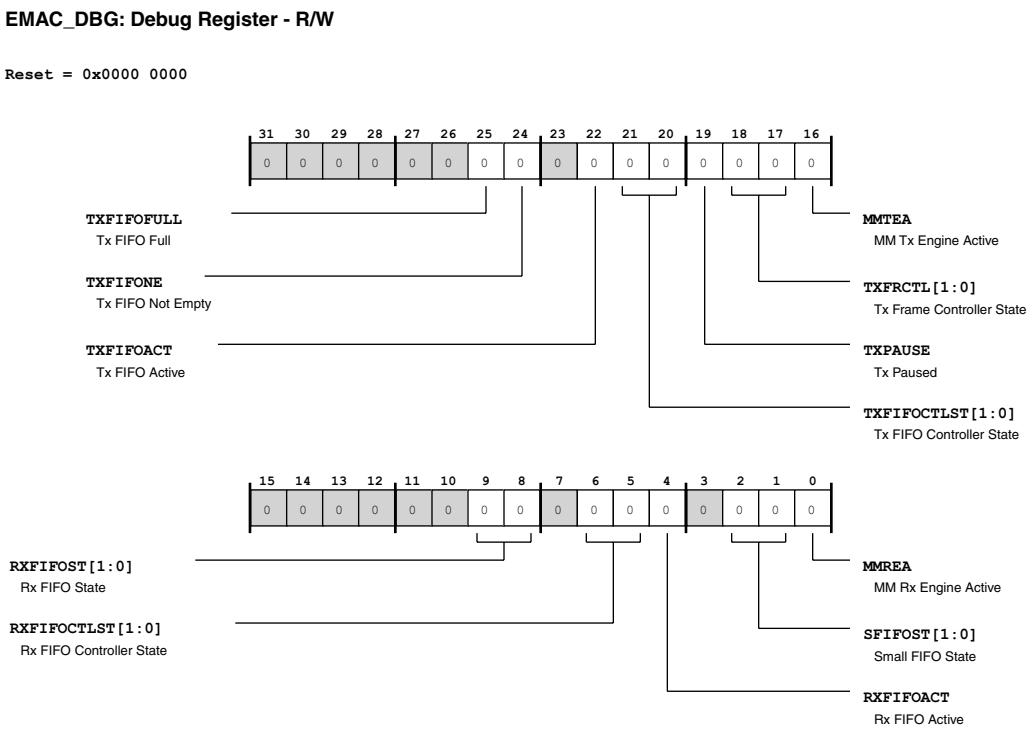


Figure 23-24: EMAC_DBG Register Diagram

Table 23-54: EMAC_DBG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/NW)	TXFIFOFULL	Tx FIFO Full. The EMAC_DBG.TXFIFOFULL bit, when high, indicates that the MFL TxStatus FIFO is full, and the MFL cannot accept any more frames for transmission.
24 (R/NW)	TXFIFONE	Tx FIFO Not Empty. The EMAC_DBG.TXFIFONE bit, when high, indicates that the MFL TxFIFO is not empty and has some data left for transmission.
22 (R/NW)	TXFIFOACT	Tx FIFO Active. The EMAC_DBG.TXFIFOACT bit, when high, indicates that the MFL TxFIFO write controller is active and transferring data to the TxFIFO.
21:20 (R/NW)	TXFIFOCTLST	Tx FIFO Controller State. The EMAC_DBG.TXFIFOCTLST bits indicate the state of the TxFIFO read controller as: 00=IDLE state, 01=READ state (transferring data to MAC transmitter), 10=Waiting for TxStatus from MAC transmitter, and 11=Writing the received TxStatus or flushing the TxFIFO

Table 23-54: EMAC_DBG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/NW)	TXPAUSE	Tx Paused. The EMAC_DBG.TXPAUSE bit, when high, indicates that the MAC transmitter is in PAUSE condition (in full-duplex only) and does not schedule any frame for transmission.	
18:17 (R/NW)	TXFRCTL	Tx Frame Controller State. The EMAC_DBG.TXFRCTL bits indicate the state of the MAC transmit frame controller module.	
		0	Idle Frame controller is in idle state.
		1	Wait Frame controller is waiting for status of previous frame or IFG/backoff period end.
		2	Pause Frame controller is generating and transmitting a PAUSE control frame (in full duplex mode).
		3	Transmit Frame controller is transferring input frame for transmission.
16 (R/NW)	MMTEA	MM Tx Engine Active. The EMAC_DBG.MMTEA bit, when high, indicates that the MAC core transmit protocol engine is actively transmitting data and is not in IDLE state.	
9:8 (R/NW)	RXFIFOST	Rx FIFO State. The EMAC_DBG.RXFIFOST bits give the status of the RxFIFO fill level and indicate the relationship to the flow-control activation threshold.	
		0	Rx FIFO Empty
		1	Rx FIFO Below De-activate FCT
		2	Rx FIFO Above De-activate FCT
		3	Rx FIFO Full

Table 23-54: EMAC_DBG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6:5 (R/NW)	RXFIFOCTLST	Rx FIFO Controller State. The EMAC_DBG.RXFIFOCTLST bits give the state of the RxFIFO read controller.	
		0	Idle Read controller is in idle state.
		1	Read Data Read controller is reading frame data.
		2	Read Status Read controller is reading frame status or time-stamp.
		3	Flush Read controller is flushing the frame data and status.
4 (R/NW)	RXFIFOACT	Rx FIFO Active. The EMAC_DBG.RXFIFOACT bit, when high, indicates that the MFL RxFIFO write controller is active and is transferring a received frame to the FIFO.	
2:1 (R/NW)	SFIFOST	Small FIFO State. The EMAC_DBG.SFIFOST bit, when high, indicates the active state of the small FIFO read and write controllers respectively of the MAC receive frame controller module.	
0 (R/NW)	MMREA	MM Rx Engine Active. The EMAC_DBG.MMREA bit, when high, indicates that the MAC core receive protocol engine is actively receiving data and is not in IDLE state.	

Interrupt Status Register

The EMAC_ISTAT register indicates EMAC interrupt status.

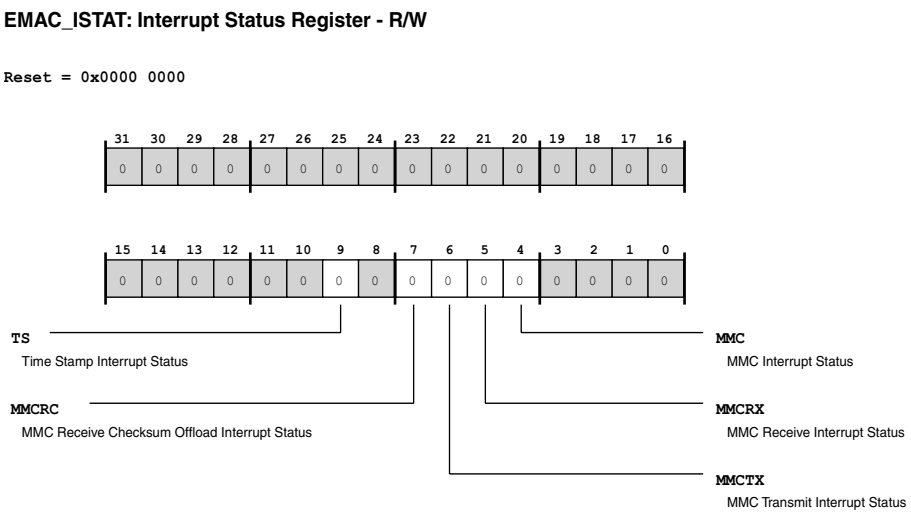


Figure 23-25: EMAC_ISTAT Register Diagram

Table 23-55: EMAC_ISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/NW)	TS	Time Stamp Interrupt Status. When Advanced Time Stamping feature is enabled, the EMAC_ISTAT.TS bit is set when: • The system time value equals or exceeds the value specified in the EMAC_TM_TGTM and EMAC_TM_NTGTM registers, or • There is an overflow in the EMAC_TM_SEC register, or • When the EMAC_TM_STMPSTAT.ATSTS bit is asserted. The EMAC_ISTAT.TS bit is cleared on reading the byte 0 of the EMAC_TM_STMPSTAT register. Otherwise, when default Time-Stamping is enabled, this bit, when set, indicates that the system time value equals or exceeds the value specified in the EMAC_TM_TGTM and EMAC_TM_NTGTM registers. In this mode, this bit is cleared after the completion of the read of the EMAC_ISTAT register. In all other modes, this bit is reserved.
7 (R/NW)	MMCRC	MMC Receive Checksum Offload Interrupt Status. The EMAC_ISTAT.MMCRC bit is set high whenever an interrupt is generated in the EMAC_IPC_RXINT. This bit is cleared when all the bits in this interrupt register are cleared.
6 (R/NW)	MMCTX	MMC Transmit Interrupt Status. The EMAC_ISTAT.MMCTX bit is set high whenever an interrupt is generated in the EMAC_MMC_TXINT register. This bit is cleared when all the bits in this interrupt register are cleared.

Table 23-55: EMAC_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/NW)	MMCRX	MMC Receive Interrupt Status. The EMAC_ISTAT.MMCRX bit is set high whenever an interrupt is generated in the EMAC_MMC_RXINT register. This bit is cleared when all the bits in this interrupt register are cleared.
4 (R/NW)	MMC	MMC Interrupt Status. The EMAC_ISTAT.MMC bit is set high whenever any of EMAC_ISTAT bits [7:5] is set (=1) and is cleared only when all of these bits are cleared (=0).

Interrupt Mask Register

The EMAC_IMSK register enables (unmasks) EMAC interrupts.

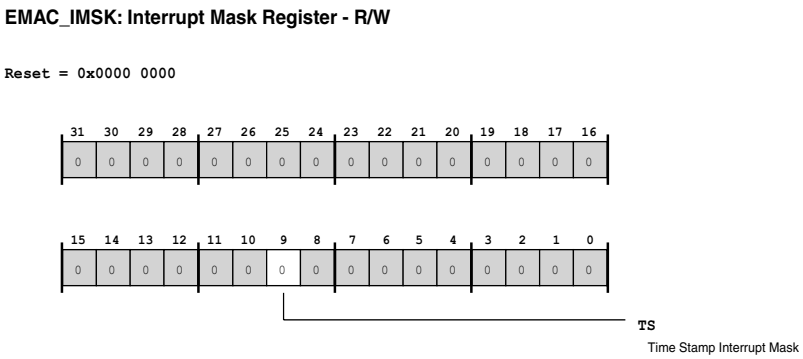


Figure 23-26: EMAC_IMSK Register Diagram

Table 23-56: EMAC_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	TS	Time Stamp Interrupt Mask. The EMAC_IMSK.TS bit, when set, disables the assertion of the interrupt signal, which is generated when the EMAC_ISTAT.TS bit is set.

MAC Address 0 High Register

The EMAC_ADDR0_HI register holds the address 0 high bits.

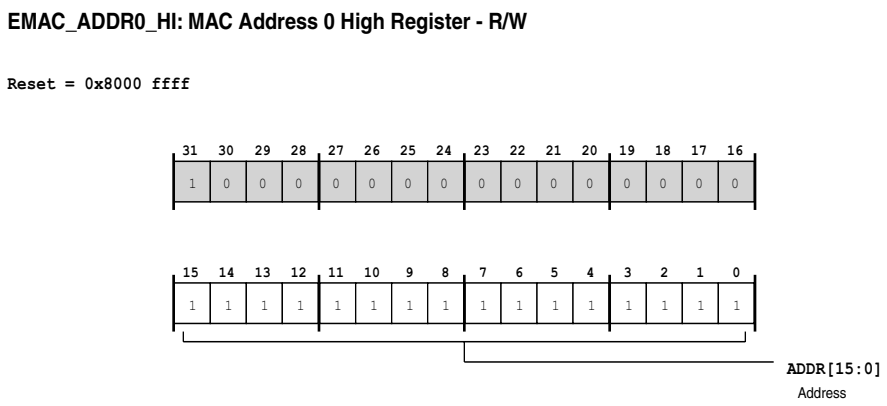


Figure 23-27: EMAC_ADDR0_HI Register Diagram

Table 23-57: EMAC_ADDR0_HI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	ADDR	Address. The EMAC_ADDR0_HI.ADDR bits contain the upper 16 bits (47:32) of the 6-byte first MAC address. This address is used by the MAC for filtering for received frames and for inserting the MAC address in the Transmit Flow Control (PAUSE) Frames.

MAC Address 0 Low Register

The EMAC_ADDR0_LO register holds the address 0 low bits.

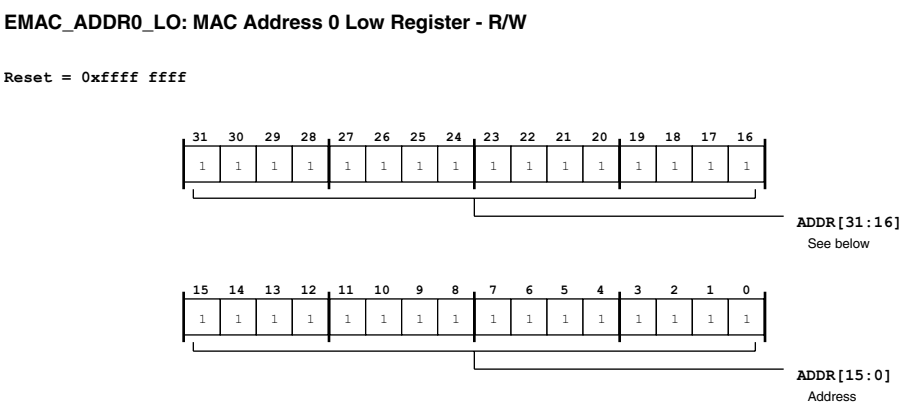


Figure 23-28: EMAC_ADDR0_LO Register Diagram

Table 23-58: EMAC_ADDR0_LO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Address. The EMAC_ADDR0_LO.ADDR bits contain the lower 32 bits of the 6-byte first MAC address. This address is used by the MAC for filtering for received frames and for inserting the MAC address in the Transmit Flow Control (PAUSE) Frames.

MMC Control Register

The EMAC_MMC_CTL register selects the MMC operating mode.

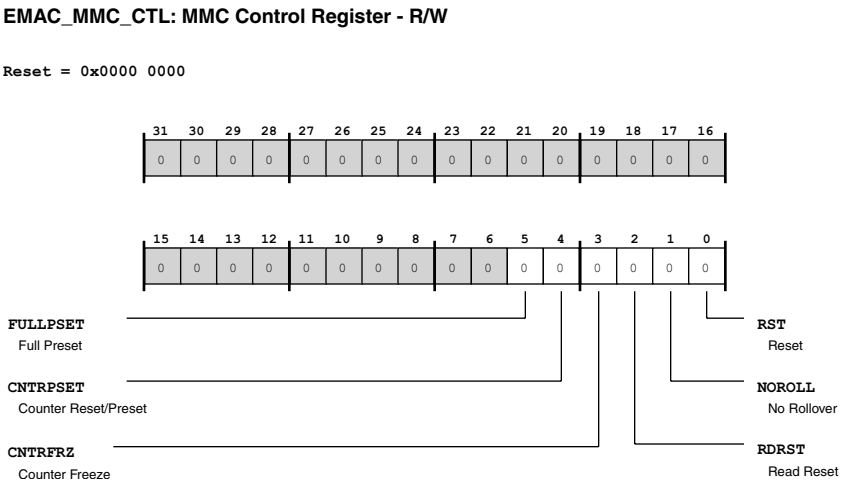


Figure 23-29: EMAC_MMC_CTL Register Diagram

Table 23-59: EMAC_MMC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W)	FULLPSET	Full Preset. The EMAC_MMC_CTL.FULLPSET bit, when =0 (and EMAC_MMC_CTL.CNTRPSET =1), presets all MMC counters to almost-half value. All octet counters get preset to 0x7FFF_F800 (half - 2KBytes) and all frame-counters gets preset to 0x7FFF_FFF0 (half - 16). When EMAC_MMC_CTL.FULLPSET =1 (and EMAC_MMC_CTL.CNTRPSET =1), all MMC counters get preset to almost-full value. All octet counters get preset to 0xFFFF_F800 (full - 2KBytes) and all frame-counters gets preset to 0xFFFF_FFF0 (full - 16). For 16-bit counters, the almost-half preset values are 0x7800 and 0x7FF0 for the respective octet and frame counters. Similarly, the almost-full preset values for the 16-bit counters are 0xF800 and 0xFFF0.
4 (R/W)	CNTRPSET	Counter Reset/Preset. The EMAC_MMC_CTL.CNTRPSET bit, when set, initializes all counters or presets counters to almost full or almost half as per EMAC_MMC_CTL.FULLPSET. The EMAC_MMC_CTL.CNTRPSET bit is cleared automatically after 1 clock cycle. This bit along with bit5 is useful for debugging and testing the assertion of interrupts because of MMC counter becoming half-full or full.
3 (R/W)	CNTRFRZ	Counter Freeze. The EMAC_MMC_CTL.CNTRFRZ bit, when set, freezes all the MMC counters to their current value. None of the MMC counters are updated due to any transmitted or received frame, until this bit is reset to 0. If any MMC counter is read with the EMAC_MMC_CTL.RDRST bit set, then that counter is also cleared in this mode.
2 (R/W)	RDRST	Read Reset. The EMAC_MMC_CTL.RDRST bit, when set, resets the MMC counters to zero after Read (self-clearing after reset). The counters are cleared when the least significant byte lane (bits[7:0]) is read.
1 (R/W)	NOROLL	No Rollover. The EMAC_MMC_CTL.NOROLL bit, when set, prevents counter rolls over to 0 after reaching max.
0 (R/W)	RST	Reset. The EMAC_MMC_CTL.RST bit, when set, resets all counters. This bit is cleared automatically after 1 clock cycle

MMC Rx Interrupt Register

The EMAC_MMC_RXINT register indicates status of MMC receive interrupts.

EMAC_MMC_RXINT: MMC Rx Interrupt Register - R/W

Reset = 0x0000 0000

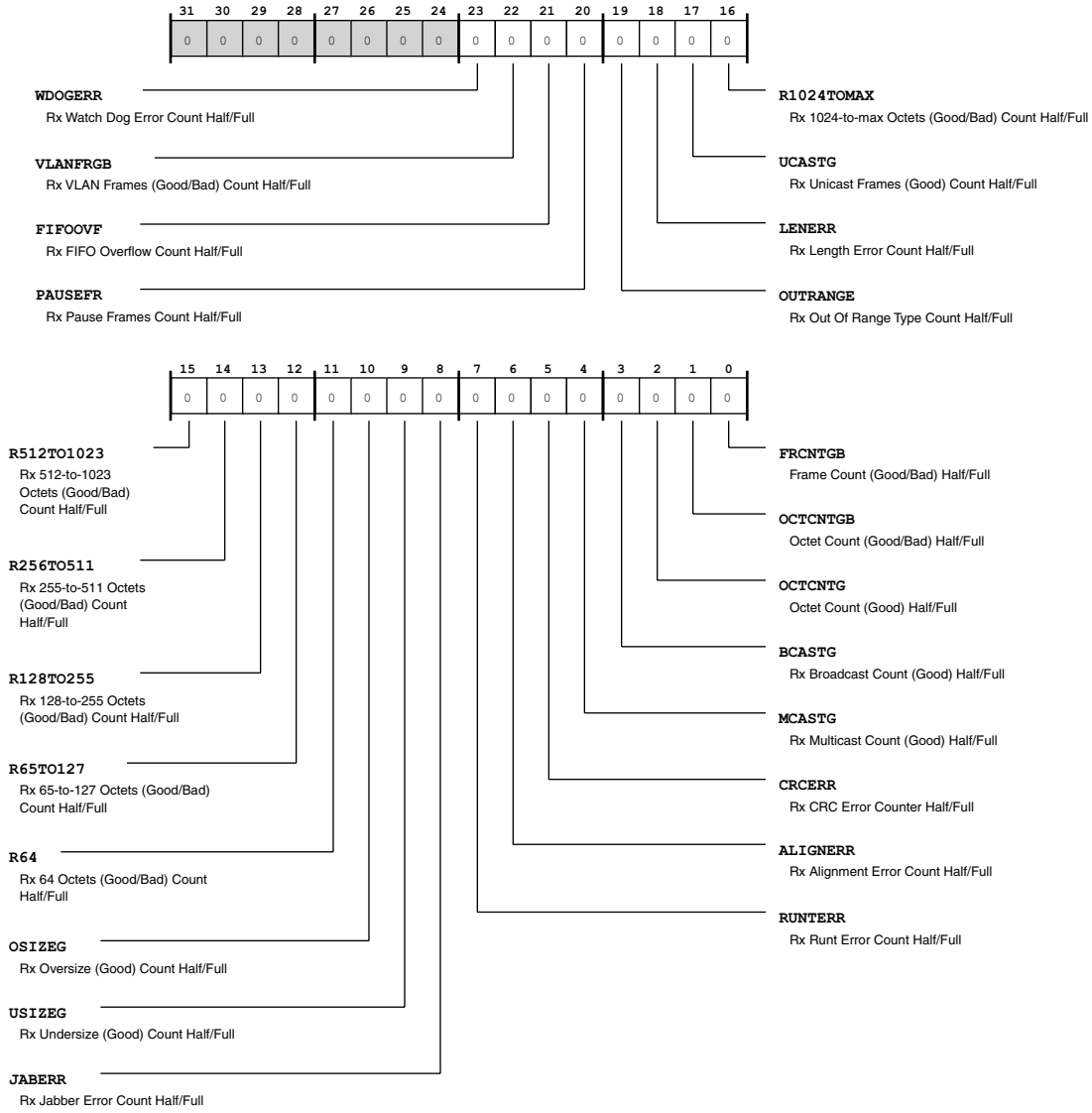


Figure 23-30: EMAC_MMC_RXINT Register Diagram

Table 23-60: EMAC_MMC_RXINT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23 (R/NW)	WDOGERR	Rx Watch Dog Error Count Half/Full. The EMAC_MMC_RXINT.WDOGERR bit is set when the EMAC_RXWDOG_ERR counter reaches full or half.
22 (R/NW)	VLANFRGB	Rx VLAN Frames (Good/Bad) Count Half/Full. The EMAC_MMC_RXINT.VLANFRGB bit is set when EMAC_RXVLANFRM_GB counter reaches full or half.
21 (R/NW)	FIFOOVF	Rx FIFO Overflow Count Half/Full. The EMAC_MMC_RXINT.FIFOOVF bit is set when EMAC_RXFIFO_OVF counter reaches full or half.
20 (R/NW)	PAUSEFR	Rx Pause Frames Count Half/Full. The EMAC_MMC_RXINT.PAUSEFR bit is set when EMAC_RXPAUSEFRM counter reaches full or half.
19 (R/NW)	OUTRANGE	Rx Out Of Range Type Count Half/Full. The EMAC_MMC_RXINT.OUTRANGE bit is set when EMAC_RXOORTYPE counter reaches full or half.
18 (R/NW)	LENERR	Rx Length Error Count Half/Full. The EMAC_MMC_RXINT.LENERR bit is set when EMAC_RXLEN_ERR counter reaches full or half.
17 (R/NW)	UCASTG	Rx Unicast Frames (Good) Count Half/Full. The EMAC_MMC_RXINT.UCASTG bit is set when EMAC_RXUCASTFRM_G counter reaches full or half.
16 (R/NW)	R1024TOMAX	Rx 1024-to-max Octets (Good/Bad) Count Half/Full. The EMAC_MMC_RXINT.R1024TOMAX bit is set when EMAC_RX1024TOMAX_GBcounter reaches full or half.
15 (R/NW)	R512TO1023	Rx 512-to-1023 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_RXINT.R512TO1023 bit is set when EMAC_RX512TO1023_GB counter reaches full or half.
14 (R/NW)	R256TO511	Rx 255-to-511 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_RXINT.R256TO511 bit is set when EMAC_RX256TO511_GB counter reaches full or half.
13 (R/NW)	R128TO255	Rx 128-to-255 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_RXINT.R128TO255 bit is set when EMAC_RX128TO255_GB counter reaches full or half.
12 (R/NW)	R65TO127	Rx 65-to-127 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_RXINT.R65TO127 bit is set when EMAC_RX65TO127_GB counter reaches full or half.

Table 23-60: EMAC_MMC_RXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/NW)	R64	Rx 64 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_RXINT.R64 bit is set when EMAC_RX64_GB counter reaches full or half.
10 (R/NW)	OSIZEG	Rx Oversize (Good) Count Half/Full. The EMAC_MMC_RXINT.OSIZEG bit is set when EMAC_RXOSIZE_G counter reaches full or half.
9 (R/NW)	USIZEG	Rx Undersize (Good) Count Half/Full. The EMAC_MMC_RXINT.USIZEG bit is set when EMAC_RXUSIZE_G counter reaches full or half.
8 (R/NW)	JABERR	Rx Jabber Error Count Half/Full. The EMAC_MMC_RXINT.JABERR bit is set when EMAC_RXJAB_ERR counter reaches full or half.
7 (R/NW)	RUNTERR	Rx Runt Error Count Half/Full. The EMAC_MMC_RXINT.RUNTERR bit is set when EMAC_RXRUNT_ERR counter reaches full or half.
6 (R/NW)	ALIGNERR	Rx Alignment Error Count Half/Full. The EMAC_MMC_RXINT.ALIGNERR bit is set when EMAC_RXALIGN_ERR counter reaches full or half
5 (R/NW)	CRCERR	Rx CRC Error Counter Half/Full. The EMAC_MMC_RXINT.CRCERR bit is set when EMAC_RXCRC_ERR counter reaches full or half.
4 (R/NW)	MCASTG	Rx Multicast Count (Good) Half/Full. The EMAC_MMC_RXINT.MCASTG bit is set when EMAC_RXMCASTFRM_G counter reaches full or half.
3 (R/NW)	BCASTG	Rx Broadcast Count (Good) Half/Full. The EMAC_MMC_RXINT.BCASTG bit is set when EMAC_RXBCASTFRM_G counter reaches full or half.
2 (R/NW)	OCTCNTG	Octet Count (Good) Half/Full. The EMAC_MMC_RXINT.OCTCNTG bit is set when EMAC_RXOCTCNT_G counter reaches full or half.
1 (R/NW)	OCTCNTGB	Octet Count (Good/Bad) Half/Full. The EMAC_MMC_RXINT.OCTCNTGB bit is set when EMAC_RXOCTCNT_GB counter reaches half or full.
0 (R/NW)	FRCNTGB	Frame Count (Good/Bad) Half/Full. The EMAC_MMC_RXINT.FRCNTGB bit is set when EMAC_RXFRMCNT_GB counter reaches half or full.

MMC Tx Interrupt Register

The EMAC_MMC_TXINT register indicates status of MMC transmit interrupts.

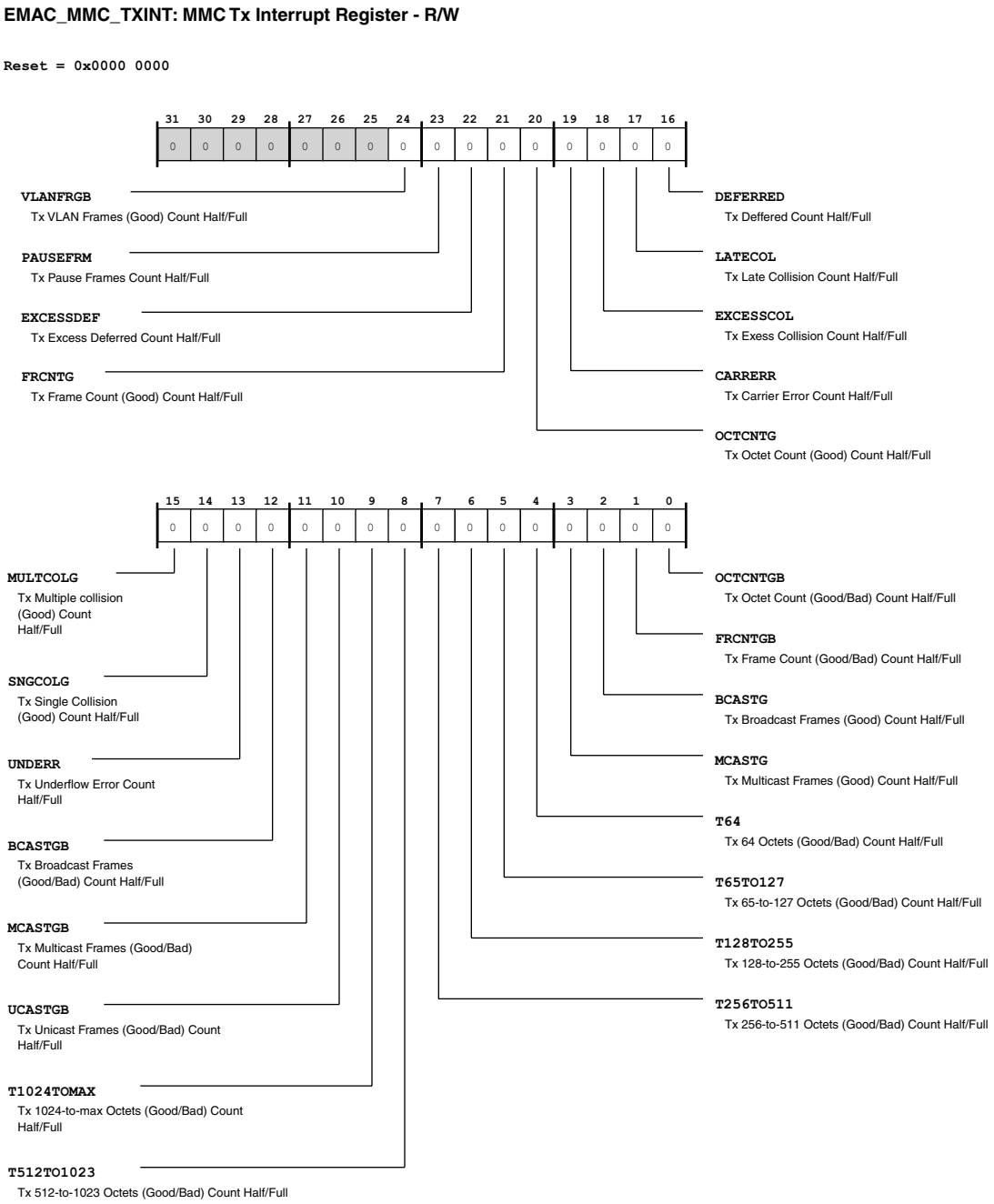


Figure 23-31: EMAC_MMC_TXINT Register Diagram

Table 23-61: EMAC_MMC_TXINT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/NW)	VLANFRGB	Tx VLAN Frames (Good) Count Half/Full. The EMAC_MMC_TXINT.VLANFRGB bit is set when EMAC_TXVLANFRM_G counter reaches full or half.
23 (R/NW)	PAUSEFRM	Tx Pause Frames Count Half/Full. The EMAC_MMC_TXINT.PAUSEFRM bit is set when EMAC_TXPAUSEFRM counter reaches full or half.
22 (R/NW)	EXCESSDEF	Tx Excess Deferred Count Half/Full. The EMAC_MMC_TXINT.EXCESSDEF bit is set when EMAC_TXEXCESSDEF counter reaches full or half.
21 (R/NW)	FRCNTG	Tx Frame Count (Good) Count Half/Full. The EMAC_MMC_TXINT.FRCNTG bit is set when EMAC_TXFRMCNT_G counter reaches full or half.
20 (R/NW)	OCTCNTG	Tx Octet Count (Good) Count Half/Full. The EMAC_MMC_TXINT.OCTCNTG bit is set when EMAC_TXOCTCNT_G counter reaches full or half.
19 (R/NW)	CARRERR	Tx Carrier Error Count Half/Full. The EMAC_MMC_TXINT.CARRERR bit is set when EMAC_TXCARR_ERR counter reaches full or half.
18 (R/NW)	EXCESSCOL	Tx Exess Collision Count Half/Full. The EMAC_MMC_TXINT.EXCESSCOL bit is set when EMAC_TXEXCESSCOL counter reaches full or half.
17 (R/NW)	LATECOL	Tx Late Collision Count Half/Full. The EMAC_MMC_TXINT.LATECOL bit is set when EMAC_TXLATECOL counter reaches full or half.
16 (R/NW)	DEFERRED	Tx Deffered Count Half/Full. The EMAC_MMC_TXINT.DEFERRED bit is set when EMAC_TXDEFERRED counter reaches full or half.
15 (R/NW)	MULTCOLG	Tx Multiple collision (Good) Count Half/Full. The EMAC_MMC_TXINT.MULTCOLG bit is set when EMAC_TXMULTCOL_G counter reaches full or half.
14 (R/NW)	SNGCOLG	Tx Single Collision (Good) Count Half/Full. The EMAC_MMC_TXINT.SNGCOLG bit is set when EMAC_TXSNGCOL_G counter reaches full or half.
13 (R/NW)	UNDERR	Tx Underflow Error Count Half/Full. The EMAC_MMC_TXINT.UNDERR bit is set when EMAC_TXUNDR_ERR counter reaches full or half.

Table 23-61: EMAC_MMC_TXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/NW)	BCASTGB	Tx Broadcast Frames (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.BCASTGB bit is set when EMAC_TXBCASTFRM_GB counter reaches full or half.
11 (R/NW)	MCASTGB	Tx Multicast Frames (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.MCASTGB bit is set when EMAC_TXMCASTFRM_GB counter reaches full or half.
10 (R/NW)	UCASTGB	Tx Unicast Frames (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.UCASTGB bit is set when EMAC_TXUCASTFRM_GB counter reaches full or half.
9 (R/NW)	T1024TOMAX	Tx 1024-to-max Octets (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.T1024TOMAX bit is set when EMAC_TX1024TOMAX_GB counter reaches full or half.
8 (R/NW)	T512TO1023	Tx 512-to-1023 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.T512TO1023 bit is set when EMAC_TX512TO1023_GB counter reaches full or half.
7 (R/NW)	T256TO511	Tx 256-to-511 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.T256TO511 bit is set when EMAC_TX256TO511_GB counter reaches full or half.
6 (R/NW)	T128TO255	Tx 128-to-255 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.T128TO255 bit is set when EMAC_TX128TO255_GB counter reaches full or half.
5 (R/NW)	T65TO127	Tx 65-to-127 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.T65TO127 bit is set when EMAC_TX65TO127_GB counter reaches full or half.
4 (R/NW)	T64	Tx 64 Octets (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.T64 bit is set when EMAC_TX64_GB counter reaches full or half.
3 (R/NW)	MCASTG	Tx Multicast Frames (Good) Count Half/Full. The EMAC_MMC_TXINT.MCASTG bit is set when EMAC_TXMCASTFRM_G counter reaches full or half.
2 (R/NW)	BCASTG	Tx Broadcast Frames (Good) Count Half/Full. The EMAC_MMC_TXINT.BCASTG bit is set when EMAC_TXBCASTFRM_G counter reaches full or half.
1 (R/NW)	FRCNTGB	Tx Frame Count (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.FRCNTGB bit is set when EMAC_TXFRMCNT_GB counter reaches full or half.

Table 23-61: EMAC_MMC_TXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/NW)	OCTCNTGB	Tx Octet Count (Good/Bad) Count Half/Full. The EMAC_MMC_TXINT.OCTCNTGB bit is set when EMAC_TXOCTCNT_GB counter reaches full or half.

MMC Rx Interrupt Mask Register

The EMAC_MMC_RXIMSK register enables (unmasks) MMC receive interrupts.

EMAC_MMC_RXIMSK: MMC Rx Interrupt Mask Register - R/W

Reset = 0x0000 0000

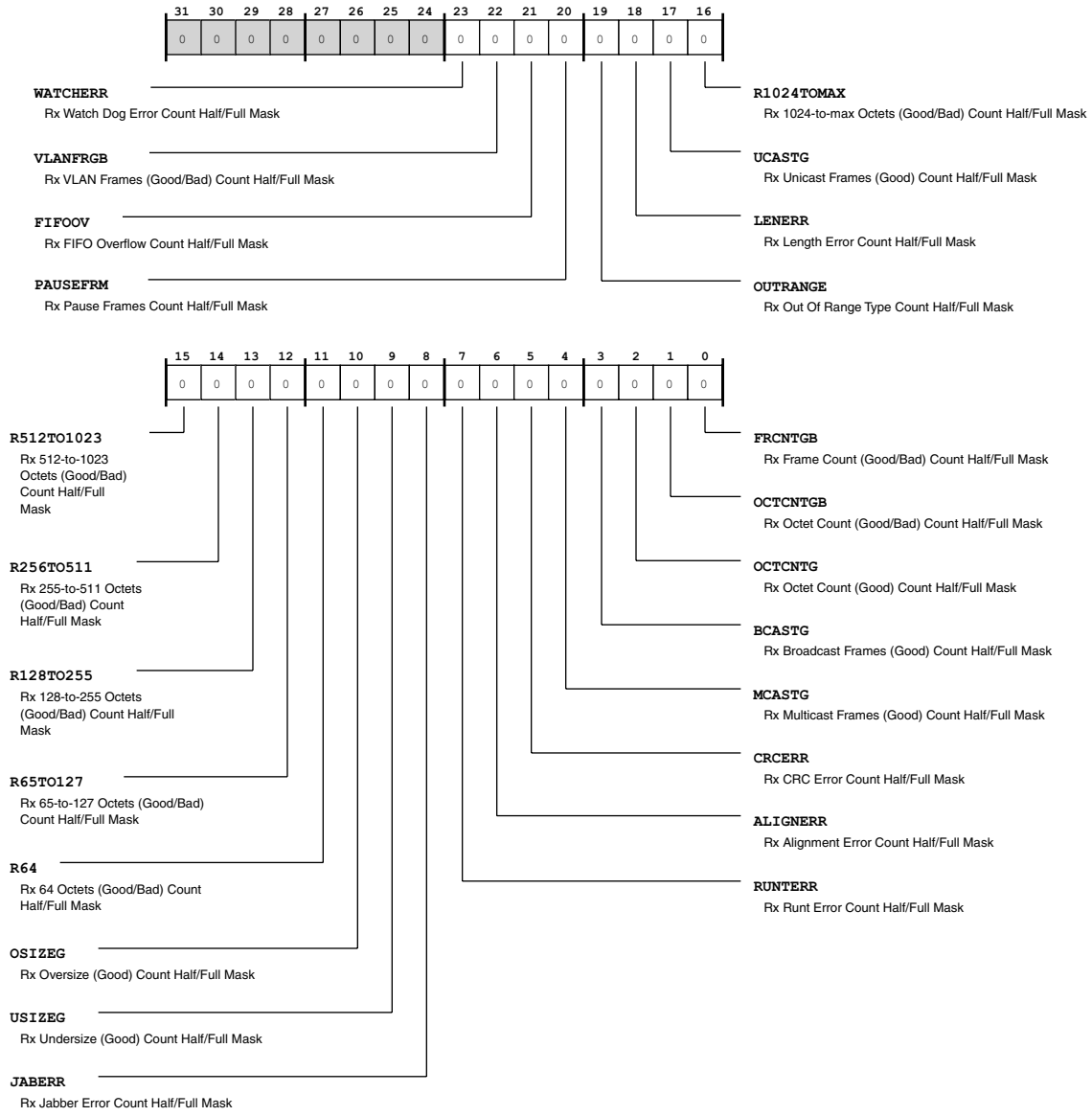


Figure 23-32: EMAC_MMC_RXIMSK Register Diagram

Table 23-62: EMAC_MMC_RXIMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23 (R/W)	WATCHERR	Rx Watch Dog Error Count Half/Full Mask. The EMAC_MMC_RXIMSK.WATCHERR bit, when set, masks the interrupt when EMAC_RXWDOG_ERR counter reaches full or half.

Table 23-62: EMAC_MMC_RXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
22 (R/W)	VLANFRGB	Rx VLAN Frames (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.VLANFRGB bit, when set, masks the interrupt when EMAC_RXVLANFRM_GB counter reaches full or half.
21 (R/W)	FIFOOV	Rx FIFO Overflow Count Half/Full Mask. The EMAC_MMC_RXIMSK.FIFOOV bit, when set, masks the interrupt when EMAC_RXFIFO_OVF counter reaches full or half.
20 (R/W)	PAUSEFRM	Rx Pause Frames Count Half/Full Mask. The EMAC_MMC_RXIMSK.PAUSEFRM bit, when set, masks the interrupt when EMAC_RXPAUSEFRM counter reaches full or half.
19 (R/W)	OUTRANGE	Rx Out Of Range Type Count Half/Full Mask. The EMAC_MMC_RXIMSK.OUTRANGE bit, when set, masks the interrupt when EMAC_RXOORTYPE counter reaches full or half.
18 (R/W)	LENERR	Rx Length Error Count Half/Full Mask. The EMAC_MMC_RXIMSK.LENERR bit, when set, masks the interrupt when EMAC_RXLEN_ERR counter reaches full or half.
17 (R/W)	UCASTG	Rx Unicast Frames (Good) Count Half/Full Mask. The EMAC_MMC_RXIMSK.UCASTG bit, when set, masks the interrupt when EMAC_RXUCASTFRM_G counter reaches full or half.
16 (R/W)	R1024TOMAX	Rx 1024-to-max Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.R1024TOMAX bit, when set, masks the interrupt when EMAC_RX1024TOMAX_GB counter reaches full or half.
15 (R/W)	R512TO1023	Rx 512-to-1023 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.R512TO1023 bit, when set, masks the interrupt when EMAC_RX512TO1023_GB counter reaches full or half.
14 (R/W)	R256TO511	Rx 255-to-511 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.R256TO511 bit, when set, masks the interrupt when EMAC_RX256TO511_GB counter reaches full or half.
13 (R/W)	R128TO255	Rx 128-to-255 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.R128TO255 bit, when set, masks the interrupt when EMAC_RX128TO255_GB counter reaches full or half.
12 (R/W)	R65TO127	Rx 65-to-127 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.R65TO127 bit, when set, masks the interrupt when EMAC_RX65TO127_GB counter reaches full or half.
11 (R/W)	R64	Rx 64 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.R64 bit, when set, masks the interrupt when EMAC_RX64_GB counter reaches full or half.

Table 23-62: EMAC_MMC_RXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/W)	OSIZEG	Rx Oversize (Good) Count Half/Full Mask. The EMAC_MMC_RXIMSK.OSIZEG bit, when set, masks the interrupt when EMAC_RXOSIZE_G counter reaches full or half.
9 (R/W)	USIZEG	Rx Undersize (Good) Count Half/Full Mask. The EMAC_MMC_RXIMSK.USIZEG bit, when set, masks the interrupt when EMAC_RXUSIZE_G counter reaches full or half.
8 (R/W)	JABERR	Rx Jabber Error Count Half/Full Mask. The EMAC_MMC_RXIMSK.JABERR bit, when set, masks the interrupt when EMAC_RXJAB_ERR counter reaches full or half.
7 (R/W)	RUNTERR	Rx Runt Error Count Half/Full Mask. The EMAC_MMC_RXIMSK.RUNTERR bit, when set, masks the interrupt when EMAC_RXRUNT_ERR counter reaches full or half.
6 (R/W)	ALIGNERR	Rx Alignment Error Count Half/Full Mask. The EMAC_MMC_RXIMSK.ALIGNERR bit, when set, masks the interrupt when EMAC_RXALIGN_ERR counter reaches full or half.
5 (R/W)	CRCERR	Rx CRC Error Count Half/Full Mask. The EMAC_MMC_RXIMSK.CRCERR bit, when set, masks the interrupt when EMAC_RXCRC_ERR counter reaches full or half.
4 (R/W)	MCASTG	Rx Multicast Frames (Good) Count Half/Full Mask. The EMAC_MMC_RXIMSK.MCASTG bit, when set, masks the interrupt when EMAC_RXMCASTFRM_G counter reaches full or half.
3 (R/W)	BCASTG	Rx Broadcast Frames (Good) Count Half/Full Mask. The EMAC_MMC_RXIMSK.BCASTG bit, when set, masks the interrupt when EMAC_RXBCASTFRM_G counter reaches full or half.
2 (R/W)	OCTCNTG	Rx Octet Count (Good) Count Half/Full Mask. The EMAC_MMC_RXIMSK.OCTCNTG bit, when set, masks the interrupt when EMAC_RXOCTCNT_G counter reaches full or half.
1 (R/W)	OCTCNTGB	Rx Octet Count (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.OCTCNTGB bit, when set, masks the interrupt when EMAC_RXOCTCNT_GB counter reaches half or full.
0 (R/W)	FRCNTGB	Rx Frame Count (Good/Bad) Count Half/Full Mask. The EMAC_MMC_RXIMSK.FRCNTGB bit, when set, masks the interrupt when EMAC_RXFRMCNT_GB counter reaches half or full.

MMC TX Interrupt Mask Register

The EMAC_MMC_TXIMSK register enables (unmasks) MMC transmit interrupts.

EMAC_MMC_TXIMSK: MMC TX Interrupt Mask Register - R/W

Reset = 0x0000 0000

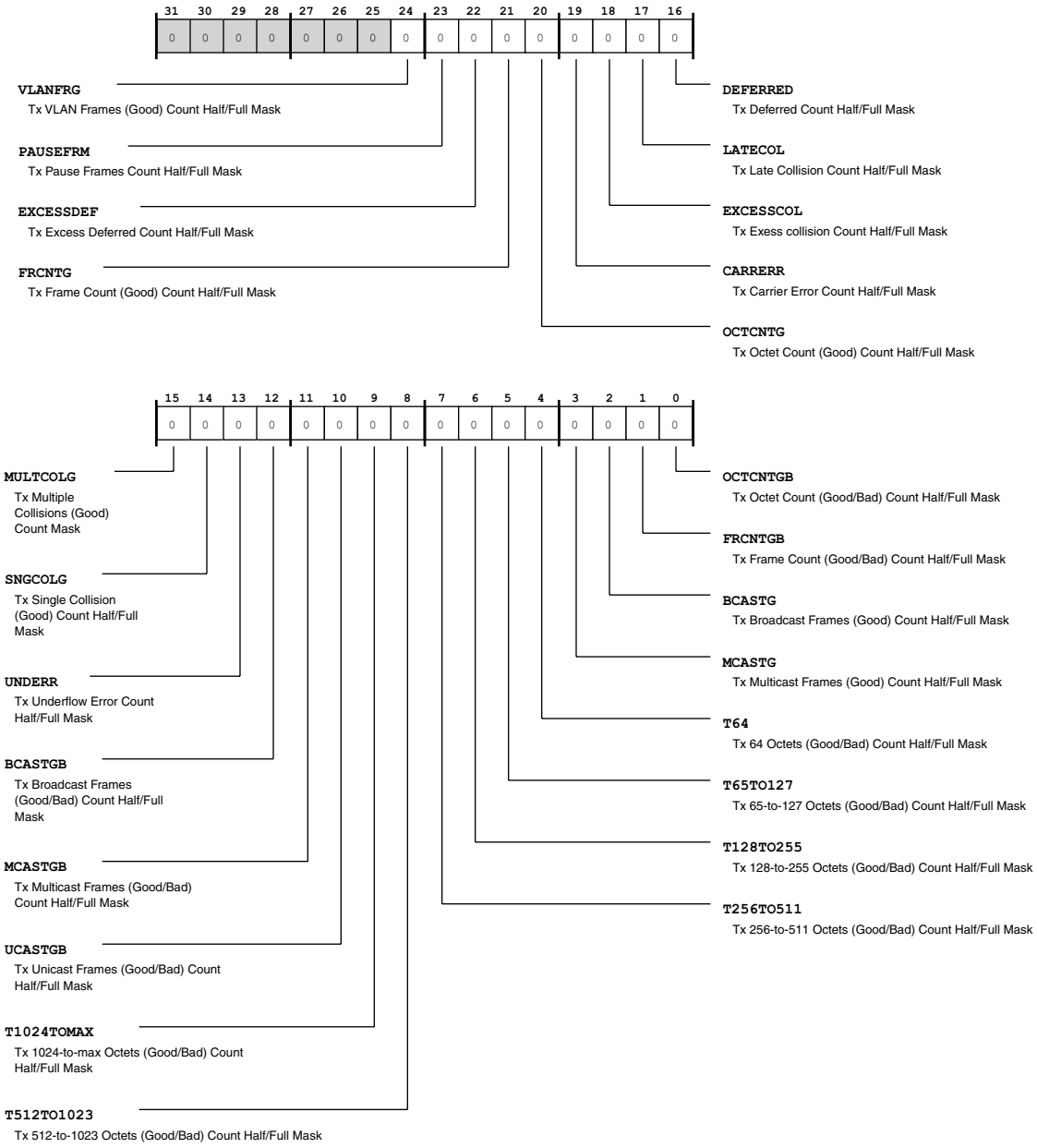


Figure 23-33: EMAC_MMC_TXIMSK Register Diagram

Table 23-63: EMAC_MMC_TXIMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/W)	VLANFRG	Tx VLAN Frames (Good) Count Half/Full Mask. The EMAC_MMC_TXIMSK.VLANFRG bit, when set, masks the interrupt when EMAC_TXVLANFRM_G counter reaches full or half.
23 (R/W)	PAUSEFRM	Tx Pause Frames Count Half/Full Mask. The EMAC_MMC_TXIMSK.PAUSEFRM bit, when set, masks the interrupt when EMAC_TXPAUSEFRM counter reaches full or half.
22 (R/W)	EXCESSDEF	Tx Excess Deferred Count Half/Full Mask. The EMAC_MMC_TXIMSK.EXCESSDEF bit, when set, masks the interrupt when EMAC_TXEXCESSDEF counter reaches full or half.
21 (R/W)	FRCNTG	Tx Frame Count (Good) Count Half/Full Mask. The EMAC_MMC_TXIMSK.FRCNTG bit, when set, masks the interrupt when EMAC_TXFRMCNT_G counter reaches full or half.
20 (R/W)	OCTCNTG	Tx Octet Count (Good) Count Half/Full Mask. The EMAC_MMC_TXIMSK.OCTCNTG bit, when set, masks the interrupt when EMAC_TXOCTCNT_G counter reaches full or half.
19 (R/W)	CARRERR	Tx Carrier Error Count Half/Full Mask. The EMAC_MMC_TXIMSK.CARRERR bit, when set, masks the interrupt when EMAC_TXCARR_ERR counter reaches full or half.
18 (R/W)	EXCESSCOL	Tx Exess collision Count Half/Full Mask. The EMAC_MMC_TXIMSK.EXCESSCOL bit, when set, masks the interrupt when EMAC_TXEXCESSCOL counter reaches full or half.
17 (R/W)	LATECOL	Tx Late Collision Count Half/Full Mask. The EMAC_MMC_TXIMSK.LATECOL bit, when set, masks the interrupt when EMAC_TXLATECOL counter reaches full or half.
16 (R/W)	DEFERRED	Tx Deferred Count Half/Full Mask. The EMAC_MMC_TXIMSK.DEFERRED bit, when set, masks the interrupt when EMAC_TXDEFERRED counter reaches full or half.
15 (R/W)	MULTCOLG	Tx Multiple Collisions (Good) Count Mask. The EMAC_MMC_TXIMSK.MULTCOLG bit, when set, masks the interrupt when EMAC_TXMULTCOL_G counter reaches full or half.
14 (R/W)	SNGCOLG	Tx Single Collision (Good) Count Half/Full Mask. The EMAC_MMC_TXIMSK.SNGCOLG bit, when set, masks the interrupt when EMAC_TXSNGCOL_G counter reaches full or half.
13 (R/W)	UNDERR	Tx Underflow Error Count Half/Full Mask. The EMAC_MMC_TXIMSK.UNDERR bit, when set, masks the interrupt when EMAC_TXUNDR_ERR counter reaches full or half.

Table 23-63: EMAC_MMC_TXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	BCASTGB	Tx Broadcast Frames (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.BCASTGB bit, when set, masks the interrupt when EMAC_TXBCASTFRM_GB counter reaches full or half.
11 (R/W)	MCASTGB	Tx Multicast Frames (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.MCASTGB bit, when set, masks the interrupt when EMAC_TXMCASTFRM_GB counter reaches full or half.
10 (R/W)	UCASTGB	Tx Unicast Frames (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.UCASTGB bit, when set, masks the interrupt when EMAC_TXUCASTFRM_GB counter reaches full or half.
9 (R/W)	T1024TOMAX	Tx 1024-to-max Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.T1024TOMAX bit, when set, masks the interrupt when EMAC_TX1024TOMAX_GB counter reaches full or half.
8 (R/W)	T512TO1023	Tx 512-to-1023 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.T512TO1023 bit, when set, masks the interrupt when EMAC_TX512TO1023_GB counter reaches full or half.
7 (R/W)	T256TO511	Tx 256-to-511 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.T256TO511 bit, when set, masks the interrupt when EMAC_TX256TO511_GB counter reaches full or half.
6 (R/W)	T128TO255	Tx 128-to-255 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.T128TO255 bit, when set, masks the interrupt when EMAC_TX128TO255_GB counter reaches full or half.
5 (R/W)	T65TO127	Tx 65-to-127 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.T65TO127 bit, when set, masks the interrupt when EMAC_TX65TO127_GB counter reaches full or half.
4 (R/W)	T64	Tx 64 Octets (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.T64 bit, when set, masks the interrupt when EMAC_TX64_GB counter reaches full or half.
3 (R/W)	MCASTG	Tx Multicast Frames (Good) Count Half/Full Mask. The EMAC_MMC_TXIMSK.MCASTG bit, when set, masks the interrupt when EMAC_TXMCASTFRM_G counter reaches full or half.
2 (R/W)	BCASTG	Tx Broadcast Frames (Good) Count Half/Full Mask. The EMAC_MMC_TXIMSK.BCASTG bit, when set, masks the interrupt when EMAC_TXBCASTFRM_G counter reaches full or half.
1 (R/W)	FRCNTGB	Tx Frame Count (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.FRCNTGB bit, when set, masks the interrupt when EMAC_TXFRMCNT_GB counter reaches full or half.

Table 23-63: EMAC_MMC_TXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	OCTCNTGB	Tx Octet Count (Good/Bad) Count Half/Full Mask. The EMAC_MMC_TXIMSK.OCTCNTGB bit, when set, masks the interrupt when EMAC_TXOCTCNT_GB counter reaches full or half.

Tx OCT Count (Good/Bad) Register

The EMAC_TXOCTCNT_GB register contains the count of the number of bytes transmitted, exclusive of the preamble and retried bytes, in good and bad frames.

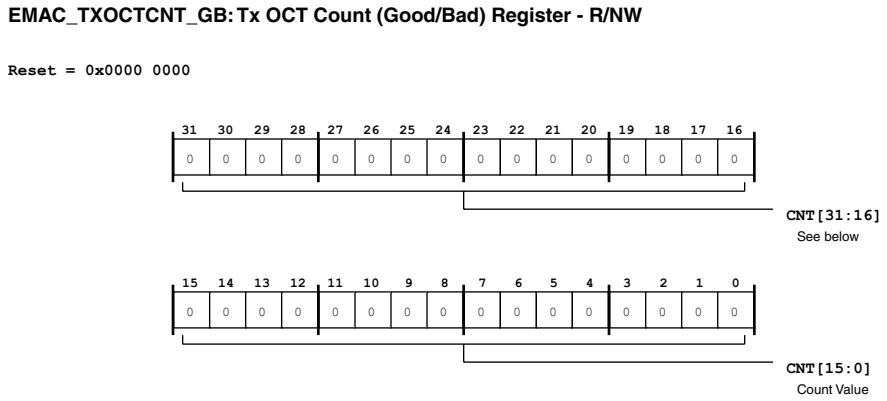


Figure 23-34: EMAC_TXOCTCNT_GB Register Diagram

Table 23-64: EMAC_TXOCTCNT_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Frame Count (Good/Bad) Register

The EMAC_TXFRMCNT_GB register contains the count of the number of good and bad frames transmitted, exclusive of retried frames.

EMAC_TXFRMCNT_GB: Tx Frame Count (Good/Bad) Register - R/NW

Reset = 0x0000 0000

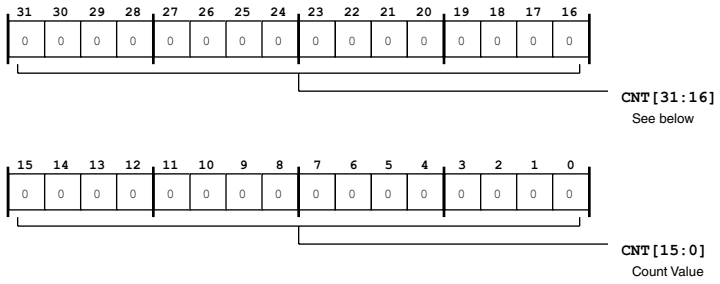


Figure 23-35: EMAC_TXFRMCNT_GB Register Diagram

Table 23-65: EMAC_TXFRMCNT_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Broadcast Frames (Good) Register

The EMAC_TXBCASTFRM_G register contains the count of the number of good broadcast frames transmitted.

EMAC_TXBCASTFRM_G: Tx Broadcast Frames (Good) Register - R/NW

Reset = 0x0000 0000

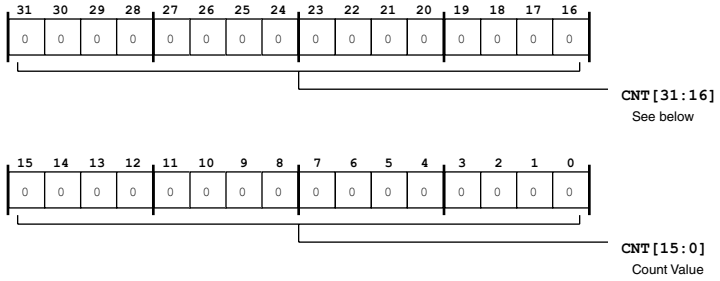


Figure 23-36: EMAC_TXBCASTFRM_G Register Diagram

Table 23-66: EMAC_TXBCASTFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Multicast Frames (Good) Register

The EMAC_TXMCASTFRM_G register contains the count of the number of good multicast frames transmitted.

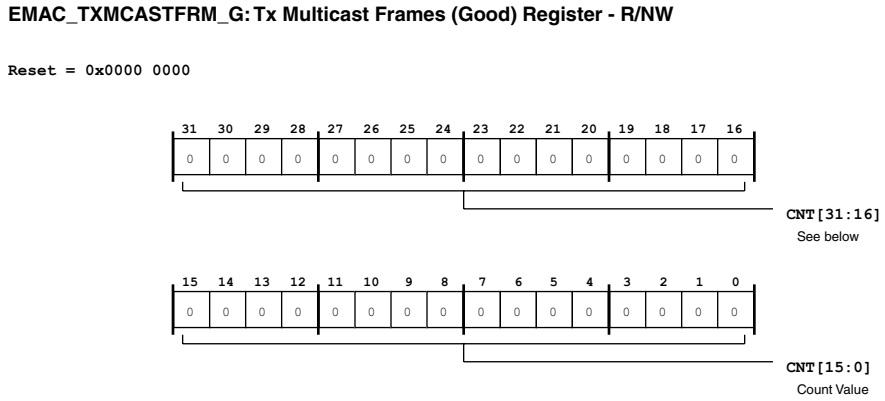


Figure 23-37: EMAC_TXMCASTFRM_G Register Diagram

Table 23-67: EMAC_TXMCASTFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx 64-Byte Frames (Good/Bad) Register

The EMAC_TX64_GB register contains the count of the number of good and bad frames transmitted with length 64 bytes, exclusive of preamble and retried frames.

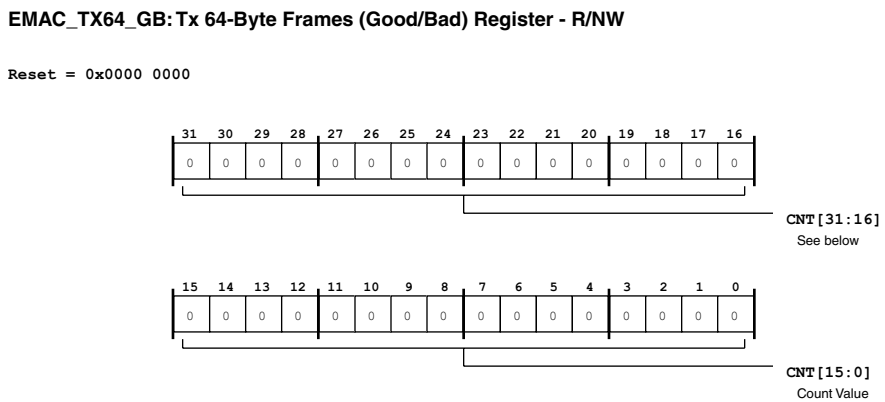


Figure 23-38: EMAC_TX64_GB Register Diagram

Table 23-68: EMAC_TX64_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx 65- to 127-Byte Frames (Good/Bad) Register

The EMAC_TX65TO127_GB register contains the count of the number of good and bad frames transmitted with length between 65 and 127 (inclusive) bytes, exclusive of preamble and retried frames.

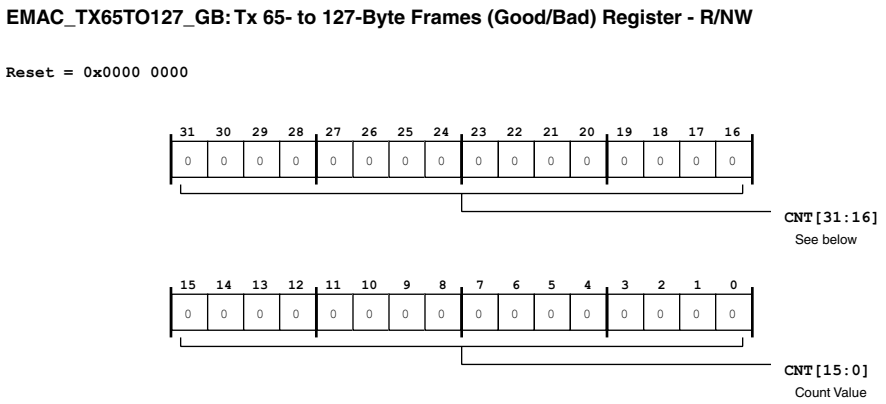


Figure 23-39: EMAC_TX65TO127_GB Register Diagram

Table 23-69: EMAC_TX65TO127_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx 128- to 255-Byte Frames (Good/Bad) Register

The EMAC_TX128TO255_GB register contains the count of the number of good and bad frames transmitted with length between 128 and 255 (inclusive) bytes, exclusive of preamble and retried frames.

EMAC_TX128TO255_GB: Tx 128- to 255-Byte Frames (Good/Bad) Register - R/NW

Reset = 0x0000 0000

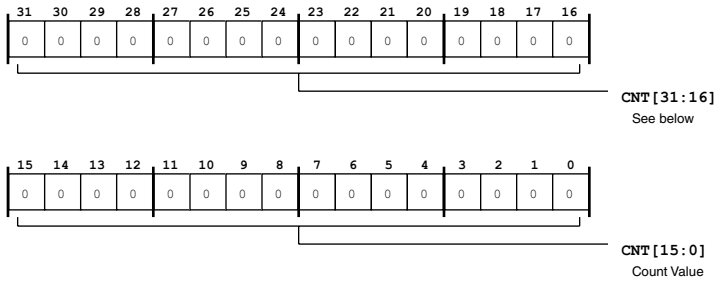


Figure 23-40: EMAC_TX128TO255_GB Register Diagram

Table 23-70: EMAC_TX128TO255_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx 256- to 511-Byte Frames (Good/Bad) Register

The EMAC_TX256TO511_GB register contains the count of the number of good and bad frames transmitted with length between 256 and 511 (inclusive) bytes, exclusive of preamble and retried frames.

EMAC_TX256TO511_GB: Tx 256- to 511-Byte Frames (Good/Bad) Register - R/NW

Reset = 0x0000 0000

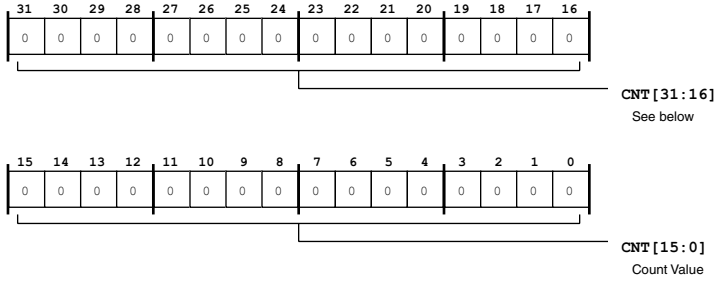


Figure 23-41: EMAC_TX256TO511_GB Register Diagram

Table 23-71: EMAC_TX256TO511_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx 512- to 1023-Byte Frames (Good/Bad) Register

The EMAC_TX512TO1023_GB register contains the count of the number of good and bad frames transmitted with length between 512 and 1023 (inclusive) bytes, exclusive of preamble and retried frames.

EMAC_TX512TO1023_GB: Tx 512- to 1023-Byte Frames (Good/Bad) Register - R/NW

Reset = 0x0000 0000

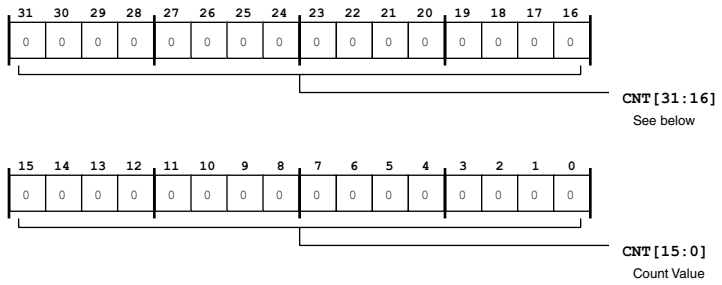


Figure 23-42: EMAC_TX512TO1023_GB Register Diagram

Table 23-72: EMAC_TX512TO1023_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx 1024- to Max-Byte Frames (Good/Bad) Register

The EMAC_TX1024TOMAX_GB register contains the count of the number of good and bad frames transmitted with length between 1024 and maxsize (inclusive) bytes, exclusive of preamble and retried frames.

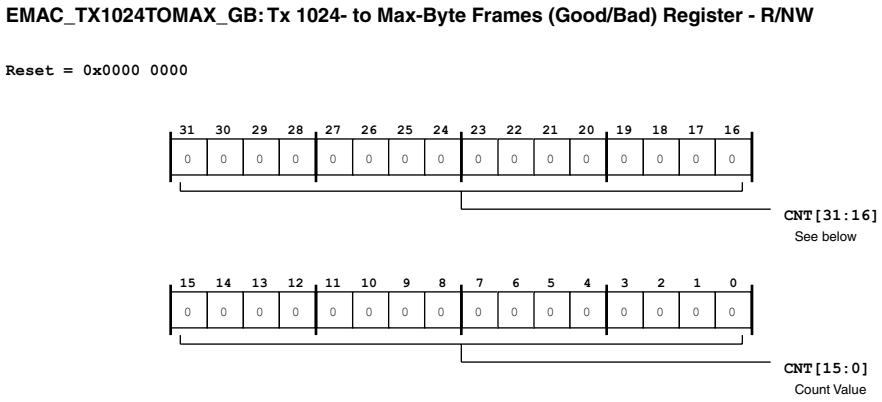


Figure 23-43: EMAC_TX1024TOMAX_GB Register Diagram

Table 23-73: EMAC_TX1024TOMAX_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Unicast Frames (Good/Bad) Register

The EMAC_TXUCASTFRM_GB register contains the count of the number of good and bad unicast frames transmitted.

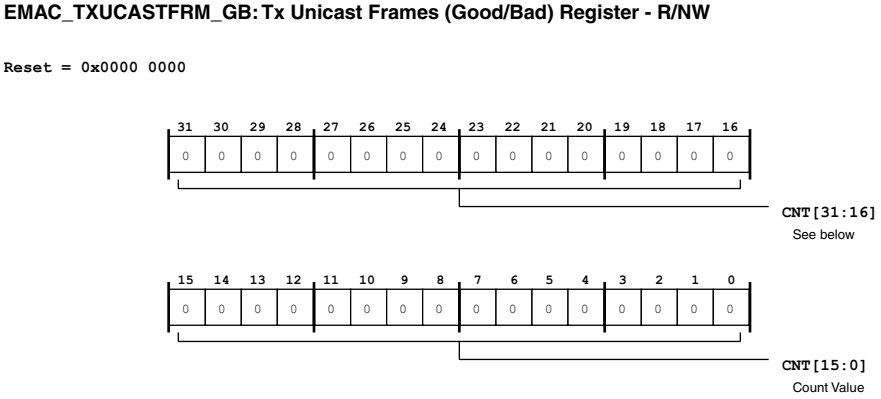


Figure 23-44: EMAC_TXUCASTFRM_GB Register Diagram

Table 23-74: EMAC_TXUCASTFRM_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Multicast Frames (Good/Bad) Register

The EMAC_TXMCASTFRM_GB register contains the count of the number of good and bad multicast frames transmitted.

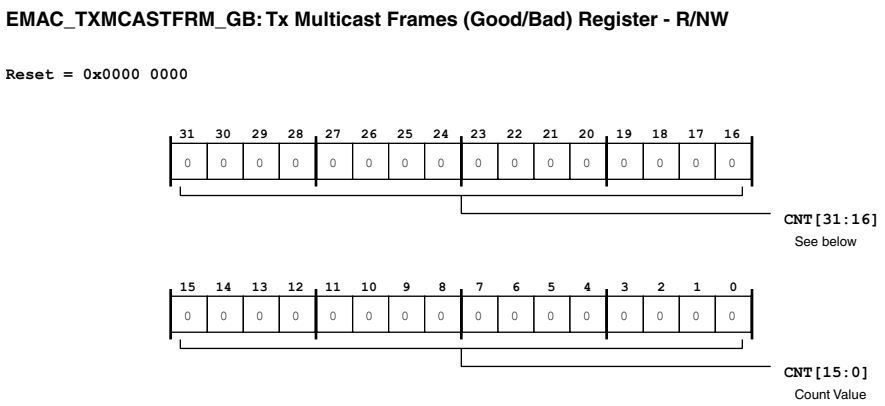


Figure 23-45: EMAC_TXMCASTFRM_GB Register Diagram

Table 23-75: EMAC_TXMCASTFRM_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Broadcast Frames (Good/Bad) Register

The EMAC_TXBCASTFRM_GB register contains the count of the number of good and bad broadcast frames transmitted.

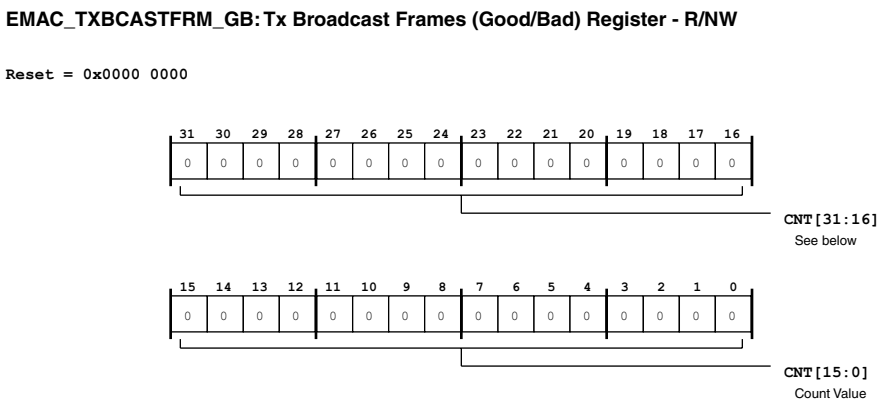


Figure 23-46: EMAC_TXBCASTFRM_GB Register Diagram

Table 23-76: EMAC_TXBCASTFRM_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Underflow Error Register

The EMAC_TXUNDR_ERR register contains a count of the number of frames aborted due to frame underflow error.

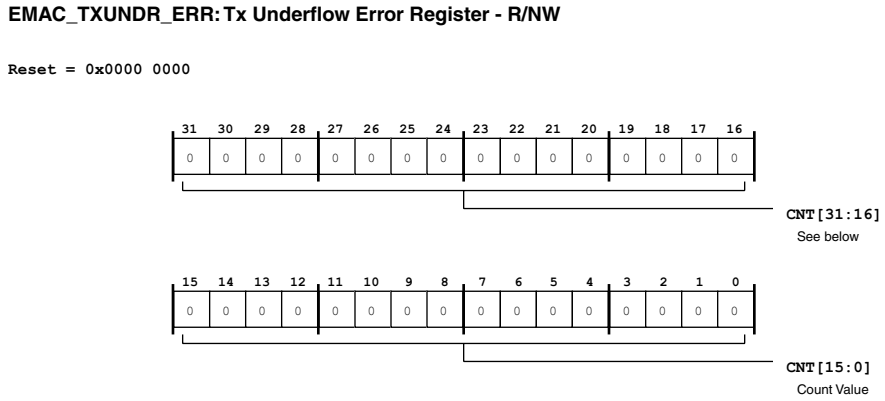


Figure 23-47: EMAC_TXUNDR_ERR Register Diagram

Table 23-77: EMAC_TXUNDR_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Single Collision (Good) Register

The EMAC_TXSNGCOL_G register contains a count of the number of successfully transmitted frames after a single collision in Half-duplex mode.

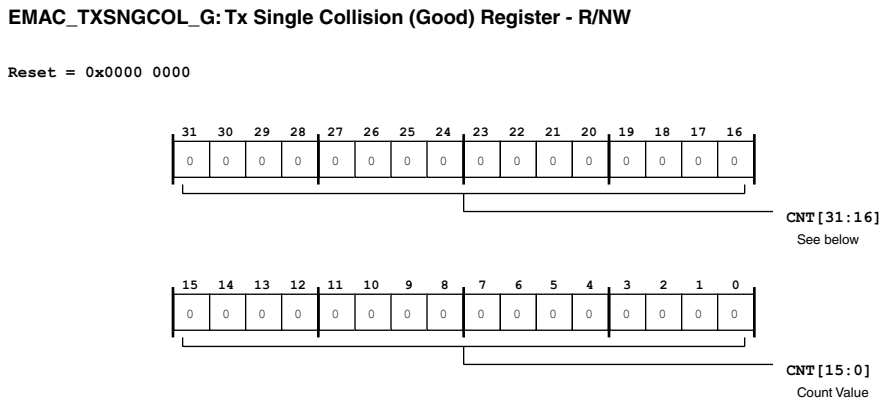


Figure 23-48: EMAC_TXSNGCOL_G Register Diagram

Table 23-78: EMAC_TXSNGCOL_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Multiple Collision (Good) Register

The EMAC_TXMULTCOL_G register contains a count of the number of successfully transmitted frames after more than a single collision in Half-duplex mode.

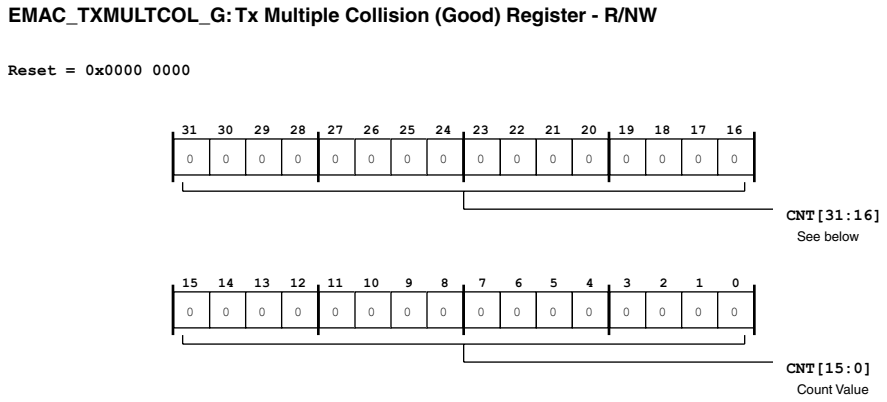


Figure 23-49: EMAC_TXMULTCOL_G Register Diagram

Table 23-79: EMAC_TXMULTCOL_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Deferred Register

The EMAC_TXDEFERRED register contains a count of the number of successfully transmitted frames after a deferral in Half-duplex mode.

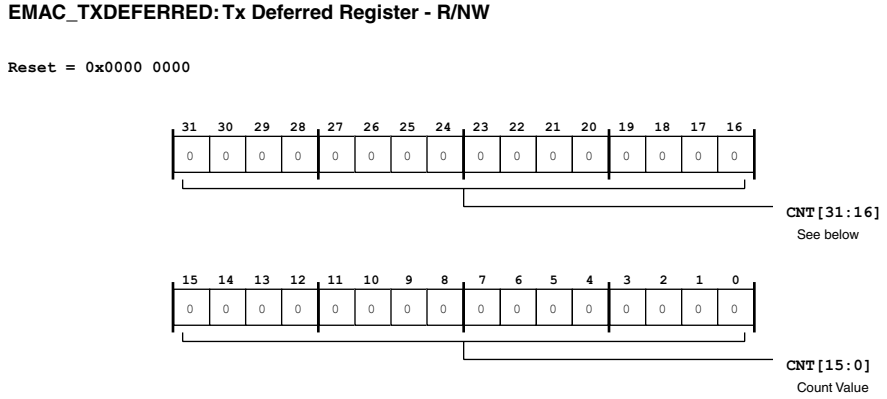


Figure 23-50: EMAC_TXDEFERRED Register Diagram

Table 23-80: EMAC_TXDEFERRED Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Late Collision Register

The EMAC_TXLATECOL register contains a count of the number of frames aborted due to late collision error.

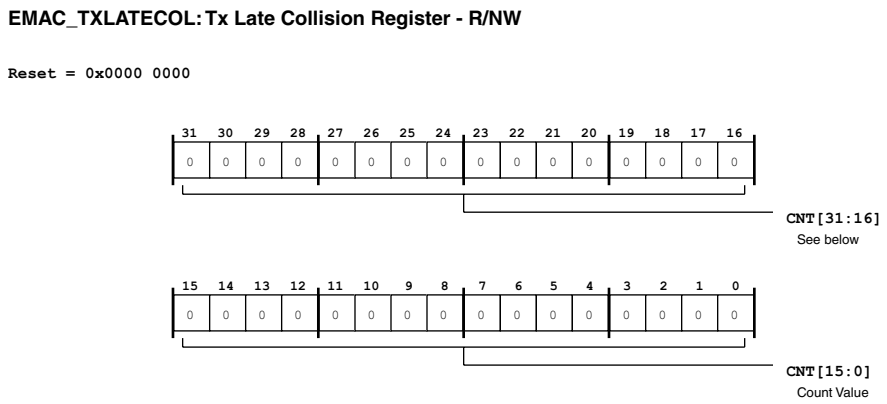


Figure 23-51: EMAC_TXLATECOL Register Diagram

Table 23-81: EMAC_TXLATECOL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Excess Collision Register

The EMAC_TXEXCESSCOL register contains a count of the number of frames aborted due to excessive (16) collision errors.

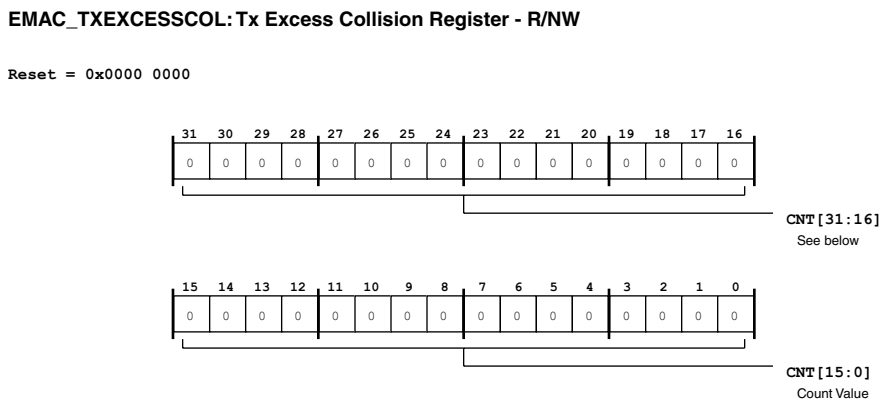


Figure 23-52: EMAC_TXEXCESSCOL Register Diagram

Table 23-82: EMAC_TXEXCESSCOL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Carrier Error Register

The EMAC_TXCARR_ERR register contains a count of the number of frames aborted due to carrier sense error (no carrier or loss of carrier).

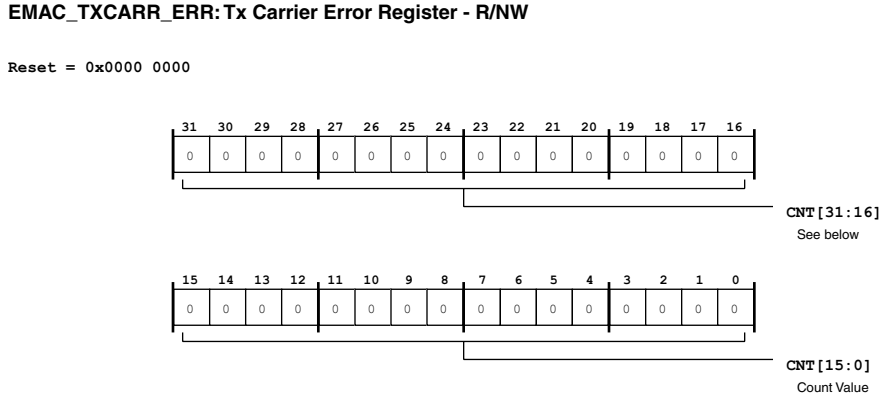


Figure 23-53: EMAC_TXCARR_ERR Register Diagram

Table 23-83: EMAC_TXCARR_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Octet Count (Good) Register

The EMAC_TXOCTCNT_G register contains a count of the number of bytes transmitted, exclusive of preamble, in good frames only.

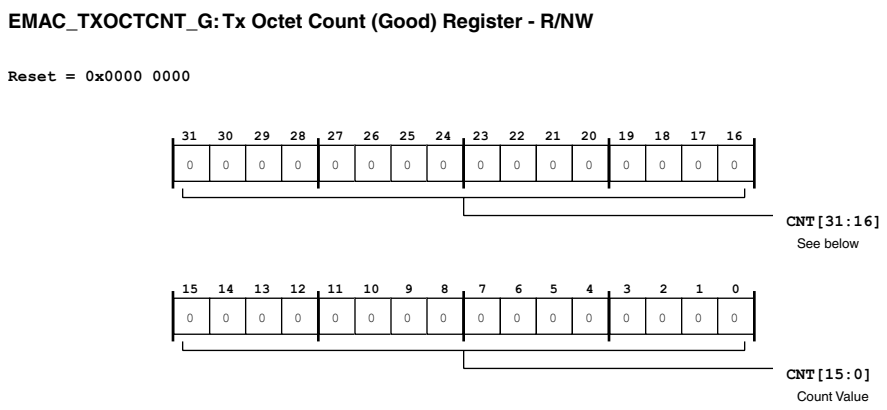


Figure 23-54: EMAC_TXOCTCNT_G Register Diagram

Table 23-84: EMAC_TXOCTCNT_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Frame Count (Good) Register

The EMAC_TXFRMCNT_G register contains a count of the number of good frames transmitted.

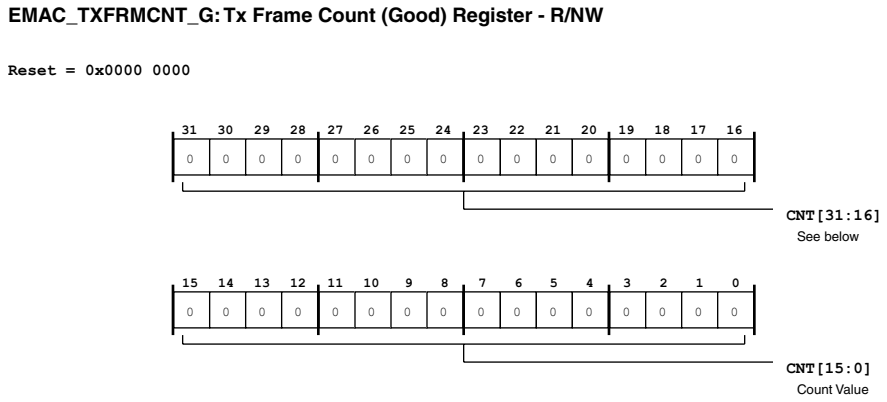


Figure 23-55: EMAC_TXFRMCNT_G Register Diagram

Table 23-85: EMAC_TXFRMCNT_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Excess Deferral Register

The EMAC_TXEXCESSDEF register contains a count of the number of frames aborted due to excessive deferral error (deferred for more than two max-sized frame times).

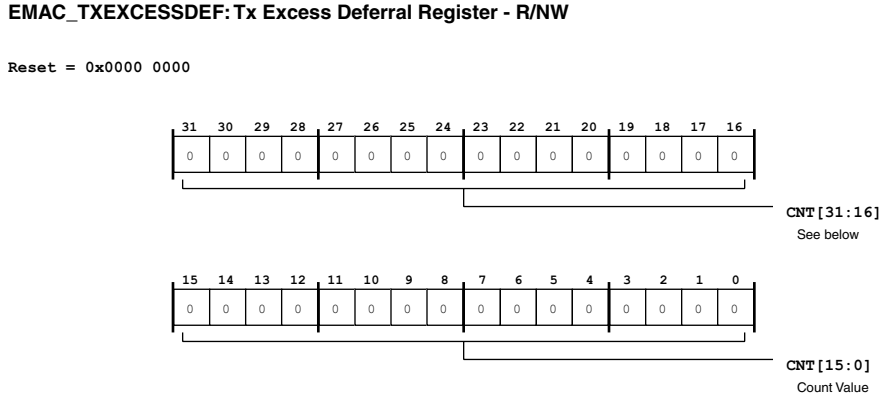


Figure 23-56: EMAC_TXEXCESSDEF Register Diagram

Table 23-86: EMAC_TXEXCESSDEF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx Pause Frame Register

The EMAC_TXPAUSEFRM register contains a count of the number of good PAUSE frames transmitted.

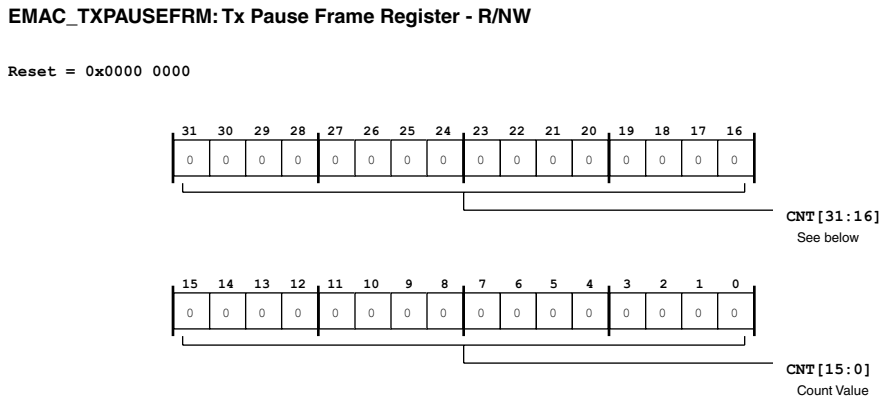


Figure 23-57: EMAC_TXPAUSEFRM Register Diagram

Table 23-87: EMAC_TXPAUSEFRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Tx VLAN Frames (Good) Register

The EMAC_TXVLANFRM_G register contains a count of the number of good VLAN frames transmitted, exclusive of retried frames.

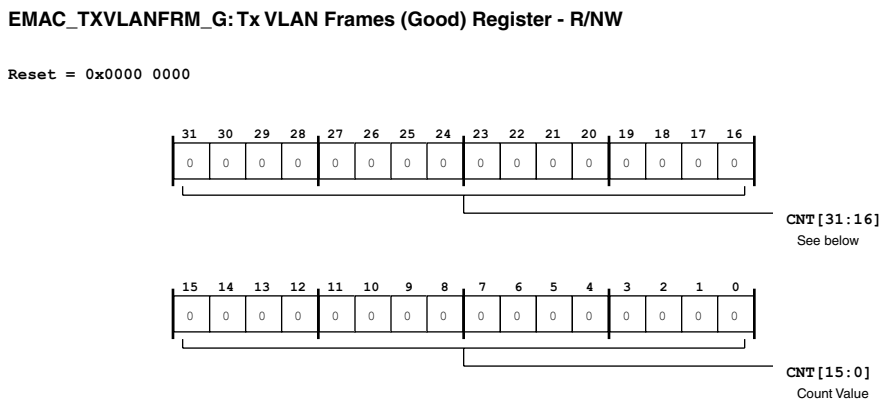


Figure 23-58: EMAC_TXVLANFRM_G Register Diagram

Table 23-88: EMAC_TXVLANFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Frame Count (Good/Bad) Register

The EMAC_RXFRMCNT_GB register contains a count of the number of good and bad frames received.

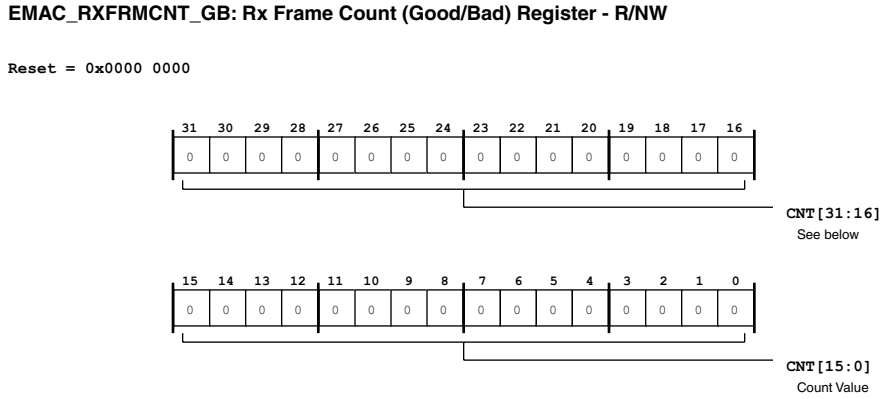


Figure 23-59: EMAC_RXFRMCNT_GB Register Diagram

Table 23-89: EMAC_RXFRMCNT_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Octet Count (Good/Bad) Register

The EMAC_RXOCTCNT_GB register contains a count of the number of bytes received, exclusive of preamble, in good and bad frames.

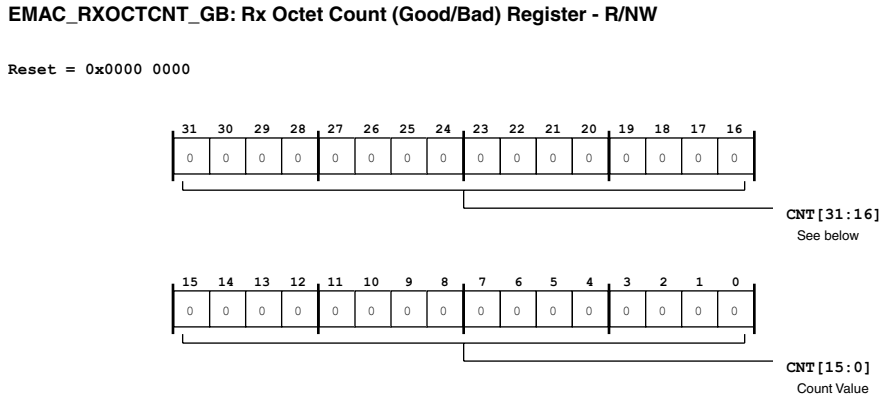


Figure 23-60: EMAC_RXOCTCNT_GB Register Diagram

Table 23-90: EMAC_RXOCTCNT_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Octet Count (Good) Register

The EMAC_RXOCTCNT_G register contains a count of the number of bytes received, exclusive of preamble, only in good frames.

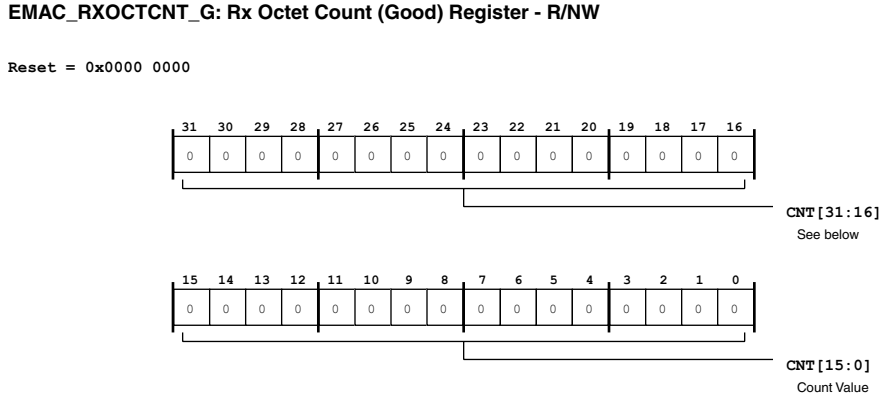


Figure 23-61: EMAC_RXOCTCNT_G Register Diagram

Table 23-91: EMAC_RXOCTCNT_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Broadcast Frames (Good) Register

The EMAC_RXBCASTFRM_G register contains a count of the number of good broadcast frames received.

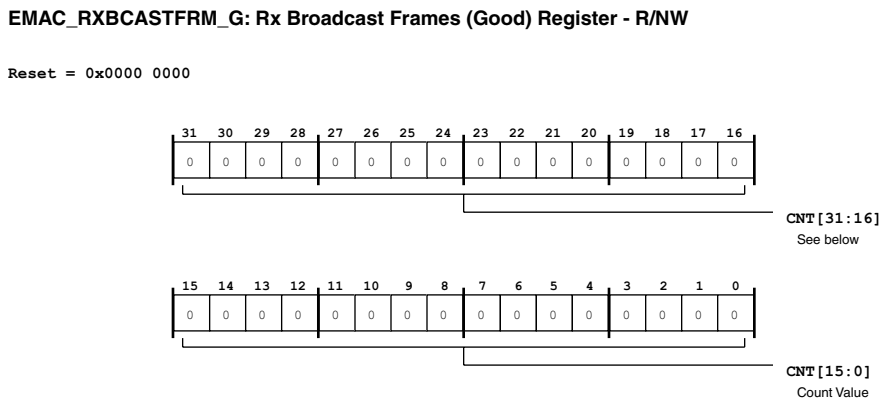


Figure 23-62: EMAC_RXBCASTFRM_G Register Diagram

Table 23-92: EMAC_RXBCASTFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Multicast Frames (Good) Register

The EMAC_RXMCASTFRM_G register contains a count of the number of good multicast frames received.

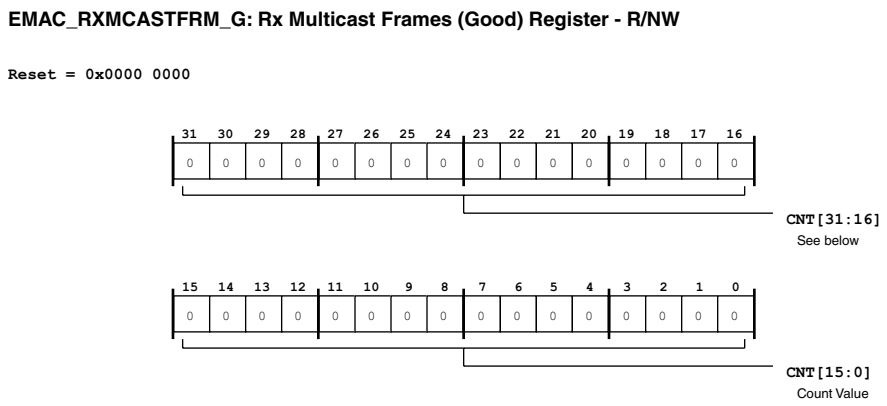


Figure 23-63: EMAC_RXMCASTFRM_G Register Diagram

Table 23-93: EMAC_RXMCASTFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx CRC Error Register

The EMAC_RXCRC_ERR register contains a count of the number of frames received with CRC error.

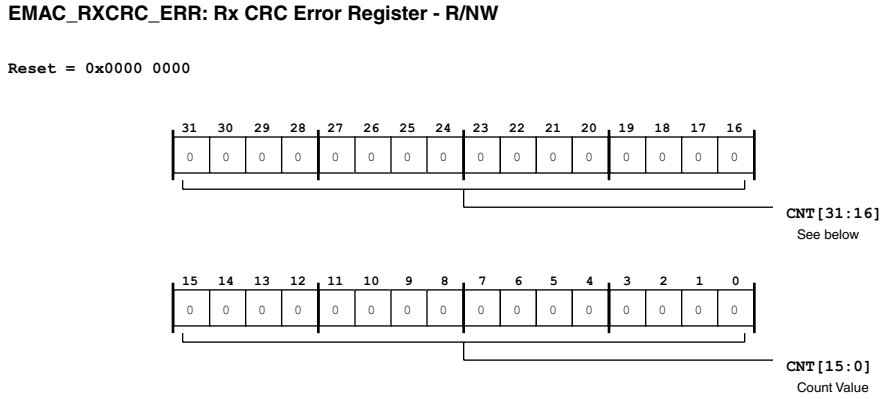


Figure 23-64: EMAC_RXCRC_ERR Register Diagram

Table 23-94: EMAC_RXCRC_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx alignment Error Register

The EMAC_RXALIGN_ERR register contains a count of the number of frames received with alignment (dribble) error. Valid only in 10/100 mode.

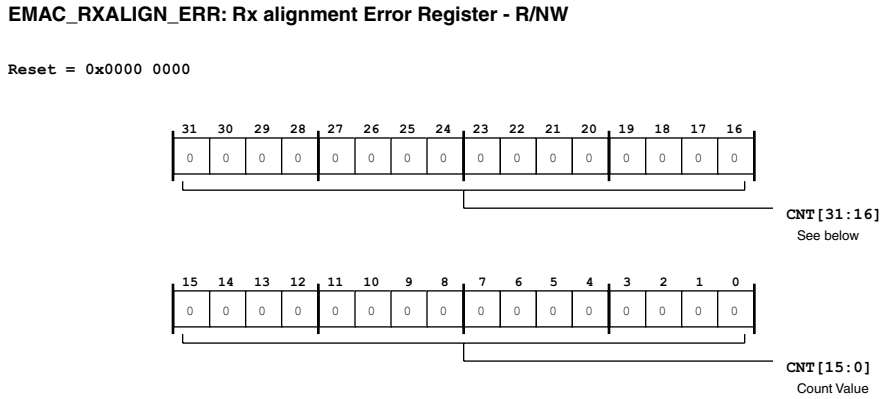


Figure 23-65: EMAC_RXALIGN_ERR Register Diagram

Table 23-95: EMAC_RXALIGN_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Runt Error Register

The EMAC_RXRUNT_ERR register contains a count of the number of frames received with runt error.

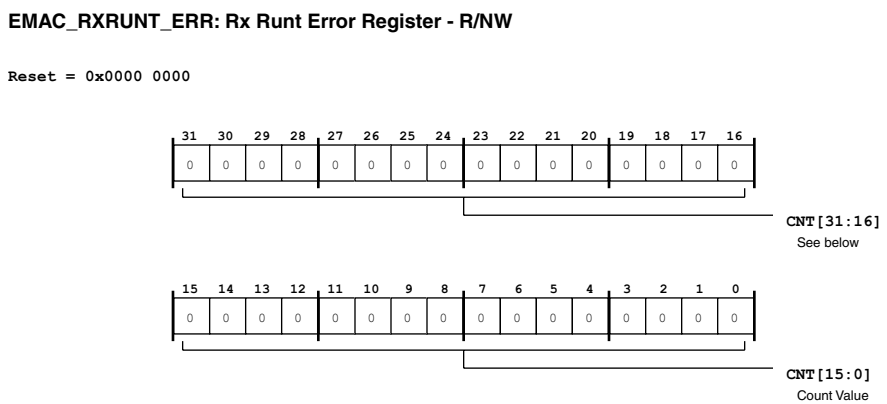


Figure 23-66: EMAC_RXRUNT_ERR Register Diagram

Table 23-96: EMAC_RXRUNT_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Jab Error Register

The EMAC_RXJAB_ERR register contains a count of the number of giant frames received with length greater than 1,518 bytes and with CRC error.

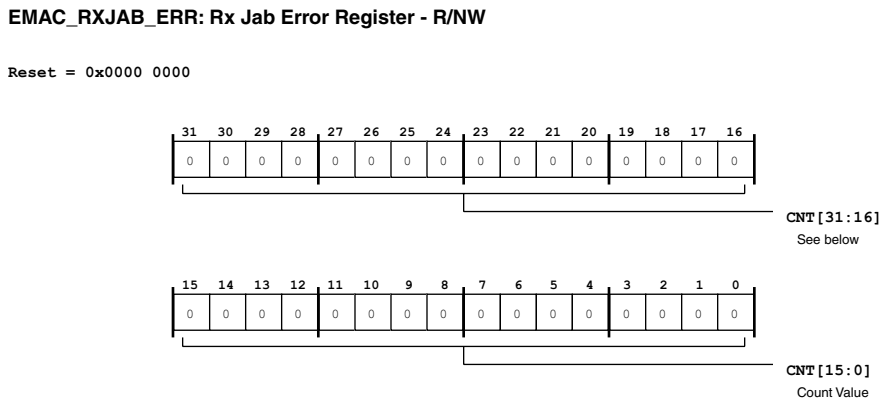


Figure 23-67: EMAC_RXJAB_ERR Register Diagram

Table 23-97: EMAC_RXJAB_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Undersize (Good) Register

The EMAC_RXUSIZE_G register contains a count of the number of frames received with length less than 64 bytes, without any errors.

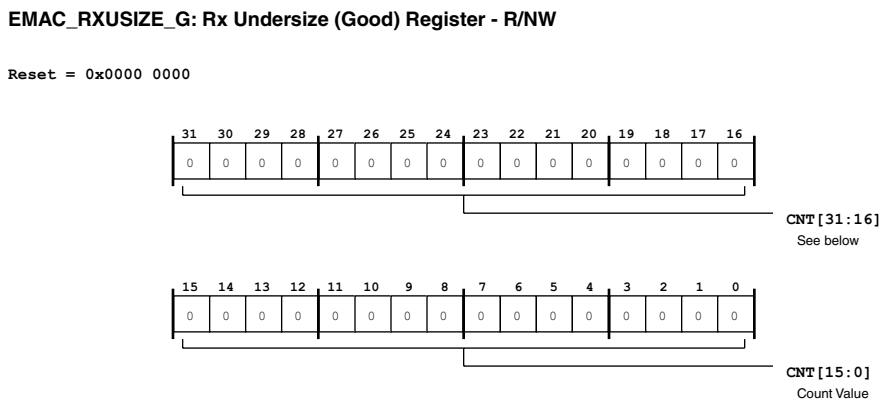


Figure 23-68: EMAC_RXUSIZE_G Register Diagram

Table 23-98: EMAC_RXUSIZE_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Oversize (Good) Register

The EMAC_RXOSIZE_G register contains a count of the number of frames received with length greater than the maxsize, without errors.

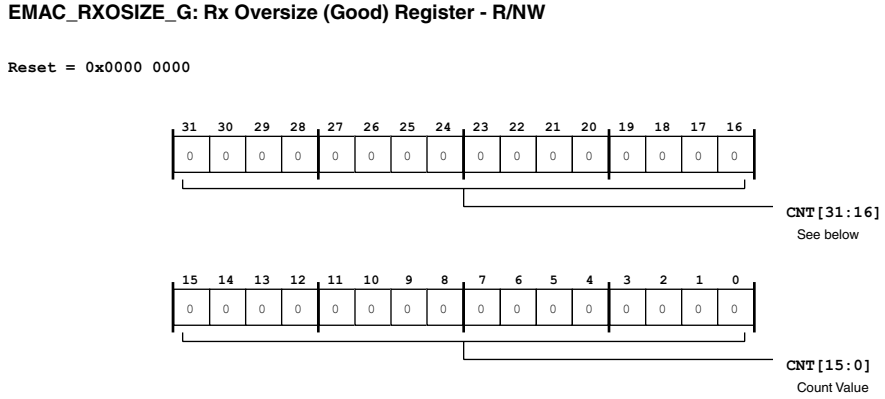


Figure 23-69: EMAC_RXOSIZE_G Register Diagram

Table 23-99: EMAC_RXOSIZE_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx 64-Byte Frames (Good/Bad) Register

The EMAC_RX64_GB register contains a count of the number of good and bad frames received with length 64 bytes, exclusive of preamble.

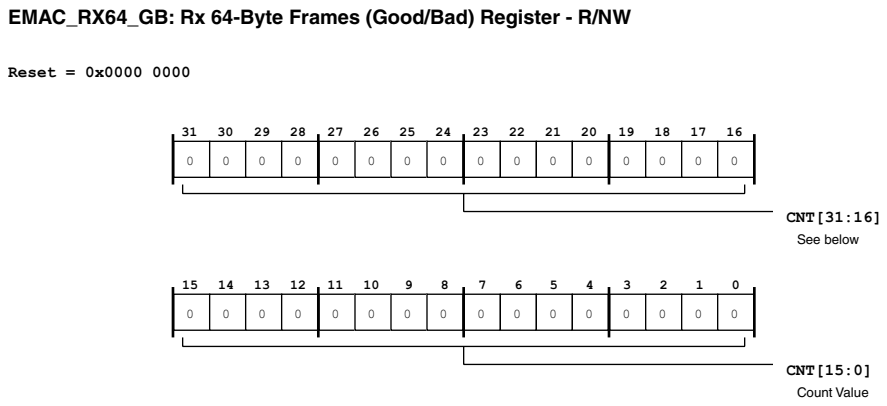


Figure 23-70: EMAC_RX64_GB Register Diagram

Table 23-100: EMAC_RX64_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx 65- to 127-Byte Frames (Good/Bad) Register

The EMAC_RX65TO127_GB register contains a count of the number of good and bad frames received with length between 65 and 127 (inclusive) bytes, exclusive of preamble.

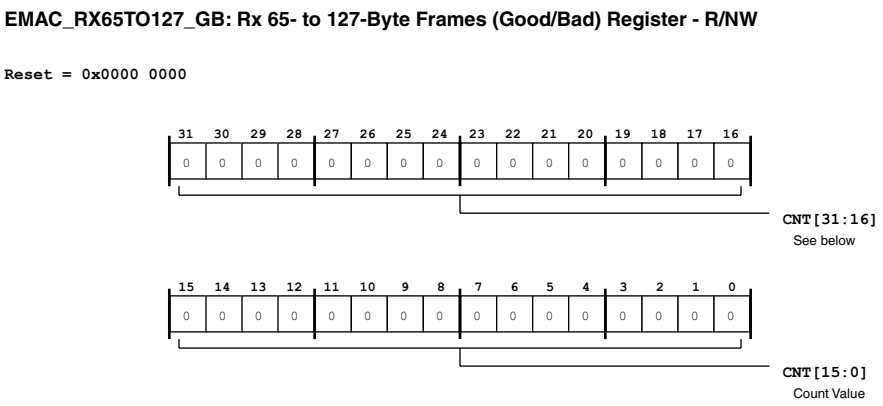


Figure 23-71: EMAC_RX65TO127_GB Register Diagram

Table 23-101: EMAC_RX65TO127_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx 128- to 255-Byte Frames (Good/Bad) Register

The EMAC_RX128TO255_GB register contains a count of the number of good and bad frames received with length between 128 and 255 (inclusive) bytes, exclusive of preamble.

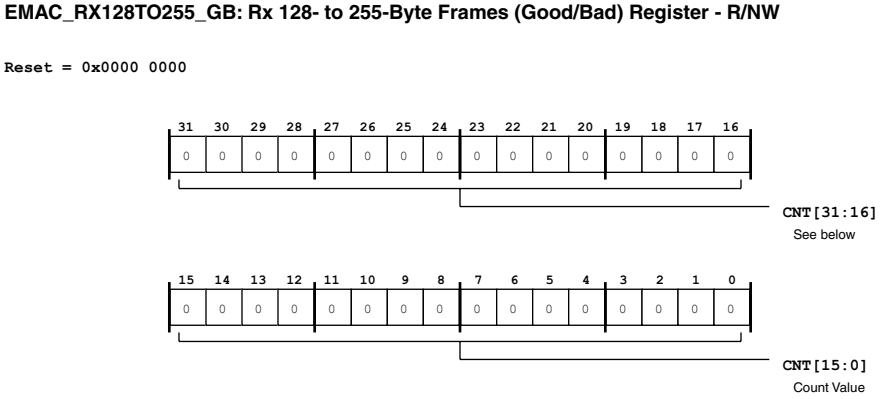


Figure 23-72: EMAC_RX128TO255_GB Register Diagram

Table 23-102: EMAC_RX128TO255_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx 256- to 511-Byte Frames (Good/Bad) Register

The EMAC_RX256TO511_GB register contains a count of the number of good and bad frames received with length between 256 and 511 (inclusive) bytes, exclusive of preamble.

EMAC_RX256TO511_GB: Rx 256- to 511-Byte Frames (Good/Bad) Register - R/NW

Reset = 0x0000 0000

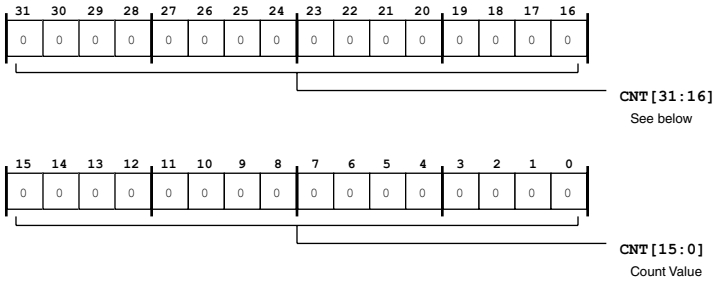


Figure 23-73: EMAC_RX256TO511_GB Register Diagram

Table 23-103: EMAC_RX256TO511_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx 512- to 1023-Byte Frames (Good/Bad) Register

The EMAC_RX512TO1023_GB register contains a count of the number of good and bad frames received with length between 512 and 1023 (inclusive) bytes, exclusive of preamble.

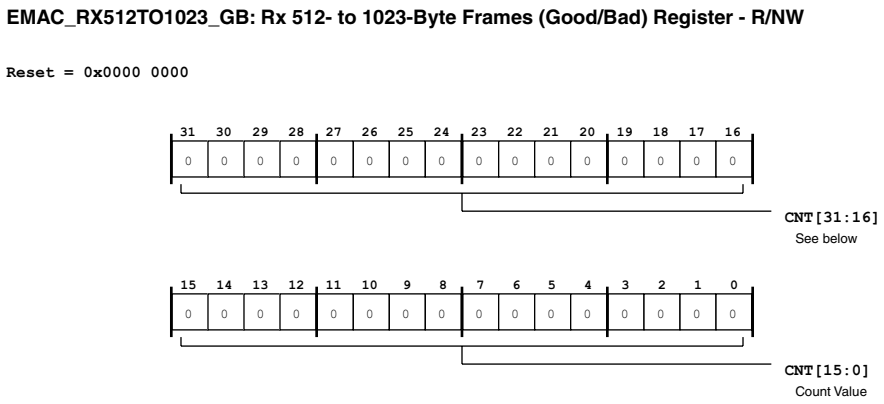


Figure 23-74: EMAC_RX512TO1023_GB Register Diagram

Table 23-104: EMAC_RX512TO1023_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx 1024- to Max-Byte Frames (Good/Bad) Register

The EMAC_RX1024TOMAX_GB register contains a count of the number of good and bad frames received with length between 1024 and maxsize (inclusive) bytes, exclusive of preamble.

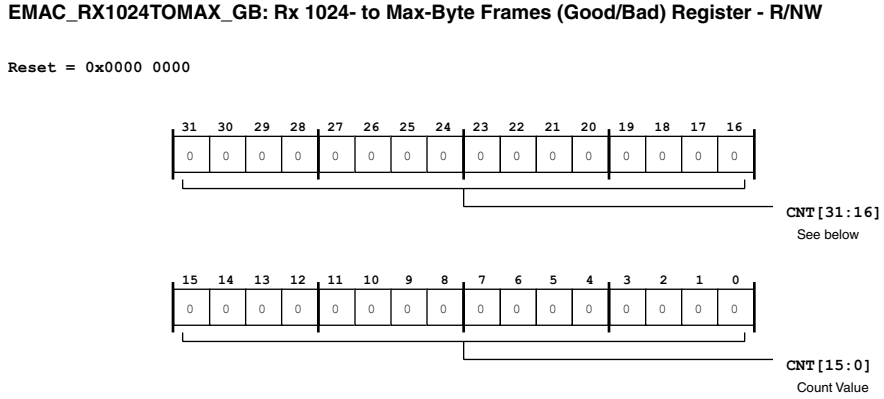


Figure 23-75: EMAC_RX1024TOMAX_GB Register Diagram

Table 23-105: EMAC_RX1024TOMAX_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Unicast Frames (Good) Register

The EMAC_RXUCASTFRM_G register contains a count of the number of good unicast frames received.

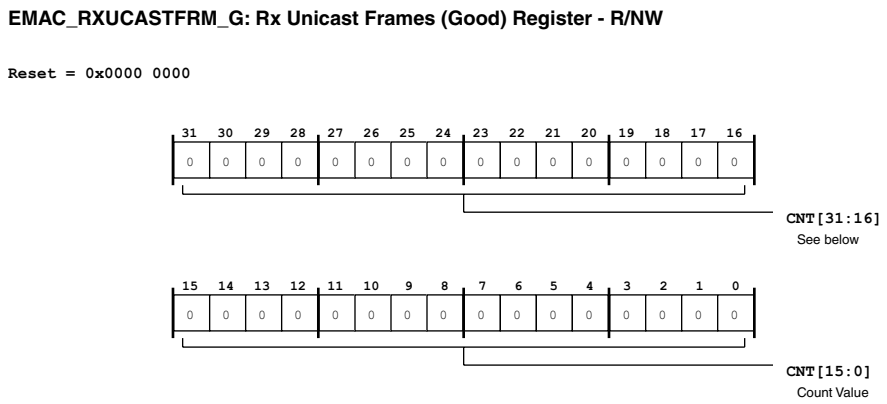


Figure 23-76: EMAC_RXUCASTFRM_G Register Diagram

Table 23-106: EMAC_RXUCASTFRM_G Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Length Error Register

The EMAC_RXLEN_ERR register contains a count of the number of frames received with length error (Length type field frame size), for all frames with valid length field.

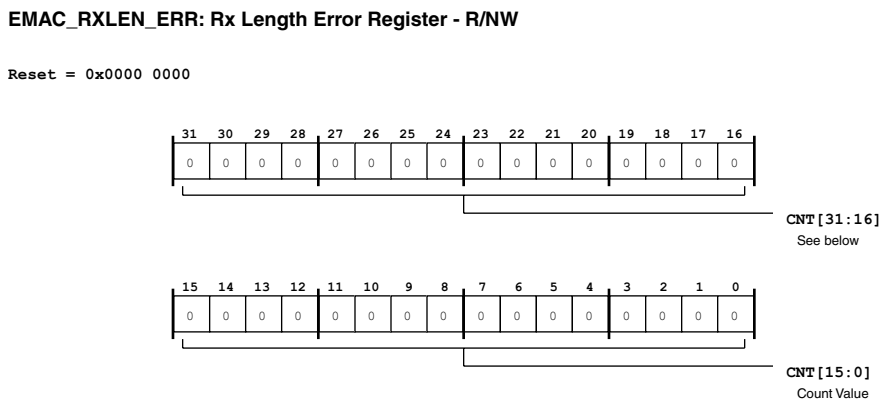


Figure 23-77: EMAC_RXLEN_ERR Register Diagram

Table 23-107: EMAC_RXLEN_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Out Of Range Type Register

The EMAC_RXOORTYPE register contains a count of the number of frames received with length field not equal to the valid frame size (greater than 1,500 but less than 1,536).

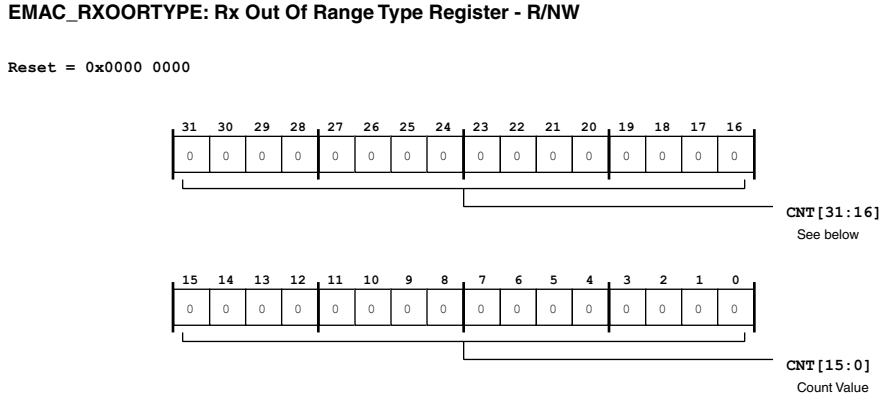


Figure 23-78: EMAC_RXOORTYPE Register Diagram

Table 23-108: EMAC_RXOORTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Pause Frames Register

The EMAC_RXPAUSEFRM register contains a count of the number of good and valid PAUSE frames received.

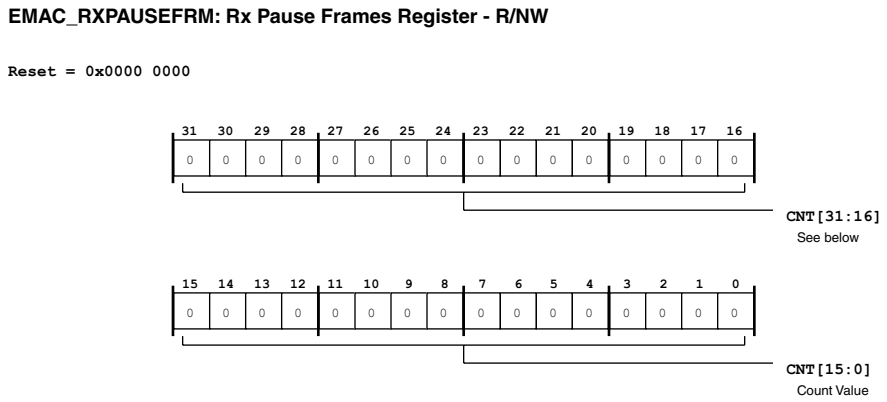


Figure 23-79: EMAC_RXPAUSEFRM Register Diagram

Table 23-109: EMAC_RXPAUSEFRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx FIFO Overflow Register

The EMAC_RXFIFO_OVF register contains a count of the number of missed received frames due to FIFO overflow.

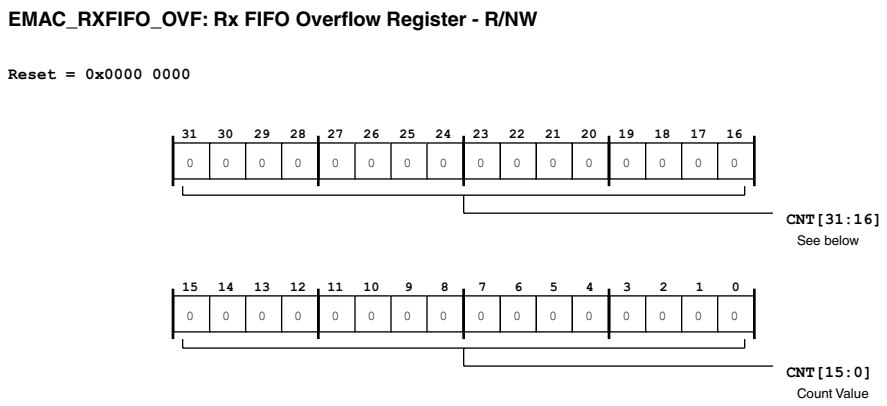


Figure 23-80: EMAC_RXFIFO_OVF Register Diagram

Table 23-110: EMAC_RXFIFO_OVF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx VLAN Frames (Good/Bad) Register

The EMAC_RXVLANFRM_GB register contains a count of the number of good and bad VLAN frames received.

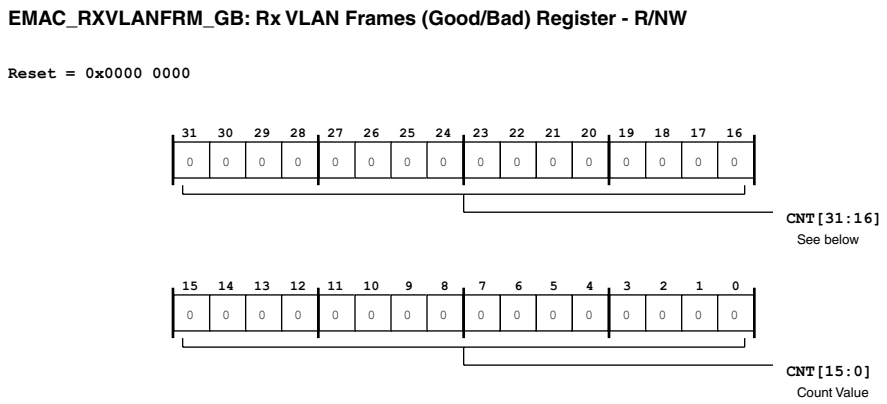


Figure 23-81: EMAC_RXVLANFRM_GB Register Diagram

Table 23-111: EMAC_RXVLANFRM_GB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx Watch Dog Error Register

The EMAC_RXWDOG_ERR register contains a count of the number of frames received with error due to watchdog timeout error (frames with a data load larger than 2,048 bytes).

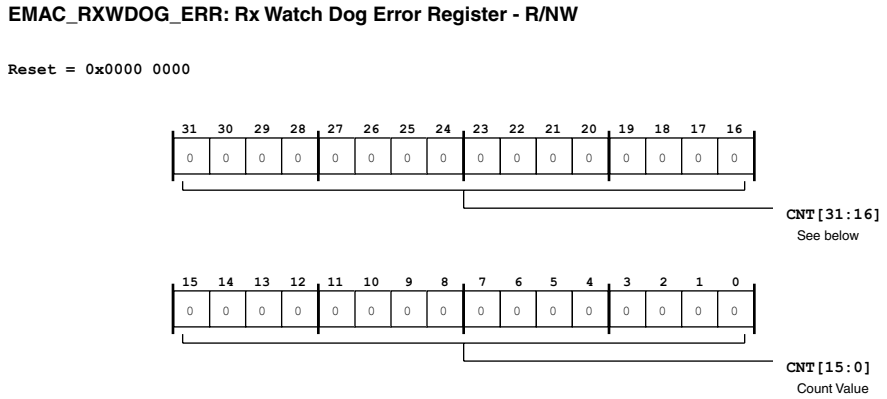


Figure 23-82: EMAC_RXWDOG_ERR Register Diagram

Table 23-112: EMAC_RXWDOG_ERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

MMC IPC Rx Interrupt Mask Register

The EMAC_IPC_RXIMSK register enables (unmasks) MMC IPC receive interrupts.

EMAC_IPC_RXIMSK: MMC IPC Rx Interrupt Mask Register - R/W

Reset = 0x0000 0000

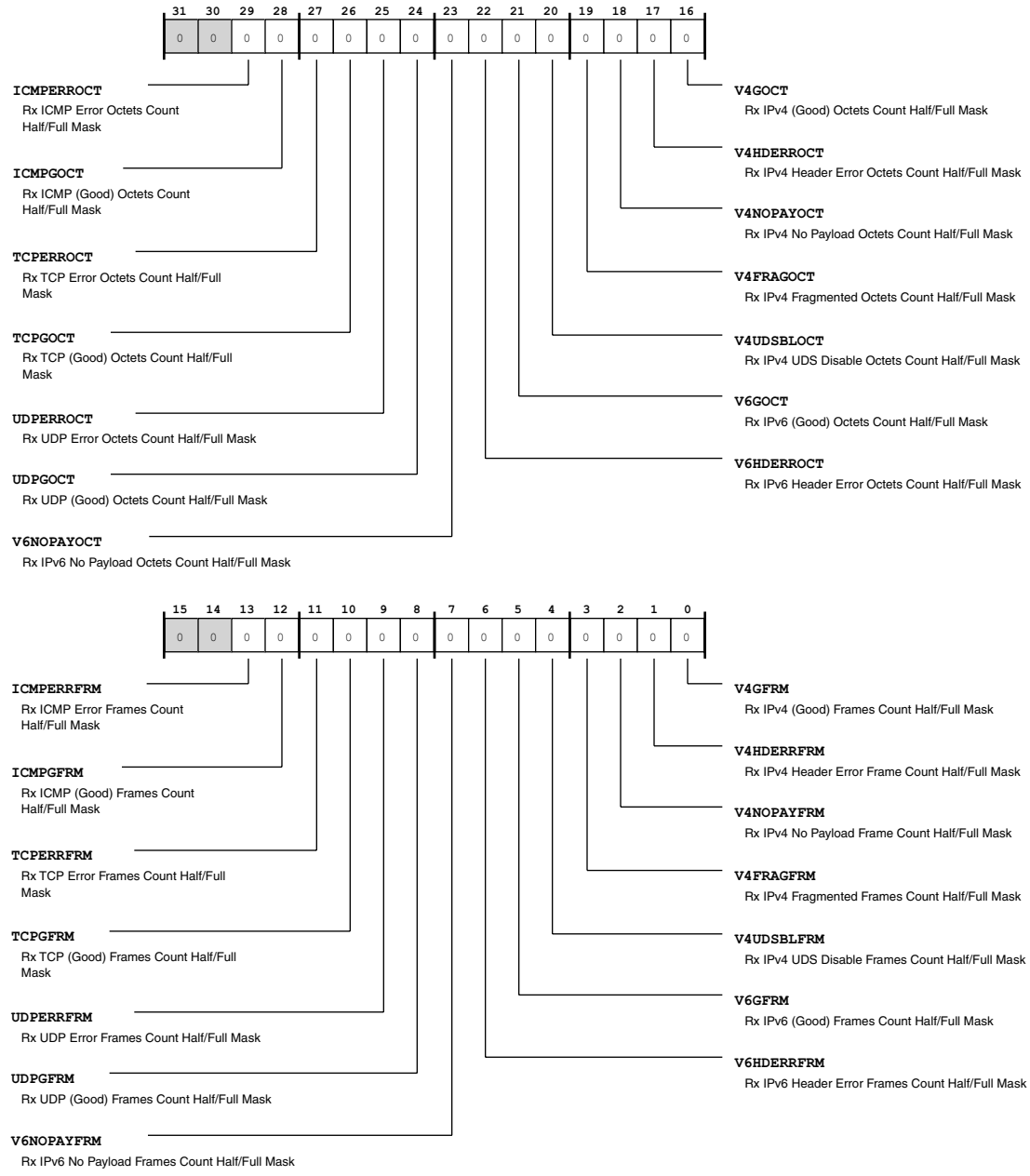


Figure 23-83: EMAC_IPC_RXIMSK Register Diagram

Table 23-113: EMAC_IPC_RXIMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/W)	ICMPERROCT	Rx ICMP Error Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.ICMPERROCT bit, when set, masks the interrupt when the EMAC_RXICMP_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
28 (R/W)	ICMPGOCT	Rx ICMP (Good) Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.ICMPGOCT bit, when set, masks the interrupt when the EMAC_RXICMP_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
27 (R/W)	TCPERROCT	Rx TCP Error Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.TCPERROCT bit, when set, masks the interrupt when the EMAC_RXTCP_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
26 (R/W)	TCPGOCT	Rx TCP (Good) Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.TCPGOCT bit, when set, masks the interrupt when the EMAC_RXTCP_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
25 (R/W)	UDPERROCT	Rx UDP Error Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.UDPERROCT bit, when set, masks the interrupt when the EMAC_RXUDP_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
24 (R/W)	UDPGOCT	Rx UDP (Good) Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.UDPGOCT bit, when set, masks the interrupt when the EMAC_RXUDP_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
23 (R/W)	V6NOPAYOCT	Rx IPv6 No Payload Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.V6NOPAYOCT bit, when set, masks the interrupt when the EMAC_RXIPV6_NOPAY_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
22 (R/W)	V6HDERROCT	Rx IPv6 Header Error Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.V6HDERROCT bit, when set, masks the interrupt when the EMAC_RXIPV6_HDR_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
21 (R/W)	V6GOCT	Rx IPv6 (Good) Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.V6GOCT bit, when set, masks the interrupt when the EMAC_RXIPV6_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.

Table 23-113: EMAC_IPC_RXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
20 (R/W)	V4UDSBLOCT	Rx IPv4 UDS Disable Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4UDSBLOCT bit, when set, masks the interrupt when the EMAC_RXIPV4_UDSBL_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
19 (R/W)	V4FRAGOCT	Rx IPv4 Fragmented Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4FRAGOCT bit, when set, masks the interrupt when the EMAC_RXIPV4_FRAG_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
18 (R/W)	V4NOPAYOCT	Rx IPv4 No Payload Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4NOPAYOCT bit, when set, masks the interrupt when the EMAC_RXIPV4_NOPAY_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
17 (R/W)	V4HDERROCT	Rx IPv4 Header Error Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4HDERROCT bit, when set, masks the interrupt when the EMAC_RXIPV4_HDR_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
16 (R/W)	V4GOCT	Rx IPv4 (Good) Octets Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4GOCT bit, when set, masks the interrupt when the EMAC_RXIPV4_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
13 (R/W)	ICMPERRFRM	Rx ICMP Error Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.ICMPERRFRM bit, when set, masks the interrupt when the EMAC_RXICMP_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
12 (R/W)	ICMPGFRM	Rx ICMP (Good) Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.ICMPGFRM bit, when set, masks the interrupt when the EMAC_RXICMP_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
11 (R/W)	TCPERRFRM	Rx TCP Error Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.TCPERRFRM bit, when set, masks the interrupt when the EMAC_RXTCP_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
10 (R/W)	TCPGFRM	Rx TCP (Good) Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.TCPGFRM bit, when set, masks the interrupt when the EMAC_RXTCP_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.

Table 23-113: EMAC_IPC_RXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W)	UDPERRFRM	Rx UDP Error Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.UDPERRFRM bit, when set, masks the interrupt when the EMAC_RXUDP_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
8 (R/W)	UDPGFRM	Rx UDP (Good) Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.UDPGFRM bit, when set, masks the interrupt when the EMAC_RXUDP_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
7 (R/W)	V6NOPAYFRM	Rx IPv6 No Payload Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.V6NOPAYFRM bit, when set, masks the interrupt when the EMAC_RXIPV6_NOPAY_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
6 (R/W)	V6HDERRFRM	Rx IPv6 Header Error Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.V6HDERRFRM bit, when set, masks the interrupt when the EMAC_RXIPV6_HDR_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
5 (R/W)	V6GFRM	Rx IPv6 (Good) Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.V6GFRM bit, when set, masks the interrupt when the EMAC_RXIPV6_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
4 (R/W)	V4UDSBLFRM	Rx IPv4 UDS Disable Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4UDSBLFRM bit, when set, masks the interrupt when the EMAC_RXIPV4_UDSBL_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
3 (R/W)	V4FRAGFRM	Rx IPv4 Fragmented Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4FRAGFRM bit, when set, masks the interrupt when the EMAC_RXIPV4_FRAG_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
2 (R/W)	V4NOPAYFRM	Rx IPv4 No Payload Frame Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4NOPAYFRM bit, when set, masks the interrupt when the EMAC_RXIPV4_NOPAY_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
1 (R/W)	V4HDERRFRM	Rx IPv4 Header Error Frame Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4HDERRFRM bit, when set, masks the interrupt when the EMAC_RXIPV4_HDR_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.

Table 23-113: EMAC_IPC_RXIMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	V4GFRM	Rx IPv4 (Good) Frames Count Half/Full Mask. The EMAC_IPC_RXIMSK.V4GFRM bit, when set, masks the interrupt when the EMAC_RXIPV4_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.

MMC IPC Rx Interrupt Register

The EMAC_IPC_RXINT register indicates status of MMC IPC receive interrupts.

EMAC_IPC_RXINT: MMC IPC Rx Interrupt Register - R/W

Reset = 0x0000 0000

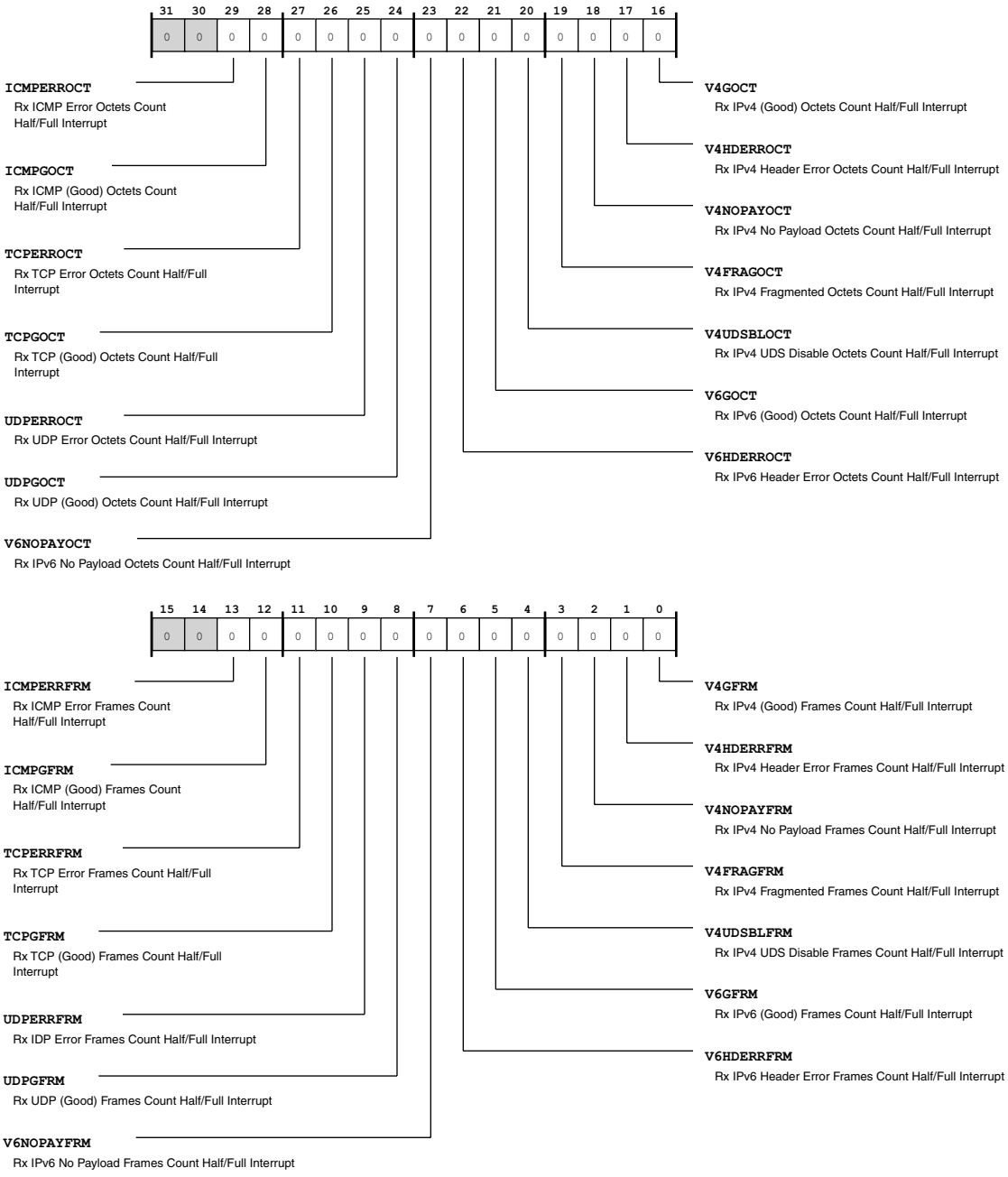


Figure 23-84: EMAC_IPC_RXINT Register Diagram

Table 23-114: EMAC_IPC_RXINT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/NW)	ICMPERROCT	Rx ICMP Error Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.ICMPERROCT bit is set when the EMAC_RXICMP_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
28 (R/NW)	ICMPGOCT	Rx ICMP (Good) Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.ICMPGOCT bit is set when the EMAC_RXICMP_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
27 (R/NW)	TCPERROCT	Rx TCP Error Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.TCPERROCT bit is set when the EMAC_RXTCP_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
26 (R/NW)	TCPGOCT	Rx TCP (Good) Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.TCPGOCT bit is set when the EMAC_RXTCP_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
25 (R/NW)	UDPERROCT	Rx UDP Error Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.UDPERROCT bit is set when the EMAC_RXUDP_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
24 (R/NW)	UDPGOCT	Rx UDP (Good) Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.UDPGOCT bit is set when the EMAC_RXUDP_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
23 (R/NW)	V6NOPAYOCT	Rx IPv6 No Payload Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.V6NOPAYOCT bit is set when the EMAC_RXIPV6_NOPAY_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
22 (R/NW)	V6HDERROCT	Rx IPv6 Header Error Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.V6HDERROCT bit is set when the EMAC_RXIPV6_HDR_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
21 (R/NW)	V6GOCT	Rx IPv6 (Good) Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.V6GOCT bit is set when the EMAC_RXIPV6_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.

Table 23-114: EMAC_IPC_RXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
20 (R/NW)	V4UDSBLOCT	Rx IPv4 UDS Disable Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4UDSBLOCT bit is set when the EMAC_RXIPV4_UDSBL_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
19 (R/NW)	V4FRAGOCT	Rx IPv4 Fragmented Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4FRAGOCT bit is set when the EMAC_RXIPV4_FRAG_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
18 (R/NW)	V4NOPAYOCT	Rx IPv4 No Payload Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4NOPAYOCT bit set when the EMAC_RXIPV4_NOPAY_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
17 (R/NW)	V4HDERROCT	Rx IPv4 Header Error Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4HDERROCT bit is set when the EMAC_RXIPV4_HDR_ERR_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
16 (R/NW)	V4GOCT	Rx IPv4 (Good) Octets Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4GOCT bit is set when the EMAC_RXIPV4_GD_OCT counter reaches half the maximum value, and also when it reaches the maximum value.
13 (R/NW)	ICMPERRFRM	Rx ICMP Error Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.ICMPERRFRM bit is set when the EMAC_RXICMP_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
12 (R/NW)	ICMPGFRM	Rx ICMP (Good) Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.ICMPGFRM bit is set when the EMAC_RXICMP_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
11 (R/NW)	TCPERRFRM	Rx TCP Error Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.TCPERRFRM bit is set when the EMAC_RXTCP_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
10 (R/NW)	TCPGFRM	Rx TCP (Good) Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.TCPGFRM bit is set when the EMAC_RXTCP_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.

Table 23-114: EMAC_IPC_RXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/NW)	UDPERRFRM	Rx IDP Error Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.UDPERRFRM bit is set when the EMAC_RXUDP_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
8 (R/NW)	UDPGFRM	Rx UDP (Good) Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.UDPGFRM bit is set when the EMAC_RXUDP_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
7 (R/NW)	V6NOPAYFRM	Rx IPv6 No Payload Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.V6NOPAYFRM bit is set when the EMAC_RXIPV6_NOPAY_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
6 (R/NW)	V6HDERRFRM	Rx IPv6 Header Error Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.V6HDERRFRM bit is set when the EMAC_RXIPV6_HDR_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
5 (R/NW)	V6GFRM	Rx IPv6 (Good) Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.V6GFRM bit is set when the EMAC_RXIPV6_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
4 (R/NW)	V4UDSBLFRM	Rx IPv4 UDS Disable Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4UDSBLFRM bit is set when the EMAC_RXIPV4_UDSBL_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
3 (R/NW)	V4FRAGFRM	Rx IPv4 Fragmented Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4FRAGFRM bit is set when the EMAC_RXIPV4_FRAG_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
2 (R/NW)	V4NOPAYFRM	Rx IPv4 No Payload Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4NOPAYFRM bit is set when the EMAC_RXIPV4_NOPAY_FRM counter reaches half the maximum value, and also when it reaches the maximum value.
1 (R/NW)	V4HDERRFRM	Rx IPv4 Header Error Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4HDERRFRM bit is set when the EMAC_RXIPV4_HDR_ERR_FRM counter reaches half the maximum value, and also when it reaches the maximum value.

Table 23-114: EMAC_IPC_RXINT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/NW)	V4GFRM	Rx IPv4 (Good) Frames Count Half/Full Interrupt. The EMAC_IPC_RXINT.V4GFRM bit is set when the EMAC_RXIPV4_GD_FRM counter reaches half the maximum value, and also when it reaches the maximum value.

Rx IPv4 Datagrams (Good) Register

The EMAC_RXIPV4_GD_FRM register contains a count of the number of good IPv4 datagrams received with the TCP, UDP, or ICMP payload.

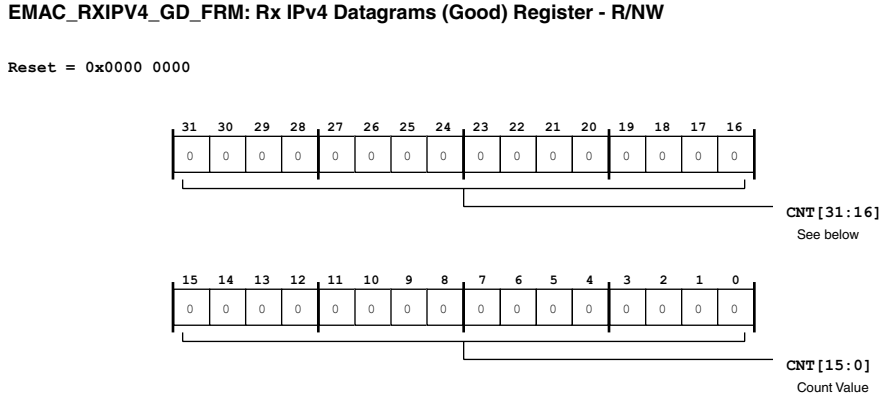


Figure 23-85: EMAC_RXIPV4_GD_FRM Register Diagram

Table 23-115: EMAC_RXIPV4_GD_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams Header Errors Register

The EMAC_RXIPV4_HDR_ERR_FRM register contains a count of the number of IPv4 datagrams received with header (checksum, length, or version mismatch) errors.

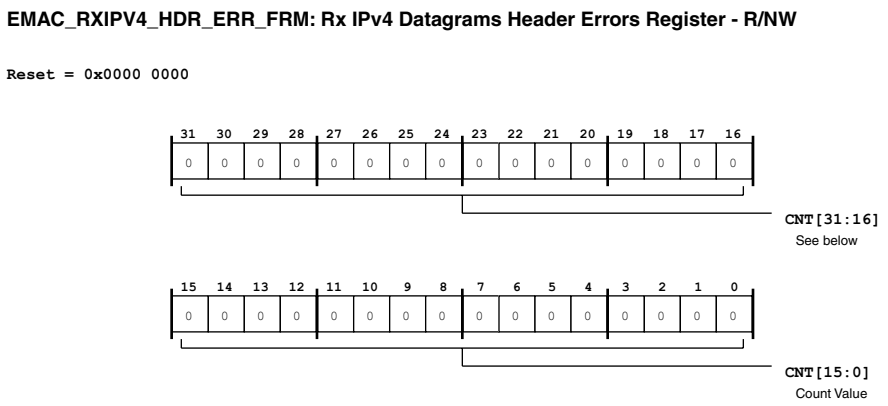


Figure 23-86: EMAC_RXIPV4_HDR_ERR_FRM Register Diagram

Table 23-116: EMAC_RXIPV4_HDR_ERR_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams No Payload Frame Register

The EMAC_RXIPV4_NOPAY_FRM register contains a count of the number of IPv4 datagram frames received that did not have a TCP, UDP, or ICMP payload processed by the Checksum engine.

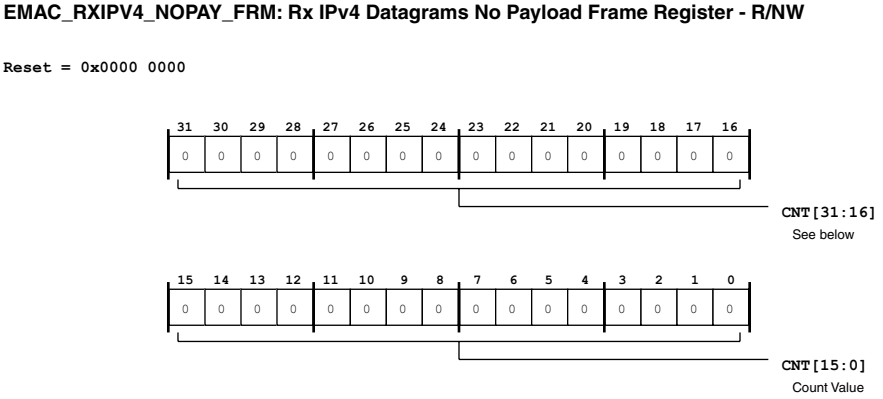


Figure 23-87: EMAC_RXIPV4_NOPAY_FRM Register Diagram

Table 23-117: EMAC_RXIPV4_NOPAY_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams Fragmented Frames Register

The EMAC_RXIPV4_FRAG_FRM register contains a count of the number of good IPv4 datagrams with fragmentation.

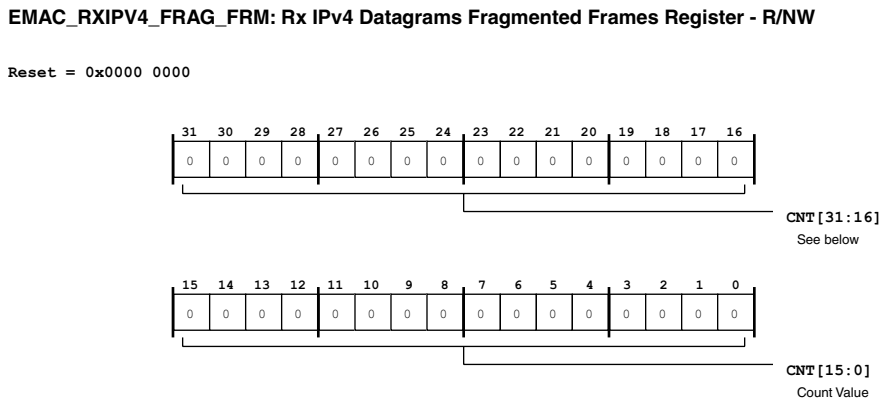


Figure 23-88: EMAC_RXIPV4_FRAG_FRM Register Diagram

Table 23-118: EMAC_RXIPV4_FRAG_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 UDP Disabled Frames Register

The EMAC_RXIPV4_UDSBL_FRM register contains a count of the number of good IPv4 datagrams received that had a UDP payload with checksum disabled.

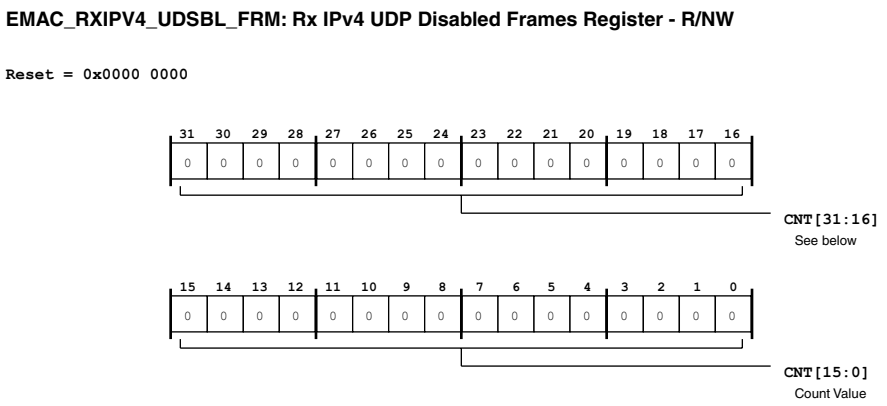


Figure 23-89: EMAC_RXIPV4_UDSBL_FRM Register Diagram

Table 23-119: EMAC_RXIPV4_UDSBL_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 Datagrams Good Frames Register

The EMAC_RXIPV6_GD_FRM register contains a count of the number of good IPv6 datagrams received with TCP, UDP, or ICMP payloads.

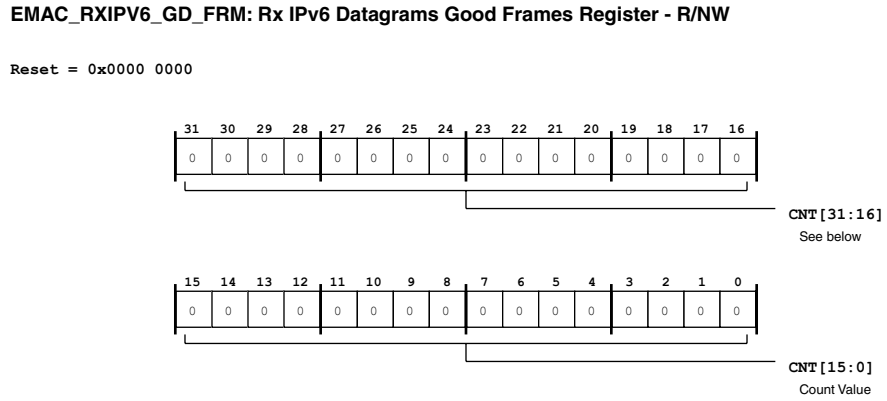


Figure 23-90: EMAC_RXIPV6_GD_FRM Register Diagram

Table 23-120: EMAC_RXIPV6_GD_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 Datagrams Header Error Frames Register

The EMAC_RXIPV6_HDR_ERR_FRM register contains a count of the number of IPv6 datagrams received with header errors (length or version mismatch).

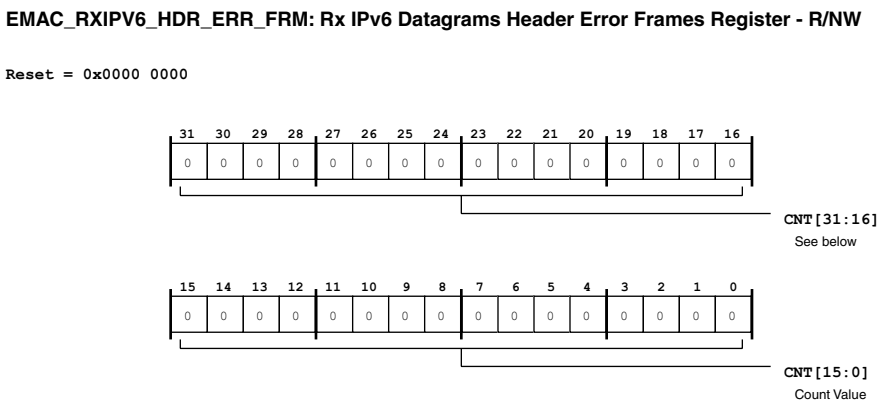


Figure 23-91: EMAC_RXIPV6_HDR_ERR_FRM Register Diagram

Table 23-121: EMAC_RXIPV6_HDR_ERR_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 Datagrams No Payload Frames Register

The EMAC_RXIPV6_NOPAY_FRM register contains a count of the number of IPv6 datagram frames received that did not have a TCP, UDP, or ICMP payload. This includes all IPv6 datagrams with fragmentation or security extension headers.

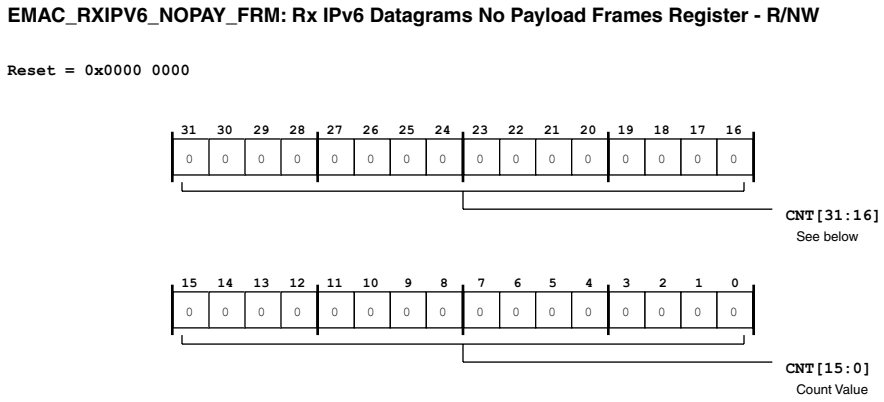


Figure 23-92: EMAC_RXIPV6_NOPAY_FRM Register Diagram

Table 23-122: EMAC_RXIPV6_NOPAY_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx UDP Good Frames Register

The EMAC_RXUDP_GD_FRM register contains a count of the number of good IP datagrams with a good UDP payload. This counter is not updated when the rxipv4_udsbl_frms counter is incremented.

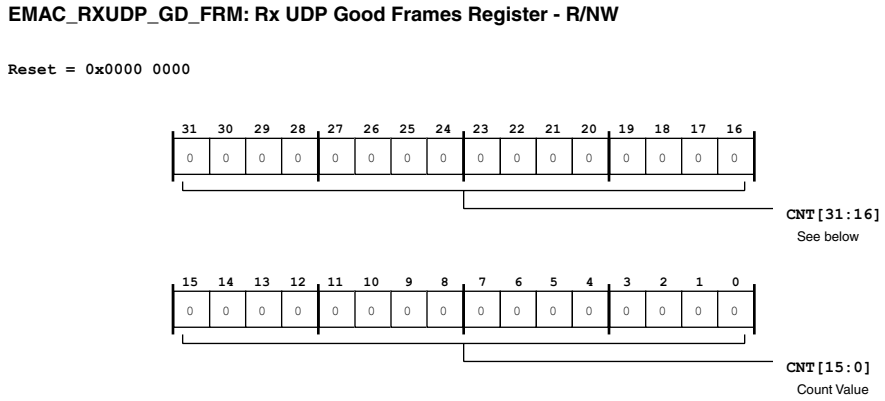


Figure 23-93: EMAC_RXUDP_GD_FRM Register Diagram

Table 23-123: EMAC_RXUDP_GD_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx UDP Error Frames Register

The EMAC_RXUDP_ERR_FRM register contains a count of the number of good IP datagrams whose UDP payload has a checksum error.

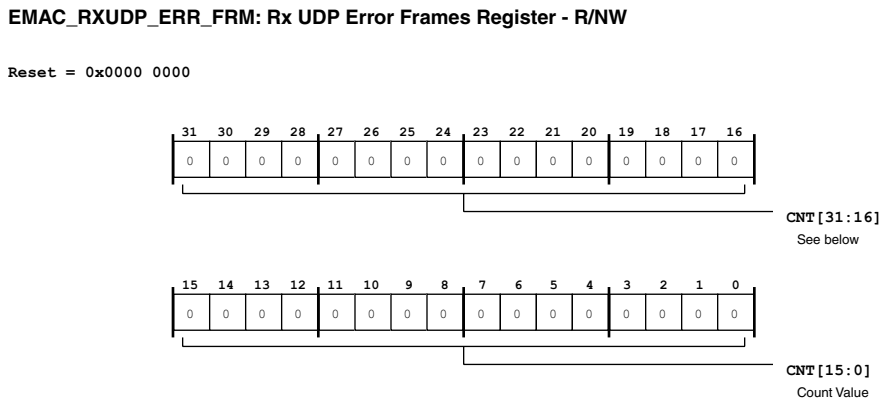


Figure 23-94: EMAC_RXUDP_ERR_FRM Register Diagram

Table 23-124: EMAC_RXUDP_ERR_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx TCP Good Frames Register

The EMAC_RXTCP_GD_FRM register contains a count of the number of good IP datagrams with a good TCP payload.

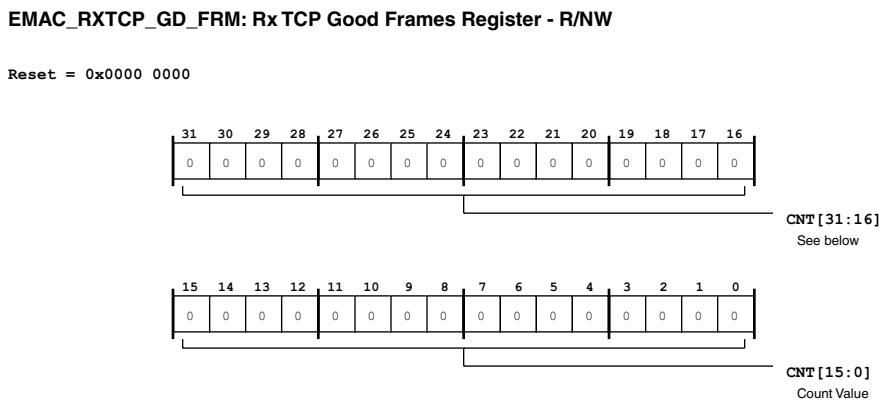


Figure 23-95: EMAC_RXTCP_GD_FRM Register Diagram

Table 23-125: EMAC_RXTCP_GD_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx TCP Error Frames Register

The EMAC_RXTCP_ERR_FRM register contains a count of the number of good IP datagrams whose TCP payload has a checksum error.

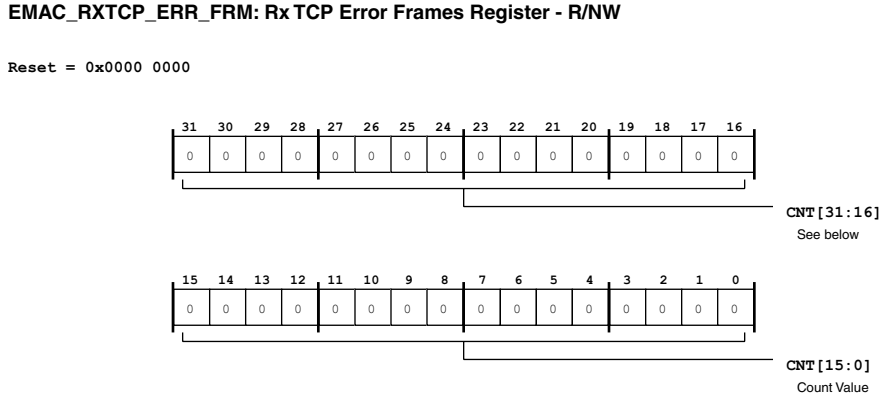


Figure 23-96: EMAC_RXTCP_ERR_FRM Register Diagram

Table 23-126: EMAC_RXTCP_ERR_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx ICMP Good Frames Register

The EMAC_RXICMP_GD_FRM register contains a count of the number of good IP datagrams with a good ICMP payload.

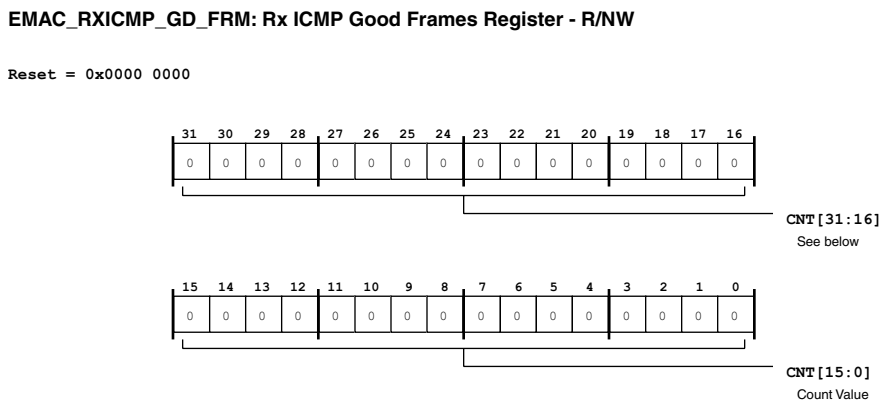


Figure 23-97: EMAC_RXICMP_GD_FRM Register Diagram

Table 23-127: EMAC_RXICMP_GD_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx ICMP Error Frames Register

The EMAC_RXICMP_ERR_FRM register contains a count of the number of good IP datagrams whose ICMP payload has a checksum error.

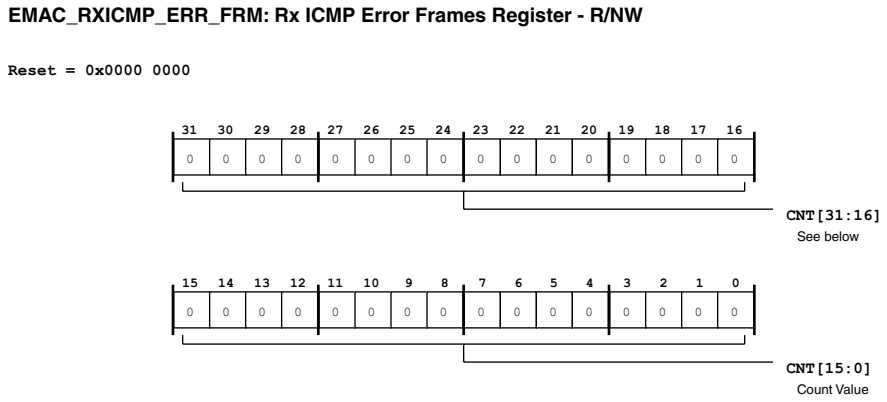


Figure 23-98: EMAC_RXICMP_ERR_FRM Register Diagram

Table 23-128: EMAC_RXICMP_ERR_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams Good Octets Register

The EMAC_RXIPV4_GD_OCT register contains a count of the number of bytes received in good IPv4 datagrams encapsulating TCP, UDP, or ICMP data.

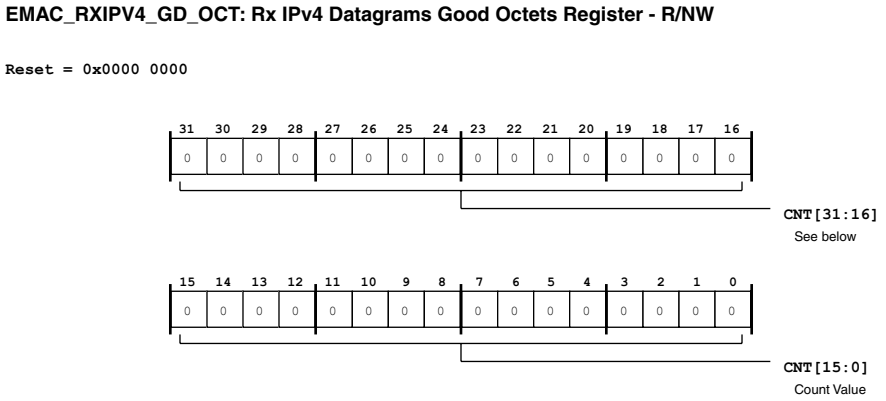


Figure 23-99: EMAC_RXIPV4_GD_OCT Register Diagram

Table 23-129: EMAC_RXIPV4_GD_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams Header Errors Register

The EMAC_RXIPV4_HDR_ERR_OCT register contains a count of the number of bytes received in IPv4 datagrams with header errors (checksum, length, version mismatch). The value in the Length field of IPv4 header is used to update this counter.

EMAC_RXIPV4_HDR_ERR_OCT: Rx IPv4 Datagrams Header Errors Register - R/NW

Reset = 0x0000 0000

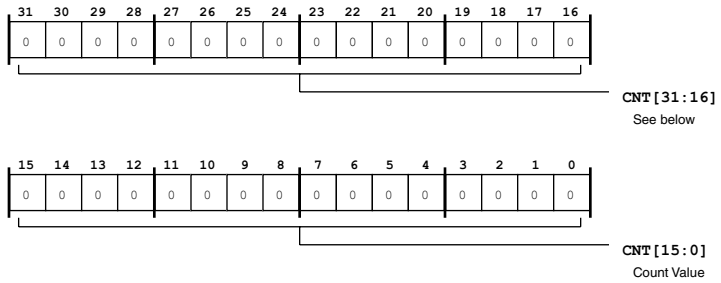


Figure 23-100: EMAC_RXIPV4_HDR_ERR_OCT Register Diagram

Table 23-130: EMAC_RXIPV4_HDR_ERR_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams No Payload Octets Register

The EMAC_RXIPV4_NOPAY_OCT register contains a count of the number of bytes received in IPv4 datagrams that did not have a TCP, UDP, or ICMP payload. The value in the IPv4 headers Length field is used to update this counter.

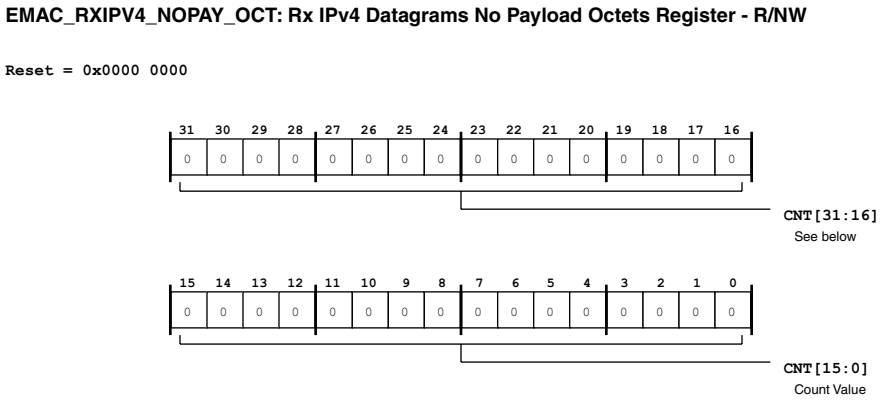


Figure 23-101: EMAC_RXIPV4_NOPAY_OCT Register Diagram

Table 23-131: EMAC_RXIPV4_NOPAY_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 Datagrams Fragmented Octets Register

The EMAC_RXIPV4_FRAG_OCT register contains a count of the number of bytes received in fragmented IPv4 datagrams. The value in the IPv4 headers Length field is used to update this counter.

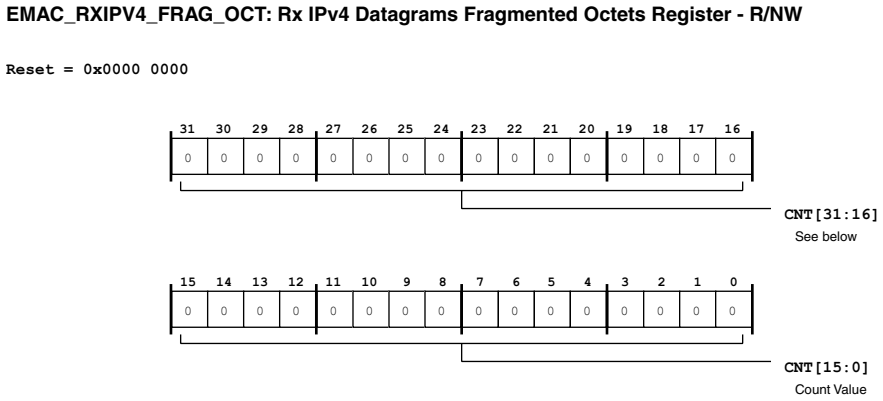


Figure 23-102: EMAC_RXIPV4_FRAG_OCT Register Diagram

Table 23-132: EMAC_RXIPV4_FRAG_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv4 UDP Disabled Octets Register

The EMAC_RXIPV4_UDSBL_OCT register contains a count of the number of bytes received in a UDP segment that had the UDP checksum disabled. This counter does not count IP Header bytes.

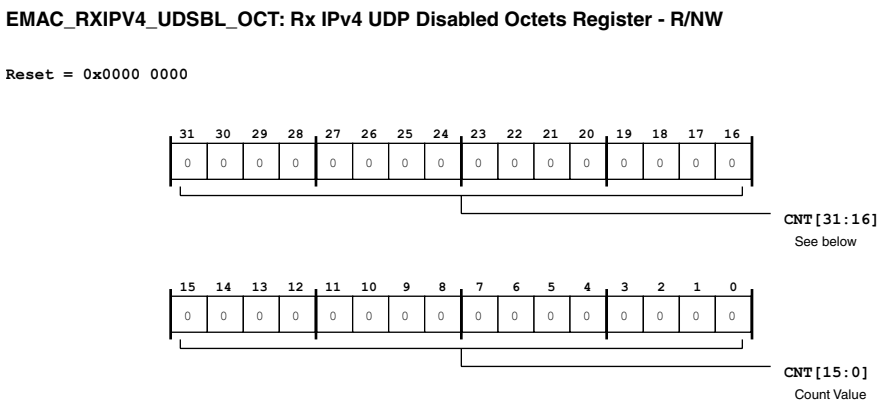


Figure 23-103: EMAC_RXIPV4_UDSBL_OCT Register Diagram

Table 23-133: EMAC_RXIPV4_UDSBL_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 Good Octets Register

The EMAC_RXIPV6_GD_OCT register contains a count of the number of bytes received in good IPv6 datagrams encapsulating TCP, UDP or ICMPv6 data

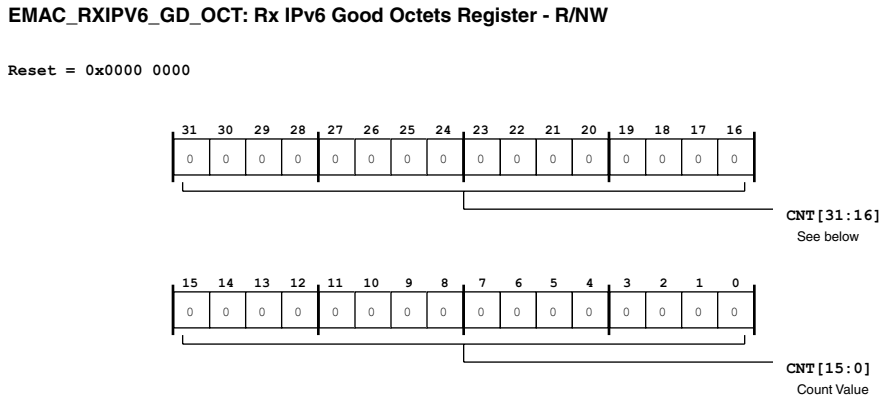


Figure 23-104: EMAC_RXIPV6_GD_OCT Register Diagram

Table 23-134: EMAC_RXIPV6_GD_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 Header Errors Register

The EMAC_RXIPV6_HDR_ERR_OCT register contains a count of the number of bytes received in IPv6 datagrams with header errors (length, version mismatch). The value in the IPv6 headers Length field is used to update this counter.

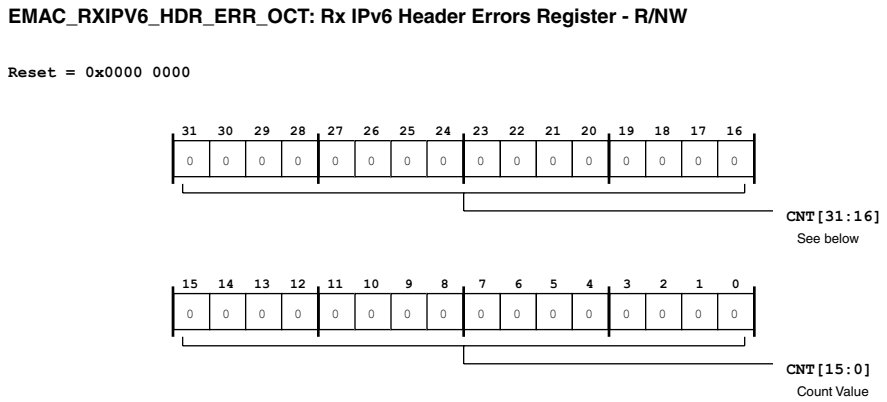


Figure 23-105: EMAC_RXIPV6_HDR_ERR_OCT Register Diagram

Table 23-135: EMAC_RXIPV6_HDR_ERR_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx IPv6 No Payload Octets Register

The EMAC_RXIPV6_NOPAY_OCT register contains a count of the number of bytes received in IPv6 datagrams that did not have a TCP, UDP, or ICMP payload. The value in the IPv6 headers Length field is used to update this counter.

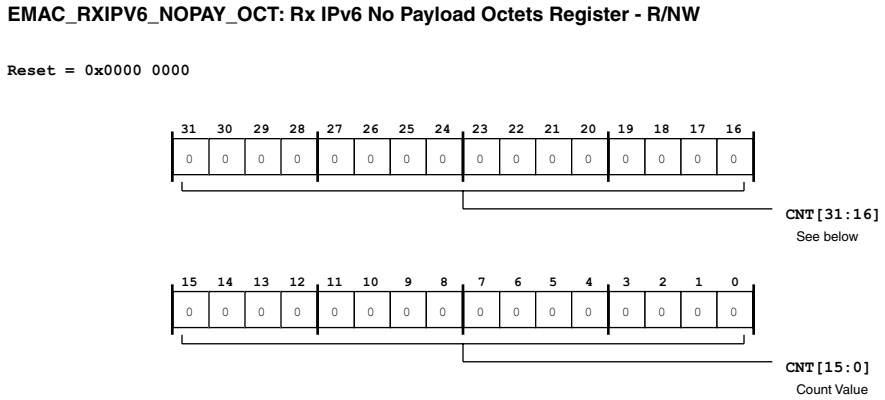


Figure 23-106: EMAC_RXIPV6_NOPAY_OCT Register Diagram

Table 23-136: EMAC_RXIPV6_NOPAY_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx UDP Good Octets Register

The EMAC_RXUDP_GD_OCT register contains a count of the number of bytes received in a good UDP segment. This counter (and the counters below) does not count IP header bytes.

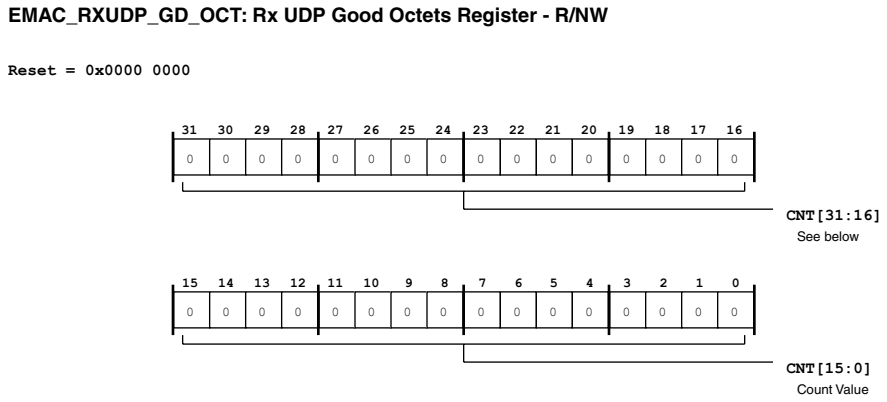


Figure 23-107: EMAC_RXUDP_GD_OCT Register Diagram

Table 23-137: EMAC_RXUDP_GD_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx UDP Error Octets Register

The EMAC_RXUDP_ERR_OCT register contains a count of the number of bytes received in a UDP segment that had checksum errors.

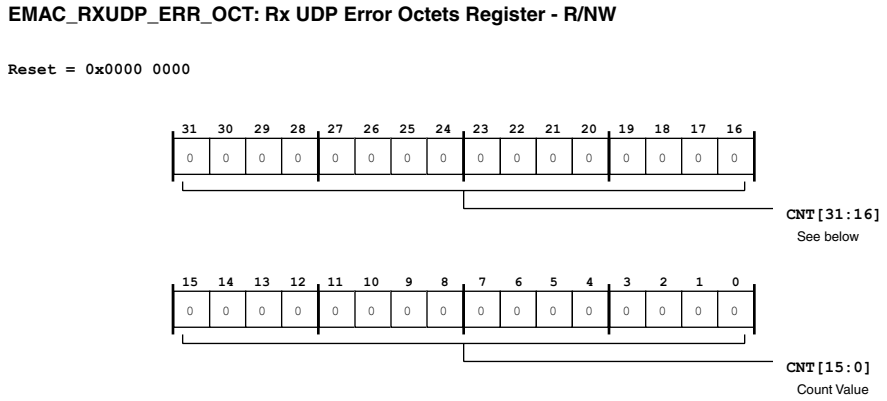


Figure 23-108: EMAC_RXUDP_ERR_OCT Register Diagram

Table 23-138: EMAC_RXUDP_ERR_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx TCP Good Octets Register

The EMAC_RXTCP_GD_OCT register contains a count of the number of bytes received in a good TCP segment.

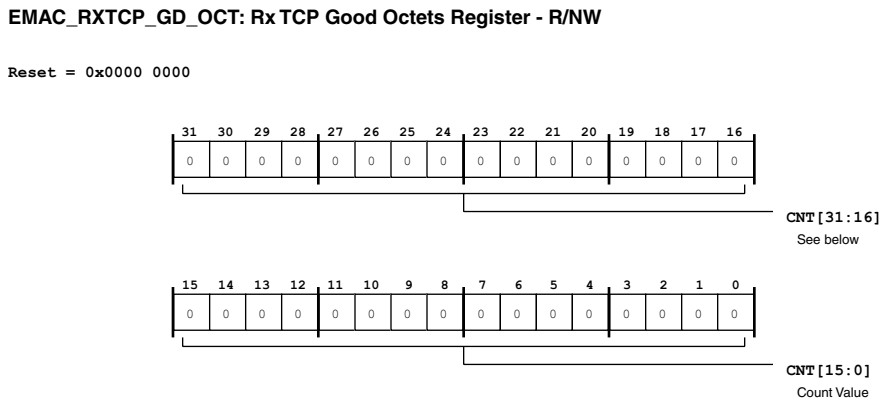


Figure 23-109: EMAC_RXTCP_GD_OCT Register Diagram

Table 23-139: EMAC_RXTCP_GD_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx TCP Error Octets Register

The EMAC_RXTCP_ERR_OCT register contains a count of the number of bytes received in a TCP segment with checksum errors.

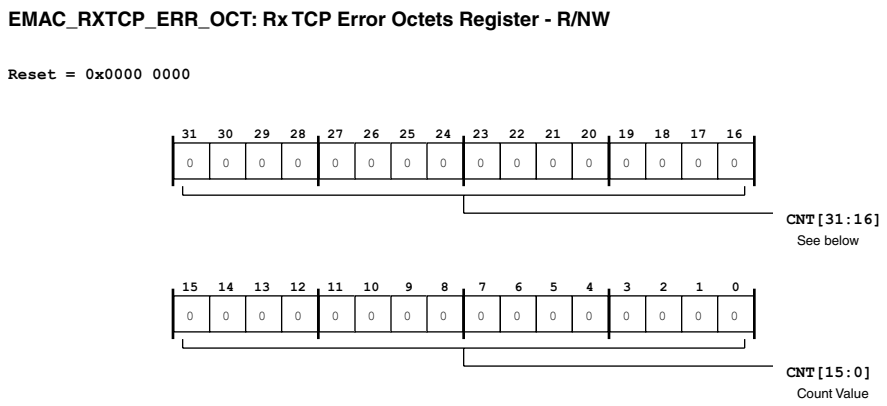


Figure 23-110: EMAC_RXTCP_ERR_OCT Register Diagram

Table 23-140: EMAC_RXTCP_ERR_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx ICMP Good Octets Register

The EMAC_RXICMP_GD_OCT register contains a count of the Number of bytes received in a good ICMP segment.

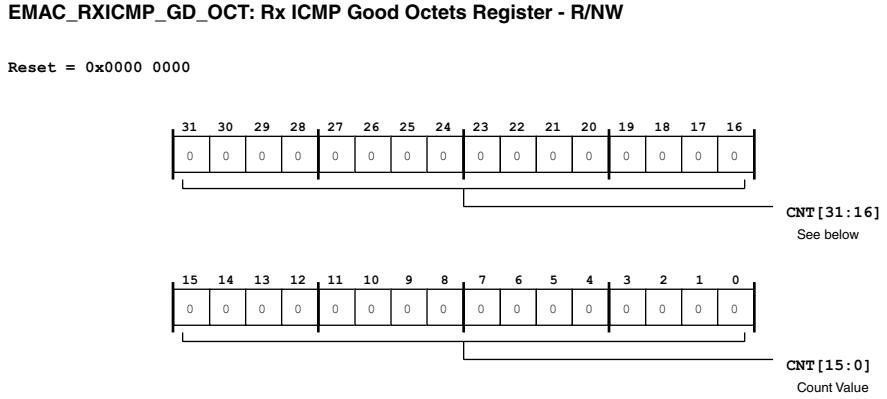


Figure 23-111: EMAC_RXICMP_GD_OCT Register Diagram

Table 23-141: EMAC_RXICMP_GD_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Rx ICMP Error Octets Register

The EMAC_RXICMP_ERR_OCT register contains a count of the number of bytes received in an ICMP segment with checksum errors.

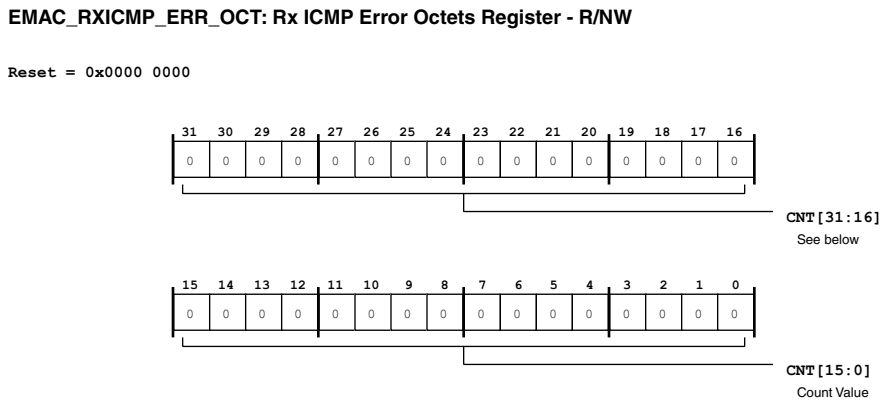


Figure 23-112: EMAC_RXICMP_ERR_OCT Register Diagram

Table 23-142: EMAC_RXICMP_ERR_OCT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Count Value.

Time Stamp Control Register

The EMAC_TM_CTL register controls time stamp generation and update. The EMAC_TM_CTL.SNAPTYPSEL, EMAC_TM_CTL.TSMSTRENA, and EMAC_TM_CTL.TSEVNTENA bits work together to decide the set of PTP packet types for which snapshot needs to be taken. (Encoding shown in table.)

Table 23-143:

SNAPTYPSEL()	TSMSTRENA	TSEVNTENA	Messages for which snapshot is taken
00	X	0	SYNC, Follow_Up, Delay_Req, Delay_Resp

Table 23-143: (Continued)

SNAPTYPSEL()	TSMSTRENA	TSEVNTENA	Messages for which snapshot is taken
00	0	1	SYNC
00	1	1	Delay_Req
01	X	0	SYNC, Follow_Up, Delay_Req, Delay_Resp, Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up
01	0	1	SYNC, Pdelay_Req, Pdelay_Resp
01	1	1	Delay_Req, Pdelay_Req, Pdelay_Resp
10	X	X	SYNC, Delay_Req
11	X	X	Pdelay_Req, Pdelay_Resp

EMAC_TM_CTL: Time Stamp Control Register - R/W

Reset = 0x0000 2000

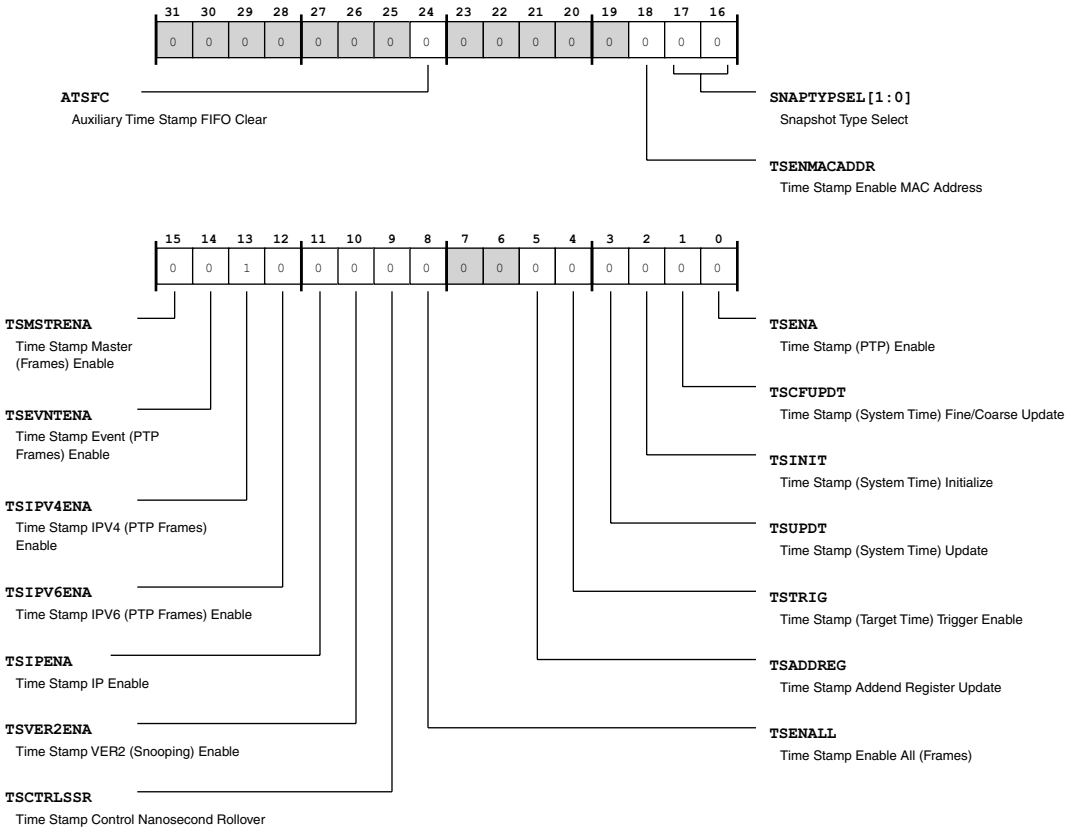


Figure 23-113: EMAC_TM_CTL Register Diagram

Table 23-144: EMAC_TM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
24 (R/W)	ATSFC	Auxiliary Time Stamp FIFO Clear. The EMAC_TM_CTL.ATSFC bit, when set, resets the pointers of the Auxiliary Snapshot FIFO. This bit is cleared when the pointers are reset and the FIFO is empty. When this bit is cleared, auxiliary snapshots gets stored in the FIFO.	
18 (R/W)	TSENMACADDR	Time Stamp Enable MAC Address. The EMAC_TM_CTL.TSENMACADDR bit, when set, uses the DA MAC address (that matches the EMAC_ADDR0_LO and EMAC_ADDR0_HI registers) to filter the PTP frames when PTP is sent directly over Ethernet.	
		0	Disable PTP MAC address filter
		1	Enable PTP MAC address filter
17:16 (R/W)	SNAPTYPSEL	Snapshot Type Select. The EMAC_TM_CTL.SNAPTYPSEL bits along with bit 15 and 14 decide the set of PTP packet types for which snapshot needs to be taken. (See the table in the EMAC_TM_CTL register description.)	
15 (R/W)	TSMSTRENA	Time Stamp Master (Frames) Enable. The EMAC_TM_CTL.TSMSTRENA bit, when set, takes the snapshot for messages relevant to master node only else snapshot is taken for PTP messages relevant to slave node.	
		0	Enable Snapshot for Slave Messages
		1	Enable Snapshot for Master Messages
14 (R/W)	TSEVNTENA	Time Stamp Event (PTP Frames) Enable. The EMAC_TM_CTL.TSEVNTENA bit, when set, takes the time stamp snapshot for PTP event messages only (SYNC, Delay_Req, Pdelay_Req, or Pdelay_Resp). When reset, the snapshot is taken for all PTP messages except Announce, Management, and Signaling.	
		0	Enable Time Stamp for All Messages
		1	Enable Time Stamp for Event Messages Only

Table 23-144: EMAC_TM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	TSIPV4ENA	Time Stamp IPV4 (PTP Frames) Enable. The EMAC_TM_CTL.TSIPV4ENA bit, when set, directs the EMAC receiver to process the PTP packets encapsulated in UDP over IPv4 packets. When this bit is clear, the MAC ignores the PTP transported over UDP-IPv4 packets. This bit is set by default.	
		0	Disable Time Stamp for PTP Over IPv4 Frames
		1	Enable Time Stamp for PTP Over IPv4 Frames
12 (R/W)	TSIPV6ENA	Time Stamp IPV6 (PTP Frames) Enable. The EMAC_TM_CTL.TSIPV6ENA bit, when set, directs the EMAC receiver to process PTP packets encapsulated in UDP over IPv6 packets. When this bit is clear, the MAC ignores the PTP transported over UDP-IPv6 packets.	
		0	Disable Time Stamp for PTP Over IPv6 frames
		1	Enable Time Stamp for PTP Over IPv6 Frames
11 (R/W)	TSIPENA	Time Stamp IP Enable. The EMAC_TM_CTL.TSIPENA bit, when set, directs the EMAC receiver to process the PTP packets encapsulated directly in the Ethernet frames. When this bit is clear, the MAC ignores PTP over Ethernet packets.	
		0	Disable PTP Over Ethernet Frames
		1	Enable PTP Over Ethernet Frames
10 (R/W)	TSVER2ENA	Time Stamp VER2 (Snooping) Enable. The EMAC_TM_CTL.TSVER2ENA bit, when set, processes the PTP packets using the 1588 version 2 format (enables PTP packet snooping for VER2) else processed using the version 1 format.	
		0	Disable packet snooping for V2 frames
		1	Enable packet snooping for V2 frames

Table 23-144: EMAC_TM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	TSCTRLSSR	Time Stamp Control Nanosecond Rollover. The EMAC_TM_CTL.TSCTRLSSR bit, when set, rolls over the EMAC_TM_NSEC register after 0x3B9A_C9FF value (10^9-1) and increments the EMAC_TM_SEC register. When reset, the roll over value of EMAC_TM_NSEC register is 0x7FFF_FFFF. The nanosecond increment has to be programmed correctly depending on the PTP reference clock frequency and this bit value.	
		0	Roll Over Nanosecond After 0x7FFFFFFF
		1	Roll Over Nanosecond After 0x3B9AC9FF
8 (R/W)	TSENALL	Time Stamp Enable All (Frames). The EMAC_TM_CTL.TSENALL bit, when set, enables the time stamp snapshot for all frames received by the core.	
		0	Disable timestamp for all frames
		1	Enable timestamp for all frames
5 (R/W1S)	TSADDREG	Time Stamp Addend Register Update. The EMAC_TM_CTL.TSADDREG bit, when set, updates the contents of the EMAC_TM_ADDEND register for fine correction. This bit is cleared when the update is completed. This bit should be zero before setting it.	
4 (R/W1S)	TSTRIG	Time Stamp (Target Time) Trigger Enable. The EMAC_TM_CTL.TSTRIG bit, when set, generates the time stamp interrupt when the System Time becomes greater than the value written in EMAC_TM_TGTM register. This bit is reset after the generation of the Time Stamp Trigger Interrupt.	
		1	Interrupt (TS) if system time is greater than target time register
3 (R/W1S)	TSUPDT	Time Stamp (System Time) Update. The EMAC_TM_CTL.TSUPDT bit, when set, updates (adds/subtracts) the system time with the value specified in the EMAC_TM_SECUPDT register and EMAC_TM_NSECUPDT register. This bit should read =0 before updating it. This bit is reset when the update is completed in hardware. The EMAC_TM_NSEC register is not updated.	
		1	System time updated with Time stamp register values

Table 23-144: EMAC_TM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1S)	TSINIT	Time Stamp (System Time) Initialize. The EMAC_TM_CTL.TSINIT bit, when set, initializes (over-writes) the system time with the value specified in the EMAC_TM_SECUPDT register and EMAC_TM_NSECUPDT register. This bit should read =0 before updating it. This bit is reset when the initialize is complete. The EMAC_TM_NSEC register can only be initialized.	
		1	System time initialized with Time stamp register values
1 (R/W)	TSCFUPDT	Time Stamp (System Time) Fine/Coarse Update. The EMAC_TM_CTL.TSCFUPDT bit, when set, indicates that the system times update to be done using fine correction method. When reset, it indicates the system time stamp correction to be done using Coarse method.	
		0	Use Coarse Correction Method for System Time Update
		1	Use Fine Correction Method for System Time Update
0 (R/W)	TSENA	Time Stamp (PTP) Enable. The EMAC_TM_CTL.TSENA bit, when set, enables PTP module for time stamping transmitted and received frames. It also enables System Time which will be used for time stamping the frames. User should initialize the System Time after setting this bit.	
		0	Disable PTP Module
		1	Enable PTP Module

Time Stamp Sub Second Increment Register

The EMAC_TM_SUBSEC register contains the value by which the sub second is incremented.

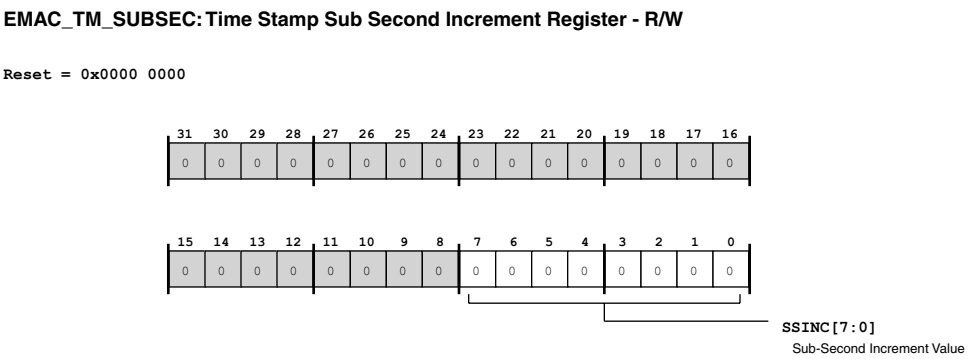


Figure 23-114: EMAC_TM_SUBSEC Register Diagram

Table 23-145: EMAC_TM_SUBSEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	SSINC	Sub-Second Increment Value. The value in the EMAC_TM_SUBSEC.SSINC bits is accumulated every PTP clock cycle with the contents of the nanosecond register. For example, when PTP clock is 50 MHz (period is 20 ns), the processor should program 20 (0x14) when the EMAC_TM_NSEC register has an accuracy of 1 ns (EMAC_TM_CTL.TSCTRLSSR bit is set). When EMAC_TM_CTL.TSCTRLSSR is clear, the EMAC_TM_NSEC register has a resolution of ~0.465ns. In this case, the processor should program a value of 43 (0x2B) that is derived by 20ns/0.465.

Time Stamp Low Seconds Register

The EMAC_TM_SEC register contains the lower 32 bits of the seconds field of the system time.

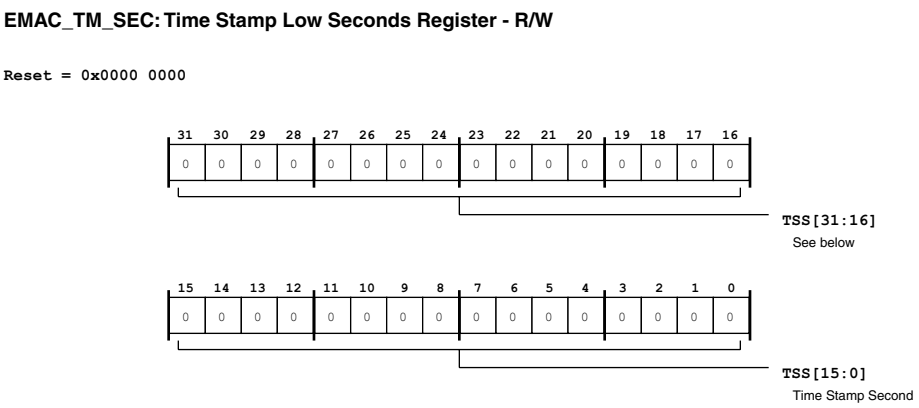


Figure 23-115: EMAC_TM_SEC Register Diagram

Table 23-146: EMAC_TM_SEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	TSS	Time Stamp Second. The value in the EMAC_TM_SEC.TSS bit field indicates the current value in seconds of the System Time maintained by the core.

Time Stamp Nanoseconds Register

The EMAC_TM_NSEC register contains the nanoseconds field of the system time.

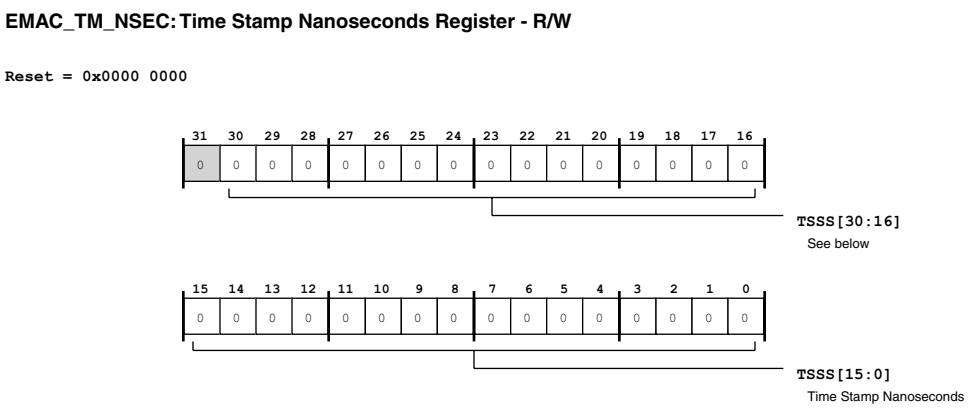


Figure 23-116: EMAC_TM_NSEC Register Diagram

Table 23-147: EMAC_TM_NSEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30:0 (R/NW)	TSSS	Time Stamp Nanoseconds. The value in the EMAC_TM_NSEC.TSSS bit field has the nanosecond representation of time, with an accuracy of 0.46 nanosecond. (When EMAC_TM_CTL.TSCTRLSSR is set, each bit represents 1 ns and the maximum value will be 0x3B9A_C9FF, after which it rolls-over to zero).

Time Stamp Seconds Update Register

The EMAC_TM_SECUPDT register contains the low 32 bits to be added to, subtracted from, or written to the seconds field of the system time.

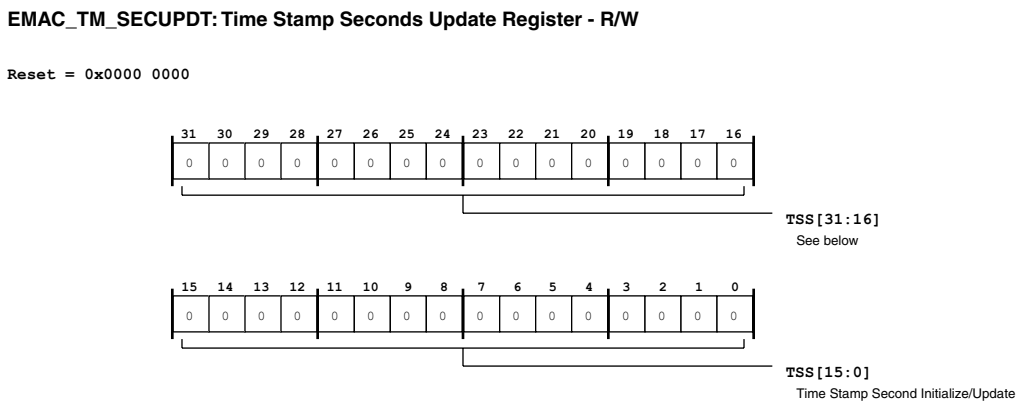


Figure 23-117: EMAC_TM_SECUPDT Register Diagram

Table 23-148: EMAC_TM_SECUPDT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TSS	Time Stamp Second Initialize/Update. The value in the EMAC_TM_SECUPDT.TSS bit field indicates the time, in seconds, to be initialized or added to or subtracted from the system time seconds.

Time Stamp Nanoseconds Update Register

The EMAC_TM_NSECUPDT register contains the low 32 bits to be added to, subtracted from, or written to the nanoseconds field of the system time.

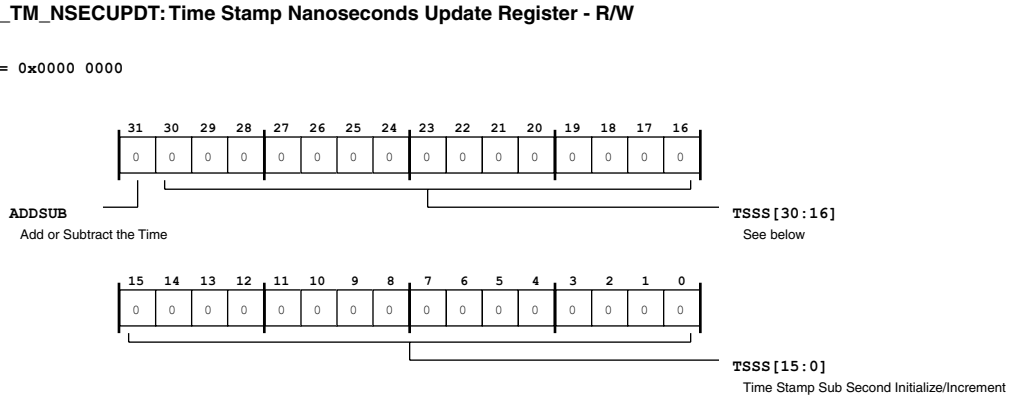


Figure 23-118: EMAC_TM_NSECUPDT Register Diagram

Table 23-149: EMAC_TM_NSECUPDT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	ADDSUB	Add or Subtract the Time. The EMAC_TM_NSECUPDT.ADDSUB bit, when set, subtracts the time value with the contents of the update registers. When this bit is reset, the time value is added with the contents of the update registers.
30:0 (R/W)	TSSS	Time Stamp Sub Second Initialize/Increment. The value in the EMAC_TM_NSECUPDT.TSSS bit field indicates the time, in nanoseconds, to be initialized or added to or subtracted from the system time nanoseconds.

Time Stamp Addend Register

The EMAC_TM_ADDEND register lets software adjust the clock frequency linearly to match the master clock frequency.

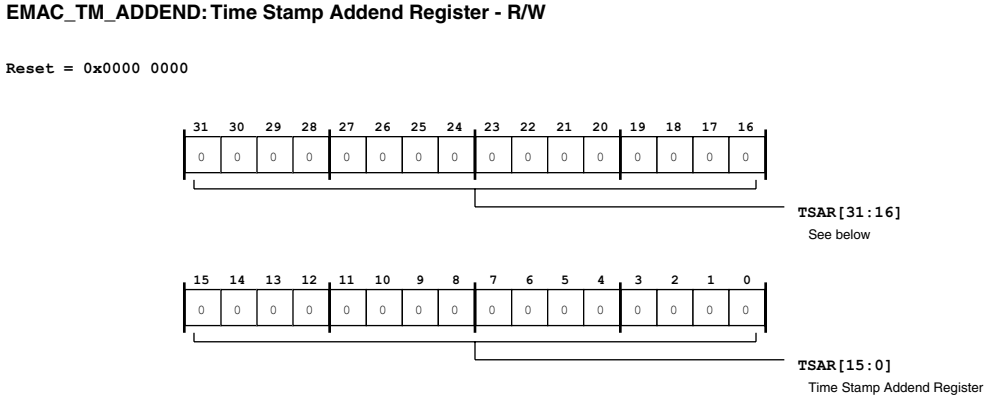


Figure 23-119: EMAC_TM_ADDEND Register Diagram

Table 23-150: EMAC_TM_ADDEND Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TSAR	Time Stamp Addend Register. The EMAC_TM_ADDEND.TSAR bits indicate the 32-bit time value to be added to the Accumulator register to achieve time synchronization.

Time Stamp Target Time Seconds Register

The EMAC_TM_TGTM register contains the high 32 bits of the target seconds field for comparison to the corresponding system time field.

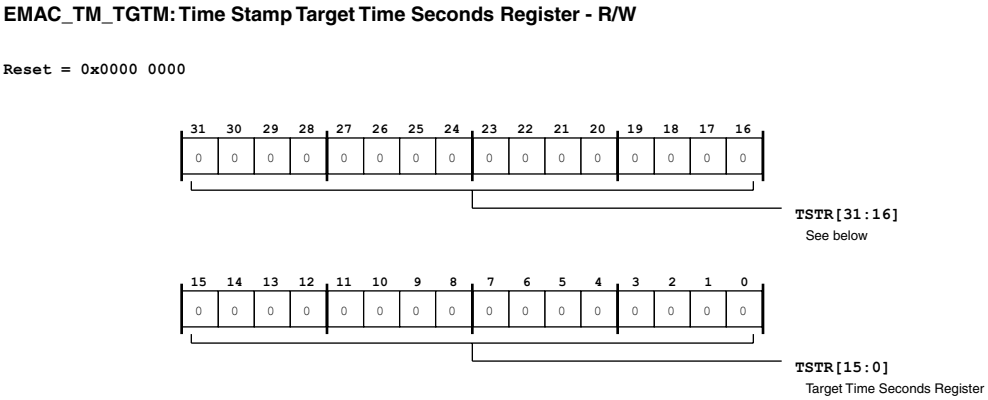


Figure 23-120: EMAC_TM_TGTM Register Diagram

Table 23-151: EMAC_TM_TGTM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TSTR	Target Time Seconds Register. The EMAC_TM_TGTM.TSTR bit field stores the time in seconds. When the time stamp value matches or exceeds both EMAC_TM_TGTM and EMAC_TM_NTGTM registers, based on the selection in the EMAC_TM_PPSCTL.TRGTMODSEL bits, the MAC starts or stops the PPS signal output and generates an interrupt (if enabled).

Time Stamp Target Time Nanoseconds Register

The EMAC_TM_NTGTM register contains the high 32 bits of the target nanoseconds field for comparison to the corresponding system time field.

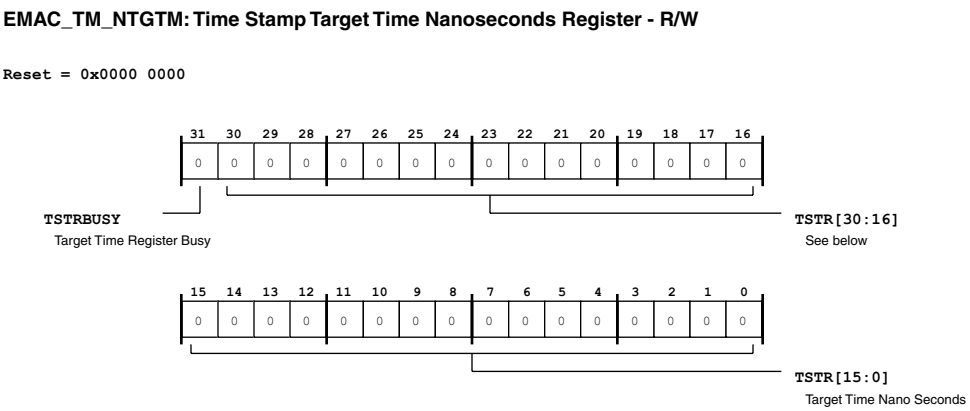


Figure 23-121: EMAC_TM_NTGTM Register Diagram

Table 23-152: EMAC_TM_NTGTM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	TSTRBUSY	Target Time Register Busy. The EMAC_TM_NTGTM.TSTRBUSY bit is set when Flexible PPS is enabled and the EMAC_TM_PPSCTL.PPSCTL field is programmed to 0001, 0010 or 0100. Programming the EMAC_TM_PPSCTL.PPSCTL field to 0001, 0010 or 0100, instructs the core to synchronize the EMAC_TM_TGTM and EMAC_TM_NTGTM registers to the PTP clock domain. The EMAC clears this bit after synchronizing the EMAC_TM_TGTM and EMAC_TM_NTGTM registers to the PTP clock domain The application must not update the EMAC_TM_TGTM and EMAC_TM_NTGTM registers when this bit is read as 1. Otherwise, the synchronization of the previous programmed time gets corrupted.
30:0 (R/W)	TSTR	Target Time Nano Seconds. The EMAC_TM_NTGTM.TSTR bit field stores the time in (signed) nanoseconds. When the value of the time stamp matches the both EMAC_TM_TGTM and EMAC_TM_NTGTM registers, based on the EMAC_TM_PPSCTL.TRGTMODSEL field, the MAC starts or stops the PPS signal output and generates an interrupt (if enabled). This value should not exceed 0x3B9A_C9FF when EMAC_TM_PPSCTL.TRGTMODSEL is set. The actual start or stop time of the PPS signal output may have an error margin up to one unit of sub-second increment value.

Time Stamp High Second Register

The EMAC_TM_HISEC register contains the upper 32 bits of the seconds field of the system time.

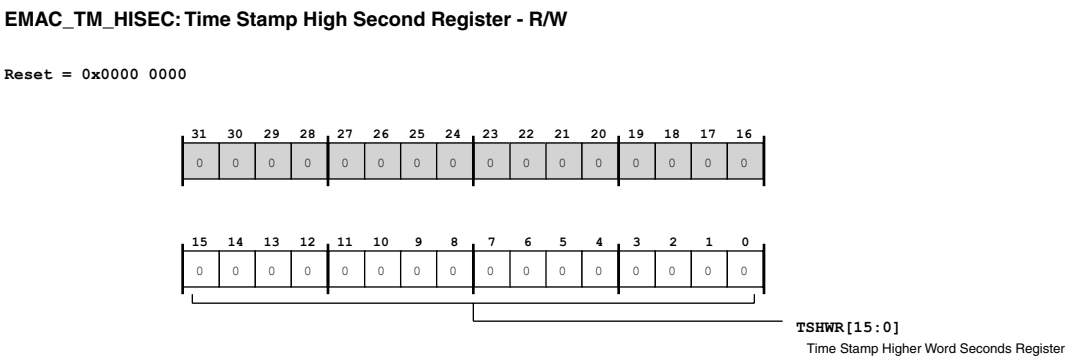


Figure 23-122: EMAC_TM_HISEC Register Diagram

Table 23-153: EMAC_TM_HISEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	TSHWR	Time Stamp Higher Word Seconds Register. The EMAC_TM_HISEC.TSHWR bit field contains the most significant 16-bits of the time stamp seconds value. The register is directly written to initialize the value. This register is incremented when there is an overflow from the 32-bits of the EMAC_TM_SEC register.

Time Stamp Status Register

The EMAC_TM_STMPSTAT register contains the PTP status.

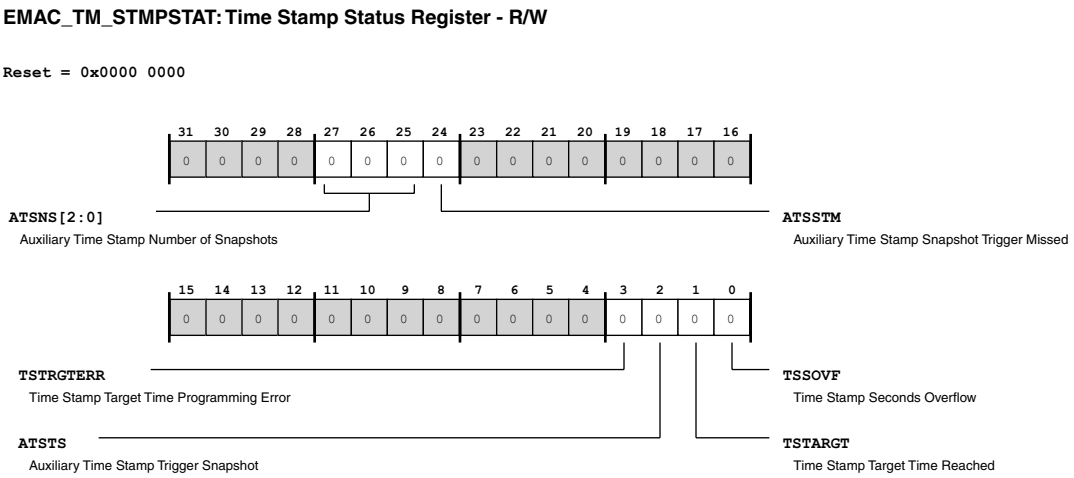


Figure 23-123: EMAC_TM_STMPSTAT Register Diagram

Table 23-154: EMAC_TM_STMPSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:25 (R/NW)	ATSNS	Auxiliary Time Stamp Number of Snapshots. The EMAC_TM_STMPSTAT.ATSNS bits indicate the number of Snapshots available in the FIFO. A value of 4 (100) indicates that the Auxiliary Snapshot FIFO is full. These bits are cleared (to 000) when the Auxiliary snapshot FIFO clear bit is set.
24 (RC/NW)	ATSSTM	Auxiliary Time Stamp Snapshot Trigger Missed. The EMAC_TM_STMPSTAT.ATSSTM bit is set when the Auxiliary time stamp snapshot FIFO is full and external trigger was set. This indicates that the latest snapshot was not stored in the FIFO.
3 (R/W)	TSTRGTERR	Time Stamp Target Time Programming Error. The EMAC_TM_STMPSTAT.TSTRGTERR bit is set when the target time, which is being programmed in the EMAC_TM_SEC and EMAC_TM_NSEC registers, has already elapsed. This bit is cleared when read by the application.
2 (RC/NW)	ATSTS	Auxiliary Time Stamp Trigger Snapshot. The EMAC_TM_STMPSTAT.ATSTS bit is set high when the auxiliary snapshot is written to the FIFO.
1 (RC/NW)	TSTARGT	Time Stamp Target Time Reached. The EMAC_TM_STMPSTAT.TSTARGT bit, when set, indicates the value of system time has reached or passed the value specified in the EMAC_TM_TGTM and EMAC_TM_NTGTM registers.
0 (RC/NW)	TSSOVF	Time Stamp Seconds Overflow. The EMAC_TM_STMPSTAT.TSSOVF bit, when set, indicates that the seconds value of the time stamp (when supporting PTP version 2 format) has overflowed beyond 0xFFFF_FFFF.

PPS Control Register

The EMAC_TM_PPSCTL register controls the interval of PPS output.

When the EMAC_TM_PPSCTL.PPSEN bit is disabled (=0, fixed PPS output), the EMAC_TM_PPSCTL.PPSCTL bits control the behavior of the PPS output signal. The default value of PPSCTRL is 0000 and the PPS output is 1 pulse every second. For other values of PPSCTRL, the PPS output becomes a generated clock. (See bit enumerations for frequencies.) In the binary rollover mode, the PPS output has a duty cycle of 50 percent with these frequencies. In the digital rollover mode, the PPS output frequency is an average number. The actual clock is of different frequency that gets synchronized every second. This behavior is

because of the non-linear toggling of the bits in the digital rollover mode in System Time - Nanoseconds Register.

When the EMAC_TM_PPSCTL.PPSEN bit is enabled (=1, flexible PPS output), the EMAC_TM_PPSCTL.PPSCTL bits function as PPSCMD. (See bit enumerations for commands.)Programming these bits with a non-zero value instructs the core to initiate an event. After the command is transferred or synchronized to the PTP clock domain, these bits gets cleared automatically. Software should ensure that these bits are programmed only when they are "all-zero".

- EMAC_TM_PPSCTL.PPSCTL=0000 CMD=No Command (PPS output =1Hz)
- EMAC_TM_PPSCTL.PPSCTL=0001

  (CMD=START Single; BR=2kHz; DR=1kHz) This PPS command generates single pulse rising at start point defined in Target Time Registers and of duration defined in PPS Width Register. (Binary Rollover = 2 kHz; Digital Rollover = 1 kHz.)
- EMAC_TM_PPSCTL.PPSCTL=0010

  (CMD=START Pulse; BR=4kHz; DR=2kHz) This PPS command generates the train of pulses rising at the start point defined in the Target Time Registers and of duration defined in the PPS Width Register and repeated at interval defined in the PPS Interval Register. By default, the PPS pulse train is free-running unless stopped by 'STOP Pulse train at time' or 'STOP Pulse Train immediately' commands. (Binary Rollover = 4 kHz; Digital Rollover = 2 kHz.)
- EMAC_TM_PPSCTL.PPSCTL=0011

  (CMD=Cancel START; BR=8kHz; DR=4kHz) This PPS command cancels the START Single Pulse and START Pulse Train commands if the system time has not crossed the programmed start time. (Binary Rollover = 4 kHz; Digital Rollover = 4 kHz.)
- EMAC_TM_PPSCTL.PPSCTL=0100

  (CMD=STOP Pulse Time; BR=16kHz; DR=8kHz) This PPS command stops the train of pulses initiated by the START Pulse Train command (PPSCMD = 0010) after the time programmed in the Target Time registers elapses. (Binary Rollover = 16 kHz; Digital Rollover = 8 kHz.)
- EMAC_TM_PPSCTL.PPSCTL=0101

  (CMD=STOP Pulse Now) This command immediately stops the train of pulses initiated by the START Pulse Train command (PPSCMD = 0010).
- EMAC_TM_PPSCTL.PPSCTL=0110

  (CMD=Cancel STOP Pulse) This PPS command cancels the STOP pulse train at time command if the programmed stop time has not elapsed. The PPS pulse train becomes free-running on the successful execution of this command.

All values not shown in the bit enumerations are reserved.

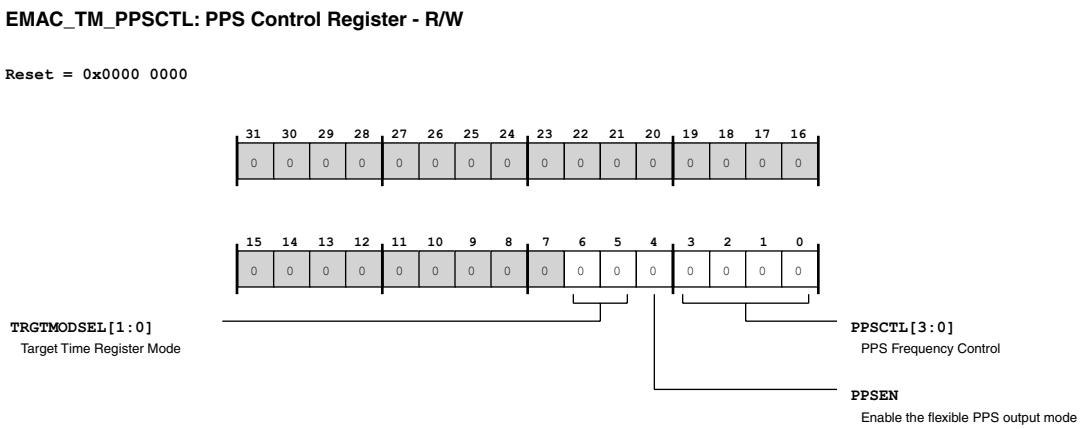


Figure 23-124: EMAC_TM_PPSCTL Register Diagram

Table 23-155: EMAC_TM_PPSCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6:5 (R/W)	TRGTMODSEL	Target Time Register Mode. The EMAC_TM_PPSCTL.TRGTMODSEL bits select the target time register mode.	
		0	Interrupt Only The Target Time registers are programmed only for interrupt event generation.
		1	Reserved
		2	Interrupt and PPS Start/Stop The Target Time registers are programmed for interrupt event and for starting or stopping the PPS output signal generation.
		3	PPS Start/Stop Only The Target Time registers are programmed only for starting or stopping the PPS output signal generation. No interrupt is asserted.
4 (R/W)	PPSEN	Enable the flexible PPS output mode. The EMAC_TM_PPSCTL.PPSEN bit enables PPS operation. When set low, the EMAC_TM_PPSCTL.PPSCTL field controls frequency of Fixed PPS output. When set high, EMAC_TM_PPSCTL.PPSCTL field is used to command Flexible PPS output.	

Table 23-155: EMAC_TM_PPSCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/W)	PPSCTL	PPS Frequency Control. When the EMAC_TM_PPSCTL.PPSEN bit is disabled (=0, fixed PPS output), the EMAC_TM_PPSCTL.PPSCTL bits control the behavior of the PPS output signal. When the EMAC_TM_PPSCTL.PPSEN bit is enabled (=1, flexible PPS output), the EMAC_TM_PPSCTL.PPSCTL bits function as PPSCMD. (See bit enumerations for PPS output frequency, rollover, and PPS commands.)Programming these bits with a non-zero value instructs the core to initiate an event. After the command is transferred or synchronized to the PTP clock domain, these bits gets cleared automatically. Software should ensure that these bits are programmed only when they are "all-zero". All values not shown in the bit enumerations are reserved. For more information about the EMAC_TM_PPSCTL.PPSCTL bits, see the register description.	
		0	CMD=No Command
		1	CMD=START Single; BR=2kHz; DR=1kHz For more info, see register description.
		2	CMD=START Pulse; BR=4kHz; DR=2kHz For more info, see register description.
		3	CMD=Cancel START; BR=8kHz; DR=4kHz For more info, see register description.
		4	CMD=STOP Pulse Time; BR=16kHz; DR=8kHz For more info, see register description.
		5	CMD=STOP Pulse Now For more info, see register description.
		6	CMD=Cancel STOP Pulse For more info, see register description.

Time Stamp Auxiliary TS Nano Seconds Register

The EMAC_TM_AUXSTMP_NSEC register contains the low 32 bits (nanoseconds field) of the auxiliary time stamp.

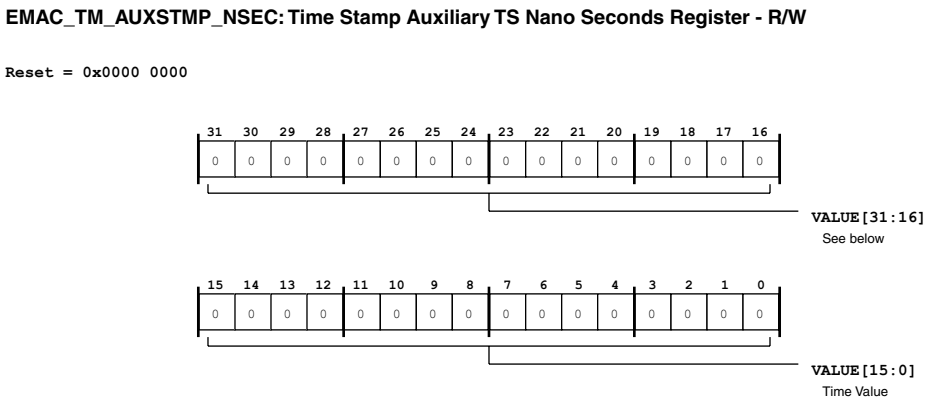


Figure 23-125: EMAC_TM_AUXSTMP_NSEC Register Diagram

Table 23-156: EMAC_TM_AUXSTMP_NSEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Time Value.

Time Stamp Auxiliary TM Seconds Register

The EMAC_TM_AUXSTMP_SEC register contains the low 32 bits of the seconds field of the auxiliary time stamp.

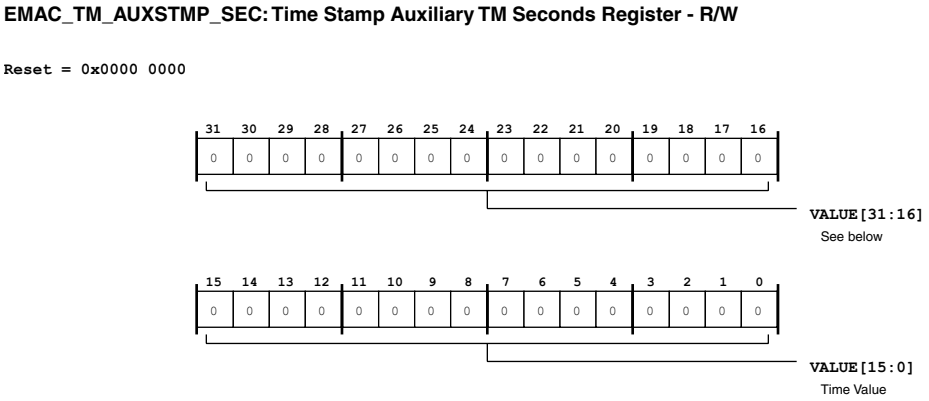


Figure 23-126: EMAC_TM_AUXSTMP_SEC Register Diagram

Table 23-157: EMAC_TM_AUXSTMP_SEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Time Value.

Time Stamp PPS Interval Register

The EMAC_TM_PPSINTVL register contains the interval value for the time between rising edges (period) of PPS output.

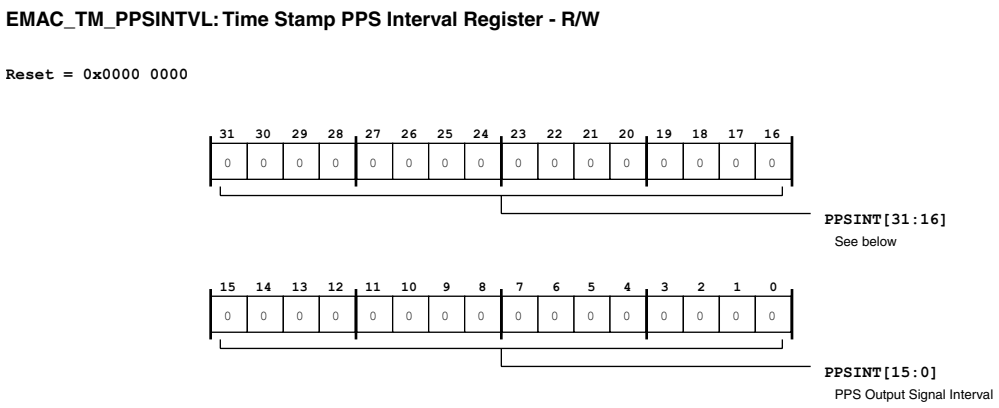


Figure 23-127: EMAC_TM_PPSINTVL Register Diagram

Table 23-158: EMAC_TM_PPSINTVL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	PPSINT	PPS Output Signal Interval. The EMAC_TM_PPSINTVL.PPSINT bits store the interval between the rising edges of PPS signal output in terms of units of sub-second increment value. You need to program one value less than the required interval. For example, if the PTP reference clock is 50 MHz (period of 20ns), and desired interval between rising edges of PPS signal output is 100ns (that is, 5 units of sub-second increment value), then you should program value 4 (5-1) in this register.

PPS Width Register

The EMAC_TM_PPSWIDTH register contains the interval value for the time between a rising and the next falling edge (width) of PPS output.

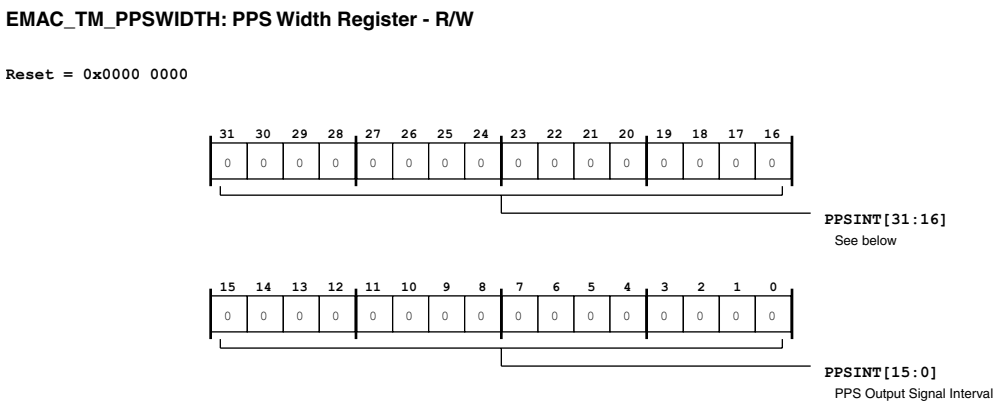


Figure 23-128: EMAC_TM_PPSWIDTH Register Diagram

Table 23-159: EMAC_TM_PPSWIDTH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	PPSINT	PPS Output Signal Interval. The EMAC_TM_PPSWIDTH.PPSINT bits store the interval between the rising edges of PPS signal output in terms of units of sub-second increment value. You need to program one value less than the required interval. For example, if the PTP reference clock is 50 MHz (period of 20ns), and desired interval between rising edges of PPS signal output is 100ns (that is, 5 units of sub-second increment value), then you should program value 4 (5-1) in this register.

DMA Bus Mode Register

The EMAC_DMA_BUSMODE register selects the DMA bus operating modes for EMAC DMA.

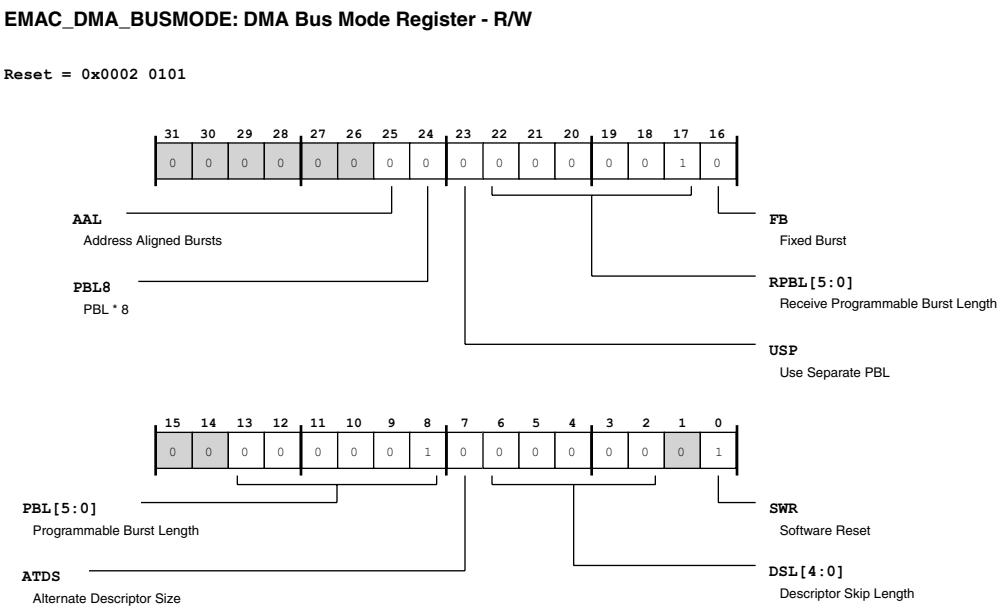


Figure 23-129: EMAC_DMA_BUSMODE Register Diagram

Table 23-160: EMAC_DMA_BUSMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/W)	AAL	Address Aligned Bursts. The EMAC_DMA_BUSMODE.AAL bit, when set high and the FB bit equals 1, directs the SCB interface to generate all bursts aligned to the start address LS bits. If the FB bit is equal to 0, the first burst (accessing the data buffers start address) is not aligned, but subsequent bursts are aligned to the address.
24 (R/W)	PBL8	PBL * 8. The EMAC_DMA_BUSMODE.PBL8 bit, when set high, multiplies the PBL value programmed (bits [22:17] and bits [13:8]) eight times. Therefore, the DMA transfers the data in 8, 16, and 32 beats depending on the PBL value.
23 (R/W)	USP	Use Separate PBL. The EMAC_DMA_BUSMODE.USP bit, when set high, configures the Rx DMA to use the value configured in bits [22:17] as PBL while the PBL value in bits [13:8] is applicable to Tx DMA operations only.

Table 23-160: EMAC_DMA_BUSMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
22:17 (R/W)	RPBL	Receive Programmable Burst Length. The EMAC_DMA_BUSMODE.RPBL bits indicate the maximum number of beats to be transferred in one Rx DMA transaction. This is the maximum value that is used in a single block Read/Write. The Rx DMA always attempts to burst as specified in RPBL every time it starts a Burst transfer on the host bus. RPBL can be programmed with permissible values of 1, 2, 4, 8, 16, and 32. Any other value results in undefined behavior. These bits are valid and applicable only when USP is set high.
16 (R/W)	FB	Fixed Burst. The EMAC_DMA_BUSMODE.FB bit controls whether the SCB Master interface performs fixed burst transfers or not. See the EMAC_DMA_BMMODE.UNDEF bit description for more information.
13:8 (R/W)	PBL	Programmable Burst Length. The EMAC_DMA_BUSMODE.PBL bits indicate the maximum number of beats to be transferred in one DMA transaction. This is the maximum value that is used in a single block Read/Write. The DMA always attempts to burst as specified in PBL each time it starts a Burst transfer on the host bus. Any other value results in undefined behavior. When USP is set high, this PBL value is applicable for Tx DMA transactions only. PBL-max limit = (FIFO size / 2) / 4. PBL-max limit (transmit) = 256 bytes / 2 /4 = 32. PBL-max limit (receive) = 128 bytes / 2 /4 = 16. Note that this PBL is at the DMA end. If PBL= 32 and if BLEN16 is enabled, the DMA automatically splits 32 bursts in to 2 x 16 bursts. If EMAC_DMA_BUSMODE.PBL =8, and if EMAC_DMA_BMMODE.BLEN16 is enabled, the max burst is limited to EMAC_DMA_BMMODE.BLEN8. If EMAC_DMA_BUSMODE.PBL8 bit is set, the programmed PBL value is multiplied by 8 times internally. However, the result cannot be more than the above maximum limits specified above.

Table 23-160: EMAC_DMA_BUSMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W)	ATDS	Alternate Descriptor Size. The EMAC_DMA_BUSMODE.ATDS bit, when set, increases the size of the alternate descriptor to 32 bytes (8 DWORDS). This is required when the Advanced Time Stamp feature or Full IPC Offload Engine is enabled in the receiver. When reset, the descriptor size reverts back to 4 DWORDs (16 bytes). The enhanced descriptor is not required if the Advanced Time Stamp and IPC Full Checksum Offload features are not enabled. In such case, you can use the 16 bytes descriptor to save 4 bytes of memory.
6:2 (R/W)	DSL	Descriptor Skip Length. The EMAC_DMA_BUSMODE.DSL bit specifies the number of 32-bit words to skip between two unchained descriptors. The address skipping starts from the end of current descriptor to the start of next descriptor. When DSL value is equal to zero, then the descriptor table is taken as contiguous by the DMA, in Ring mode.
0 (R/W1S)	SWR	Software Reset. The EMAC_DMA_BUSMODE.SWR bit, when set, directs the MAC DMA Controller to reset all MAC Subsystem internal registers and logic. It is cleared automatically after the reset operation has completed in all of the core clock domains. Read a 0 value in this bit before re-programming any register of the core. Note: The reset operation is completed only when all the resets in all the active clock domains are de-asserted. Therefore, it is essential that all the PHY inputs clocks (applicable for the selected PHY interface) are present for software reset completion.

DMA Tx Poll Demand Register

The EMAC_DMA_TXPOLL register directs the EMAC to poll the transmit descriptor list.

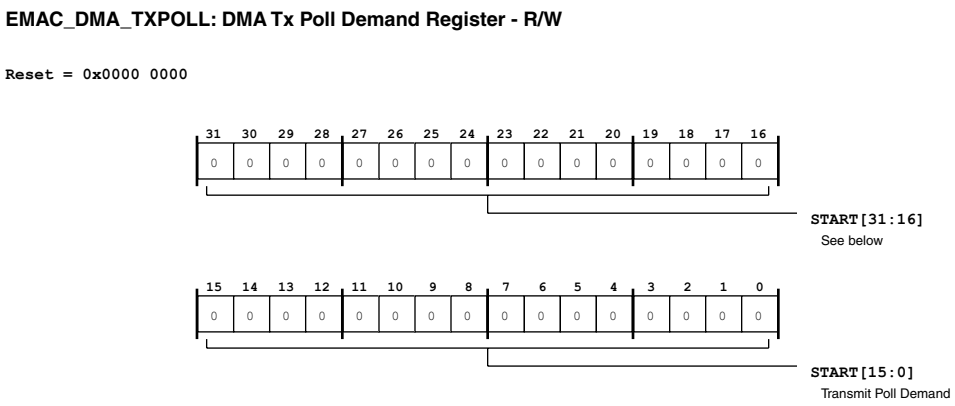


Figure 23-130: EMAC_DMA_TXPOLL Register Diagram

Table 23-161: EMAC_DMA_TXPOLL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	START	Transmit Poll Demand. The EMAC_DMA_TXPOLL.START bits, when written with any value, cause the DMA to read the current descriptor pointed to by EMAC_DMA_TXDSC_CUR register. If that descriptor is not available (owned by application), transmission returns to the Suspend state, and the EMAC_DMA_STAT.TU bit is asserted. If the descriptor is available, transmission resumes.

DMA Rx Poll Demand register

The EMAC_DMA_RXPOLL register directs the EMAC to poll the receive descriptor list.

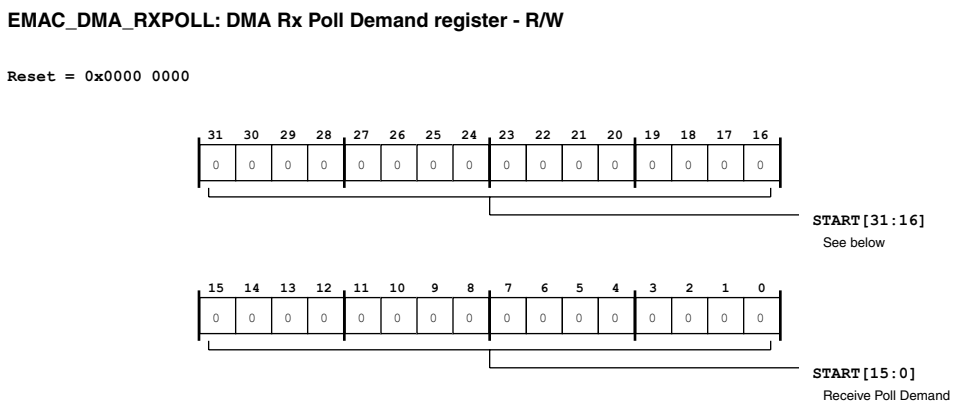


Figure 23-131: EMAC_DMA_RXPOLL Register Diagram

Table 23-162: EMAC_DMA_RXPOLL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	START	Receive Poll Demand. The EMAC_DMA_RXPOLL.START bits, when written with any value, cause the DMA to read the current descriptor pointed to by the EMAC_DMA_RXDSC_CUR register. If that descriptor is not available (owned by application), reception returns to the Suspended state, and the EMAC_DMA_STAT.RU bit is asserted. If the descriptor is available, the Receive DMA returns to the active state.

DMA Rx Descriptor List Address Register

The EMAC_DMA_RXDSC_ADDR register holds the address for the DMA receive descriptor list. Writing to this Register is permitted only when reception is stopped. When stopped, this must be written to before the receive Start command is given. The processor can write to EMAC_DMA_RXDSC_ADDR only when Rx DMA has stopped (EMAC_DMA_OPMODE.SR bit =0). When stopped, it can be written with a new descriptor list address. When the processor sets the EMAC_DMA_OPMODE.SR bit to 1, the DMA takes the newly programmed descriptor base address. If this register is not changed when the EMAC_DMA_OPMODE.SR bit is cleared to 0, the DMA takes the descriptor address where it was stopped earlier.

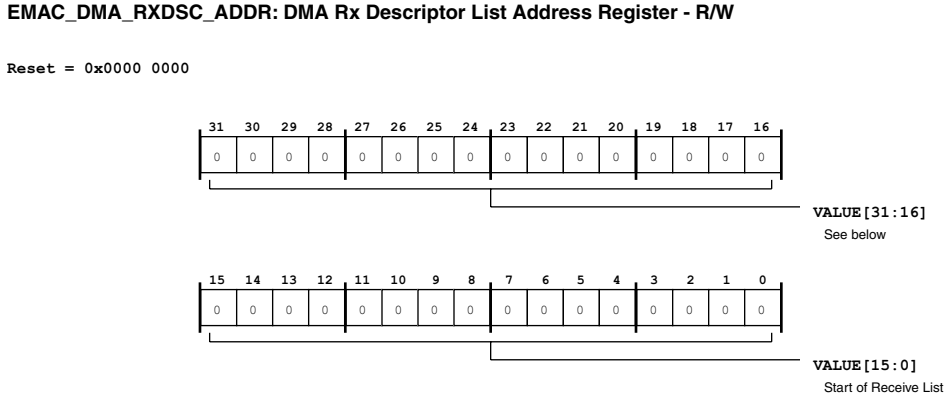


Figure 23-132: EMAC_DMA_RXDSC_ADDR Register Diagram

Table 23-163: EMAC_DMA_RXDSC_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Start of Receive List. The EMAC_DMA_RXDSC_ADDR.VALUE bit field contains the base address of the First Descriptor in the Receive Descriptor list. The LSB bits [1:0] for the 32bit bus width are ignored and are taken as all-zero by the DMA internally. Therefore, these LSB bits are Read-Only (RO).

DMA Tx Descriptor List Address Register

The EMAC_DMA_TXDSC_ADDR register holds the address for the DMA transmit descriptor list. The processor can write to this Register only when Tx DMA has stopped (EMAC_DMA_OPMODE.ST bit =0). When stopped, this can be written with a new descriptor list address. When the processor sets the EMAC_DMA_OPMODE.ST bit to 1, the DMA takes the newly programmed descriptor base address. If this register is not changed when the EMAC_DMA_OPMODE.ST bit is cleared to 0, then the DMA takes the descriptor address where it was stopped earlier.

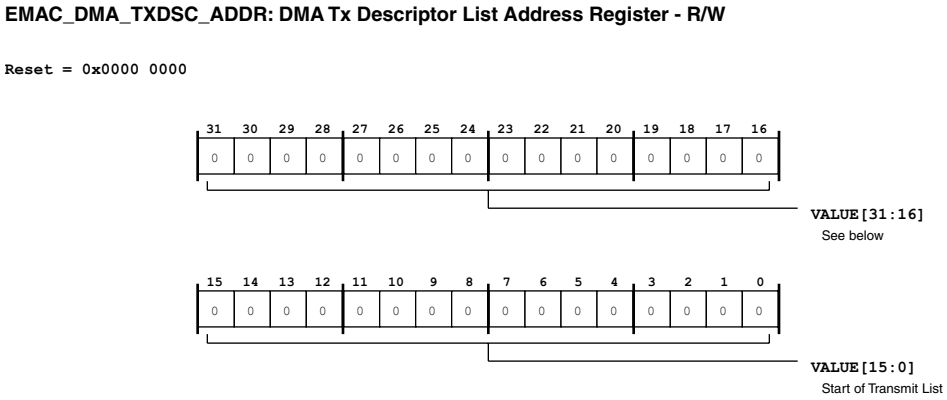


Figure 23-133: EMAC_DMA_TXDSC_ADDR Register Diagram

Table 23-164: EMAC_DMA_TXDSC_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Start of Transmit List. The EMAC_DMA_TXDSC_ADDR.VALUE bit field contains the base address of the First Descriptor in the Transmit Descriptor list. The LSB bits [1:0] for 32bit bus width are ignored and are taken as all-zero by the DMA internally. Therefore, these LSB bits are Read-Only (RO).

DMA Status Register

The EMAC_DMA_STAT register indicates EMAC DMA status.

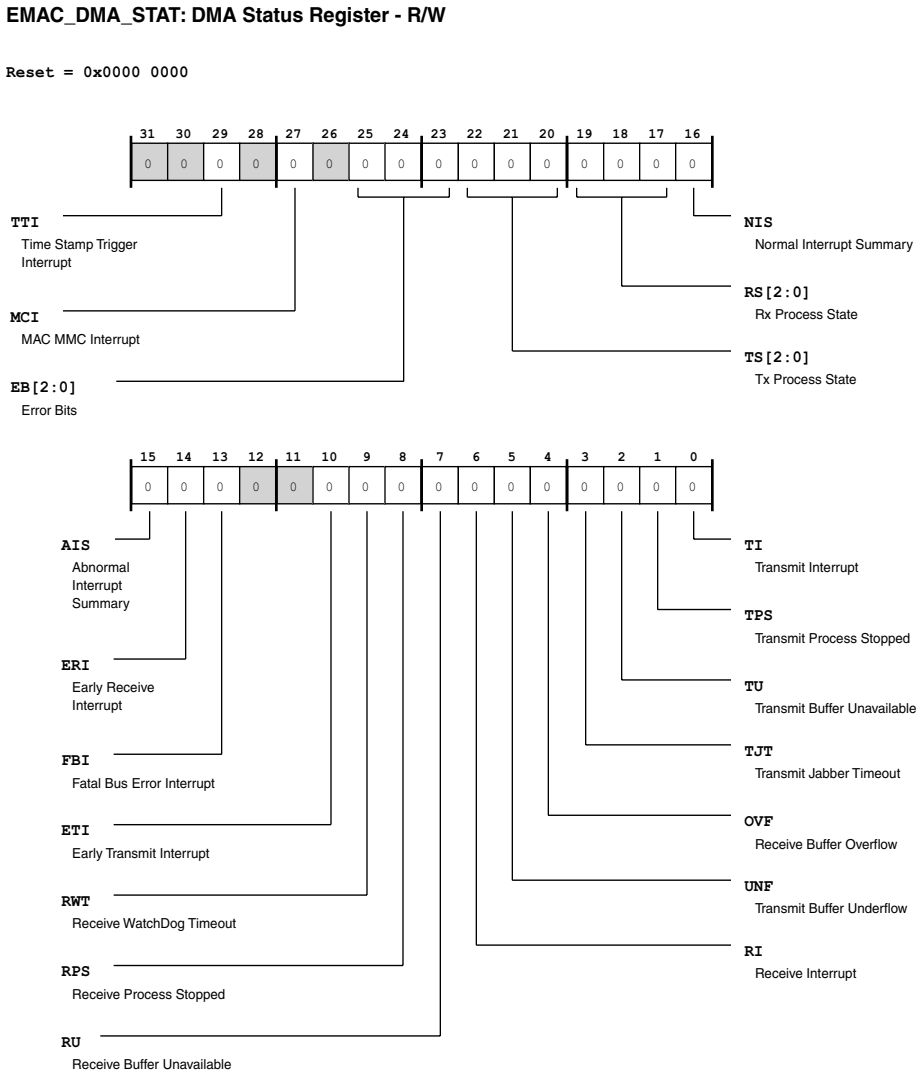


Figure 23-134: EMAC_DMA_STAT Register Diagram

Table 23-165: EMAC_DMA_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29 (R/NW)	TTI	Time Stamp Trigger Interrupt. The EMAC_DMA_STAT.TTI bit indicates an interrupt event in the MAC core's Time Stamp Generator block. The software must read the corresponding registers in the MAC core to get the exact cause of interrupt and clear its source to reset this bit to =0. When this bit is high, the interrupt signal from the MAC is high.

Table 23-165: EMAC_DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
27 (R/NW)	MCI	MAC MMC Interrupt. The EMAC_DMA_STAT.MCI bit reflects an interrupt event in the MMC module of the MAC core. The software must read the corresponding registers in the MAC core to get the exact cause of interrupt and clear the source of interrupt to make this bit as =0. The interrupt signal from the MAC is high when this bit is high.	
25:23 (R/NW)	EB	Error Bits. The EMAC_DMA_STAT.EB bits indicate the type of error that caused a Bus Error (for example, error response on the SCB interface). These bits are valid only when the EMAC_DMA_STAT.FBI bit is set. This field does not generate an interrupt.	
		0	Error during data buffer access, write transfer, Rx DMA
		1	Error during data buffer access, write transfer, Tx DMA
		2	Error during data buffer access, read transfer, Rx DMA
		3	Error during data buffer access, read transfer, Tx DMA
		4	Error during descriptor access, write transfer, Rx DMA
		5	Error during descriptor access, write transfer, Tx DMA
		6	Error during descriptor access, read transfer, Rx DMA
		7	Error during descriptor access, read transfer, Tx DMA

Table 23-165: EMAC_DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
22:20 (R/NW)	TS	Tx Process State. The EMAC_DMA_STAT.TS bits indicate the transmit DMA state. This field does not generate an interrupt.	
		0	Stopped; Reset or Stop Tx Command Issued
		1	Running; Fetching Tx Transfer Descriptor
		2	Running; Waiting for Status
		3	Reading Data from Host Memory Buffer and Queuing It to Tx Buffer
		4	TIME_STAMP Write State
		5	Reserved
		6	Suspended; Tx Descriptor Unavailable or Tx Buffer Underflow
		7	Closing Tx Descriptor
19:17 (R/NW)	RS	Rx Process State. The EMAC_DMA_STAT.RS bits indicate the receive DMA state. This field does not generate an interrupt.	
		0	Stopped: Reset or Stop Rx Command Issued.
		1	Running: Fetching Rx Transfer Descriptor.
		2	Reserved
		3	Running: Waiting for Rx Packet
		4	Suspended: Rx Descriptor Unavailable
		5	Running: Closing Rx Descriptor
		6	TIME_STAMP Write State
		7	Running: Transferring Rx Packet Data from Rx Buffer to Host Memory
16 (R/W1C)	NIS	Normal Interrupt Summary. The value of the EMAC_DMA_STAT.NIS bit field is the logical OR of the following when the corresponding interrupt bits are enabled in DMA Interrupt Enable Register: EMAC_DMA_STAT.TI, EMAC_DMA_STAT.TU, EMAC_DMA_STAT.RI, and EMAC_DMA_STAT.ERI. Only unmasked bits affect the Normal Interrupt Summary bit. This is a sticky bit and must be cleared (by writing a 1 to this bit) each time a corresponding bit that causes EMAC_DMA_STAT.NIS to be set is cleared.	

Table 23-165: EMAC_DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	AIS	Abnormal Interrupt Summary. The value of the EMAC_DMA_STAT.AIS bit field is the logical OR of the following when the corresponding interrupt bits are enabled in DMA Interrupt Enable Register: EMAC_DMA_IEN.TPS, EMAC_DMA_IEN.TJT, EMAC_DMA_IEN.OVF, EMAC_DMA_IEN.UNF, EMAC_DMA_IEN.RU, EMAC_DMA_IEN.RPS, EMAC_DMA_IEN.RWT, EMAC_DMA_IEN.ETI, and EMAC_DMA_IEN.FBI. Only unmasked bits affect the Abnormal Interrupt Summary bit. This is a sticky bit and must be cleared each time a corresponding bit that causes EMAC_DMA_STAT.AIS to be set is cleared.
14 (R/W1C)	ERI	Early Receive Interrupt. The EMAC_DMA_STAT.ERI bit indicates that the DMA had filled the first data buffer of the packet. The EMAC_DMA_STAT.RI bit automatically clears this bit.
13 (R/W1C)	FBI	Fatal Bus Error Interrupt. The EMAC_DMA_STAT.FBI bit indicates that a bus error occurred, as detailed in the EMAC_DMA_STAT.EB field. When this bit is set, the corresponding DMA engine disables all its bus accesses.
10 (R/W1C)	ETI	Early Transmit Interrupt. The EMAC_DMA_STAT.ETI bit indicates that the frame to be transmitted was fully transferred to the MFL Transmit FIFO.
9 (R/W1C)	RWT	Receive WatchDog Timeout. The EMAC_DMA_STAT.RWT bit is asserted when a frame with a length greater than 2,048 bytes is received (10, 240 when Jumbo Frame mode is enabled).
8 (R/W1C)	RPS	Receive Process Stopped. The EMAC_DMA_STAT.RPS bit is asserted when the Receive Process enters the Stopped state.
7 (R/W1C)	RU	Receive Buffer Unavailable. The EMAC_DMA_STAT.RU bit indicates that the Next Descriptor in the Receive List is owned by the application and cannot be acquired by the DMA. Receive Process is suspended. To resume processing Receive descriptors, the application should change the ownership of the descriptor and issue a Receive Poll Demand command. If no Receive Poll Demand is issued, Receive Process resumes when the next recognized incoming frame is received. This bit is set only when the previous Receive Descriptor was owned by the DMA.

Table 23-165: EMAC_DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W1C)	RI	Receive Interrupt. The EMAC_DMA_STAT.RI bit indicates the completion of frame reception. Specific frame status information has been posted in the descriptor. Reception remains in the Running state.
5 (R/W1C)	UNF	Transmit Buffer Underflow. The EMAC_DMA_STAT.UNF bit indicates that the Transmit Buffer had an Underflow during frame transmission. Transmission is suspended and an Underflow Error TDES0[1] is set.
4 (R/W1C)	OVF	Receive Buffer Overflow. The EMAC_DMA_STAT.OVF bit indicates that the Receive Buffer had an Overflow during frame reception. If the partial frame is transferred to application, the overflow status is set in RDES0[11].
3 (R/W1C)	TJT	Transmit Jabber Timeout. The EMAC_DMA_STAT.TJT bit indicates that the Transmit Jabber Timer expired, meaning that the transmitter had been excessively active. The transmission process is aborted and placed in the Stopped state. This causes the Transmit Jabber Timeout TDES0[14] flag to assert.
2 (R/W1C)	TU	Transmit Buffer Unavailable. The EMAC_DMA_STAT.TU bit indicates that the Next Descriptor in the Transmit List is owned by the application and cannot be acquired by the DMA. Transmission is suspended. The value in the EMAC_DMA_STAT.TS bits explain the Transmit Process state transitions. To resume processing transmit descriptors, the application should change the ownership of the bit of the descriptor and then issue a Transmit Poll Demand command.
1 (R/W1C)	TPS	Transmit Process Stopped. The EMAC_DMA_STAT.TPS bit is set when the transmission is stopped.
0 (R/W1C)	TI	Transmit Interrupt. The EMAC_DMA_STAT.TI bit indicates that frame transmission is finished and TDES1[31] is set in the First Descriptor.

DMA Operation Mode Register

The EMAC_DMA_OPMODE register selects receive and transmit DMA operating modes.

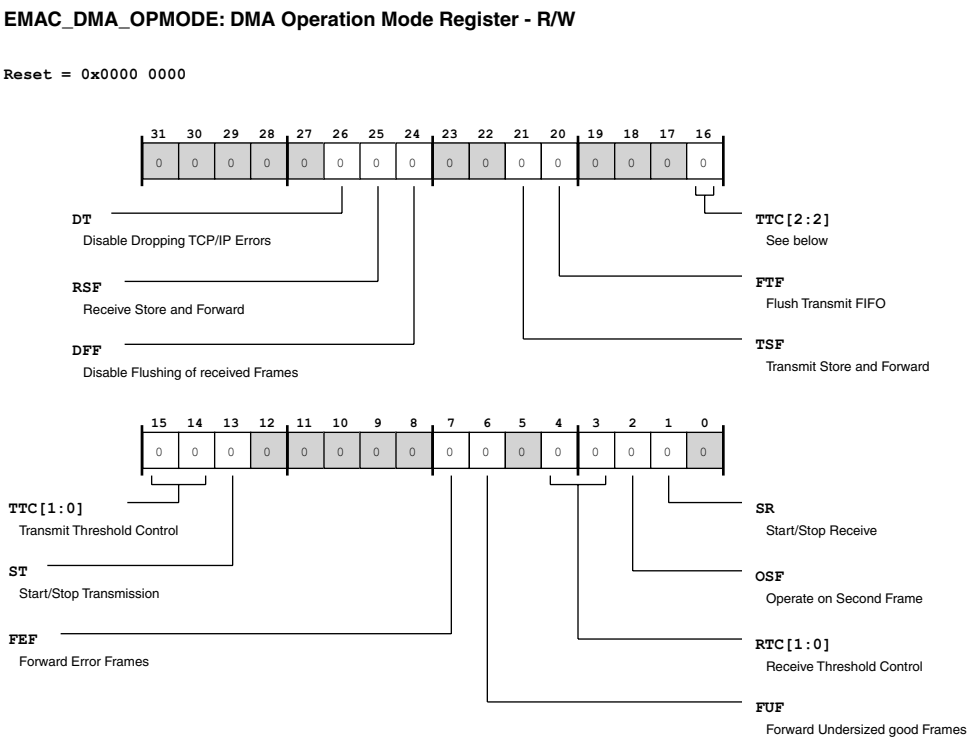


Figure 23-135: EMAC_DMA_OPMODE Register Diagram

Table 23-166: EMAC_DMA_OPMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
26 (R/W)	DT	Disable Dropping TCP/IP Errors. The EMAC_DMA_OPMODE.DT bit, when set, directs the core not to drop frames that only have errors detected by the Receive Checksum Offload engine. Such frames do not have any errors (including FCS error) in the Ethernet frame received by the MAC but have errors in the encapsulated payload only. When this bit is reset, all error frames are dropped if the EMAC_DMA_OPMODE.FEF bit is reset.
25 (R/W)	RSF	Receive Store and Forward. The EMAC_DMA_OPMODE.RSF bit, when set, directs the MFL only to read a frame from the Rx FIFO after the complete frame has been written to it, ignoring the EMAC_DMA_OPMODE.RTC bits. When this bit is reset, the Rx FIFO operates in threshold mode, subject to the threshold specified by the EMAC_DMA_OPMODE.RTC bits.
24 (R/W)	DFF	Disable Flushing of received Frames. The EMAC_DMA_OPMODE.DFF bit, when set, directs the Rx DMA not to flush any frames because of the unavailability of receive descriptors/ buffers as it does normally when this bit is reset.

Table 23-166: EMAC_DMA_OPMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	TSF	Transmit Store and Forward. The EMAC_DMA_OPMODE.TSF bit, when set, starts transmission when a full frame resides in the MFL Transmit FIFO. When this bit is set, the TTC values specified in Register 6[16:14] are ignored. This bit should be changed only when transmission is stopped.	
20 (R/W)	FTF	Flush Transmit FIFO. The EMAC_DMA_OPMODE.FTF bit, when set, directs the transmit FIFO controller logic to reset to its default values and thus all data in the Tx FIFO is lost/flushed. This bit is cleared internally when the flushing operation is completed fully. The Operation Mode register should not be written to until this bit is cleared. The data which is already accepted by the MAC transmitter is not flushed. It is scheduled for transmission and results in underflow and runt frame transmission. Note: The flush operation completes only after emptying the Tx FIFO of its contents and all the pending Transmit Status of the transmitted frames are accepted by the host. In order to complete this flush operation, the PHY transmit clock is required to be active.	
16:14 (R/W)	TTC	Transmit Threshold Control. The EMAC_DMA_OPMODE.TTC bits control the threshold level of the MFL Transmit FIFO. Transmission starts when the frame size within the MFL Transmit FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are also transmitted. These bits are used only when the EMAC_DMA_OPMODE.TSF bit is reset. The value =011 is not used.	
		0	64
		1	128
		2	192
		3	256
		4	40
		5	32
		6	24
		7	16

Table 23-166: EMAC_DMA_OPMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13 (R/W)	ST	Start/Stop Transmission. The EMAC_DMA_OPMODE.ST bit, when set, places transmission in the Running state, and the DMA checks the Transmit List at the current position for a frame to be transmitted. Descriptor acquisition is attempted either from the current position in the list, which is the Transmit List Base Address set by Transmit Descriptor List Address, or from the position retained when transmission was stopped previously. If the current descriptor is not owned by the DMA, transmission enters the Suspended state, and the EMAC_DMA_STAT.TU bit is set. The Start Transmission command is effective only when transmission is stopped. If the command is issued before setting the EMAC_DMA_TXDSC_CUR address register, then the DMA behavior is unpredictable. When this bit is reset, the transmission process is placed in the Stopped state after completing the transmission of the current frame. The Next Descriptor position in the Transmit List is saved, and becomes the current position when transmission is restarted. The stop transmission command is effective only when the transmission of the current frame is complete or the transmission is in the Suspended state.
7 (R/W)	FEF	Forward Error Frames. The EMAC_DMA_OPMODE.FEF bit, when reset, directs the Rx FIFO to drop frames with error status (CRC error, collision error, giant frame, watchdog timeout, overflow). However, if the frames start byte (write) pointer is already transferred to the read controller side (in Threshold mode), then the frames are not dropped. When EMAC_DMA_OPMODE.FEF bit is set, all frames except runt error frames are forwarded to the DMA. But when Rx FIFO overflows when a partial frame is written, then such frames are dropped even when EMAC_DMA_OPMODE.FEF is set.
6 (R/W)	FUF	Forward Undersized good Frames. The EMAC_DMA_OPMODE.FUF bit, when set, directs the Rx FIFO to forward Undersized frames (frames with no Error and length less than 64 bytes) including pad-bytes and CRC). When reset, the Rx FIFO drops all frames of less than 64 bytes, unless it is already transferred because of lower value of Receive Threshold (for example, EMAC_DMA_OPMODE.RTC =01).

Table 23-166: EMAC_DMA_OPMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4:3 (R/W)	RTC	Receive Threshold Control. The EMAC_DMA_OPMODE.RTC bits control the threshold level of the MFL Receive FIFO. Transfer (request) to DMA starts when the frame size within the MFL Receive FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are transferred automatically. These bits are valid only when the EMAC_DMA_OPMODE.RSF bit is zero, and are ignored when the EMAC_DMA_OPMODE.RSF bit is set to 1. The value =11 is not used.	
		0	64
		1	32
		2	96
		3	128
2 (R/W)	OSF	Operate on Second Frame. The EMAC_DMA_OPMODE.OSF bit, when set, instructs the DMA to process a second frame of Transmit data even before status for first frame is obtained.	
1 (R/W)	SR	Start/Stop Receive. The EMAC_DMA_OPMODE.SR bit, when set, places the Receive process in the Running state. The DMA attempts to acquire the descriptor from the Receive list and processes incoming frames. Descriptor acquisition is attempted from the current position in the list, which is the address set by DMA Receive Descriptor List Address or the position retained when the Receive process was previously stopped. If no descriptor is owned by the DMA, reception is suspended, and the EMAC_DMA_STAT.RU bit is set. The Start Receive command is effective only when reception has stopped. If the command was issued before setting EMAC_DMA_RXDSC_CURaddress register, DMA behavior is unpredictable. When this bit is cleared, Rx DMA operation is stopped after the transfer of the current frame. The next descriptor position in the Receive list is saved and becomes the current position after the Receive process is restarted. The Stop Receive command is effective only when the Receive process is in either the Running (waiting for receive packet) or in the Suspended state.	

DMA Interrupt Enable Register

The EMAC_DMA_IEN register enables (unmasks) EMAC DMA interrupts.

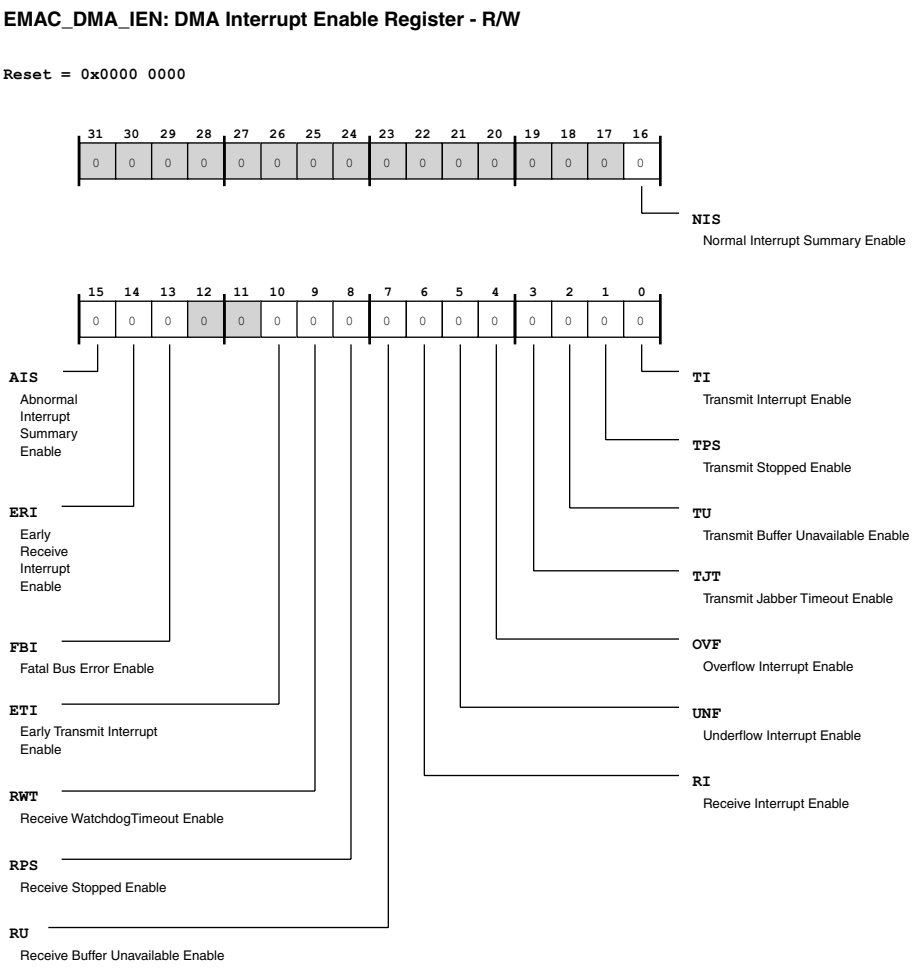


Figure 23-136: EMAC_DMA_IEN Register Diagram

Table 23-167: EMAC_DMA_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	NIS	Normal Interrupt Summary Enable. The EMAC_DMA_IEN.NIS bit, when set, enables a normal interrupt. When this bit is reset, a normal interrupt is disabled. This bit enables the following bits: EMAC_DMA_STAT.TI, EMAC_DMA_STAT.TU, EMAC_DMA_STAT.RI, and EMAC_DMA_STAT.ERI.
15 (R/W)	AIS	Abnormal Interrupt Summary Enable. The EMAC_DMA_IEN.AIS bit, when set, enables an abnormal interrupt. When this bit is reset, an Abnormal Interrupt is disabled. This bit enables the following bits: EMAC_DMA_STAT.TPS, EMAC_DMA_STAT.TJT, EMAC_DMA_STAT.OVF, EMAC_DMA_STAT.RU, EMAC_DMA_STAT.RPS, EMAC_DMA_STAT.RWT, EMAC_DMA_STAT.ETI, and EMAC_DMA_STAT.FBI.

Table 23-167: EMAC_DMA_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W)	ERI	Early Receive Interrupt Enable. The EMAC_DMA_IEN.ERI bit, when set (and with EMAC_DMA_IEN.NIS =1), enables the Early Receive Interrupt. When this bit is reset, Early Receive Interrupt is disabled.
13 (R/W)	FBI	Fatal Bus Error Enable. The EMAC_DMA_IEN.FBI bit, when set (and with EMAC_DMA_IEN.AIS =1), enables the Fatal Bus Error Interrupt. When this bit is reset, Fatal Bus Error Enable Interrupt is disabled.
10 (R/W)	ETI	Early Transmit Interrupt Enable. The EMAC_DMA_IEN.ETI bit, when this bit is set (and with EMAC_DMA_IEN.AIS =1), enables the Early Transmit Interrupt. When this bit is reset, Early Transmit Interrupt is disabled.
9 (R/W)	RWT	Receive WatchdogTimeout Enable. The EMAC_DMA_IEN.RWT bit, when set (and with EMAC_DMA_IEN.AIS =1), enables the Receive Watchdog Timeout Interrupt. When this bit is reset, Receive Watchdog Timeout Interrupt is disabled.
8 (R/W)	RPS	Receive Stopped Enable. The EMAC_DMA_IEN.RPS bit, when set (and with EMAC_DMA_IEN.AIS =1), enables the Receive Stopped Interrupt is enabled. When this bit is reset, Receive Stopped Interrupt is disabled.
7 (R/W)	RU	Receive Buffer Unavailable Enable. The EMAC_DMA_IEN.RU bit, when set (and with EMAC_DMA_IEN.AIS =1), enables the Receive Buffer Unavailable Interrupt. When this bit is reset, the Receive Buffer Unavailable Interrupt is disabled.
6 (R/W)	RI	Receive Interrupt Enable. The EMAC_DMA_IEN.RI bit, when set (and with EMAC_DMA_IEN.NIS =1), enables the Receive Interrupt. When this bit is reset, Receive Interrupt is disabled.
5 (R/W)	UNF	Underflow Interrupt Enable. The EMAC_DMA_IEN.UNF bit, when set (and with EMAC_DMA_IEN.AIS =1), enables the Transmit Underflow Interrupt. When this bit is reset, Underflow Interrupt is disabled.
4 (R/W)	OVF	Overflow Interrupt Enable. The EMAC_DMA_IEN.OVF bit, when set (and with EMAC_DMA_IEN.AIS =1), enables the Receive Overflow Interrupt. When this bit is reset, Overflow Interrupt is disabled.

Table 23-167: EMAC_DMA_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	TJT	Transmit Jabber Timeout Enable. The EMAC_DMA_IEN.TJT bit, when set (and with EMAC_DMA_IEN.AIS =1), enables the Transmit Jabber Timeout Interrupt. When this bit is reset, Transmit Jabber Timeout Interrupt is disabled.
2 (R/W)	TU	Transmit Buffer Unavailable Enable. The EMAC_DMA_IEN.TU bit, when set (and with EMAC_DMA_IEN.NIS =1), enables the Transmit Buffer Unavailable Interrupt. When this bit is reset, Transmit Buffer Unavailable Interrupt is disabled.
1 (R/W)	TPS	Transmit Stopped Enable. The EMAC_DMA_IEN.TPS bit, when set (and with EMAC_DMA_IEN.AIS =1), enables the Transmission Stopped Interrupt. When this bit is reset, Transmission Stopped Interrupt is disabled.
0 (R/W)	TI	Transmit Interrupt Enable. The EMAC_DMA_IEN.TI bit, when set (and with EMAC_DMA_IEN.NIS =1), enables the Transmit Interrupt. When this bit is reset, Transmit Interrupt is disabled.

DMA Missed Frame Register

The EMAC_DMA_MISS_FRM register contains counters for EMAC DMA missed frames and buffer overflows.

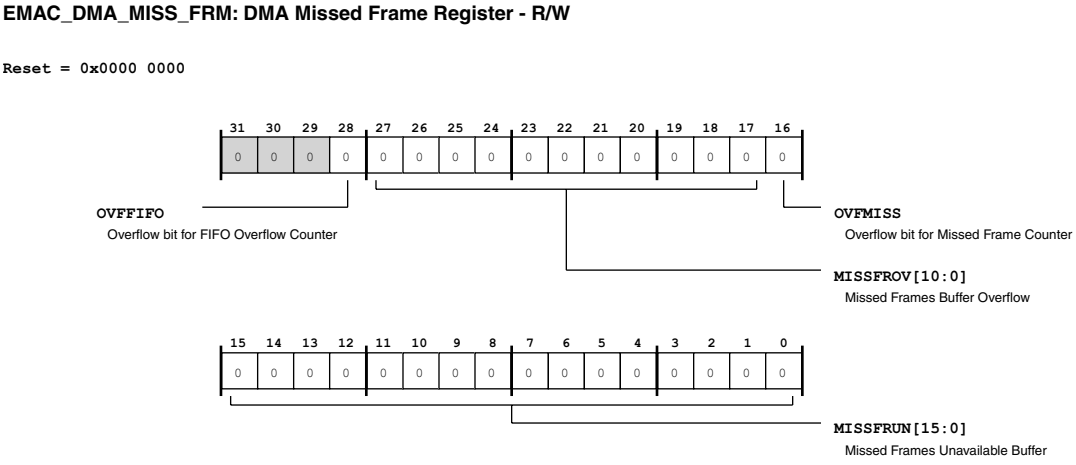


Figure 23-137: EMAC_DMA_MISS_FRM Register Diagram

Table 23-168: EMAC_DMA_MISS_FRM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
28 (RC/NW)	OVFFIFO	Overflow bit for FIFO Overflow Counter. The EMAC_DMA_MISS_FRM.OVFFIFO bit holds the overflow bit for FIFO Overflow Counter.
27:17 (RC/NW)	MISSFROV	Missed Frames Buffer Overflow. The EMAC_DMA_MISS_FRM.MISSFROV bits indicate the number of frames missed by the application due to buffer overflow.
16 (RC/NW)	OVFMISS	Overflow bit for Missed Frame Counter. The EMAC_DMA_MISS_FRM.OVFMISS bit holds the overflow bit for the Missed Frame Counter.
15:0 (RC/NW)	MISSFRUN	Missed Frames Unavailable Buffer. The EMAC_DMA_MISS_FRM.MISSFRUN bits indicate the number of frames missed by the controller because of the Application Receive Buffer being unavailable.

DMA Rx Interrupt Watch Dog Register

The EMAC_DMA_RXIWDOG register contains the timeout value for the EMAC DMA receive interrupt watch dog timer.

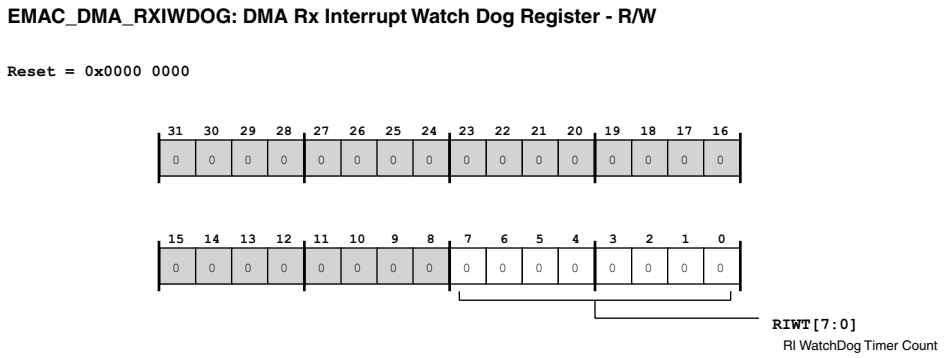


Figure 23-138: EMAC_DMA_RXIWDOG Register Diagram

Table 23-169: EMAC_DMA_RXIWDOG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	RIWT	RI WatchDog Timer Count. The EMAC_DMA_RXIWDOG.RIWT bit field indicates the number of system clock cycles multiplied by 256 for which the watchdog timer is set. The watchdog timer gets triggered with the programmed value after the Rx DMA completes the transfer of a frame for which the RI status bit is not set because of the setting in the corresponding descriptor RDES1[31]. When the watch-dog timer runs out, the RI bit is set and the timer is stopped. The watchdog timer is reset when EMAC_DMA_STAT.RI bit is set high because of automatic setting of EMAC_DMA_STAT.RI as per RDES1[31] of any received frame.

DMA SCB Bus Mode Register

The EMAC_DMA_BMMODE register selects EMAC DMA system cross bar bus mode features.

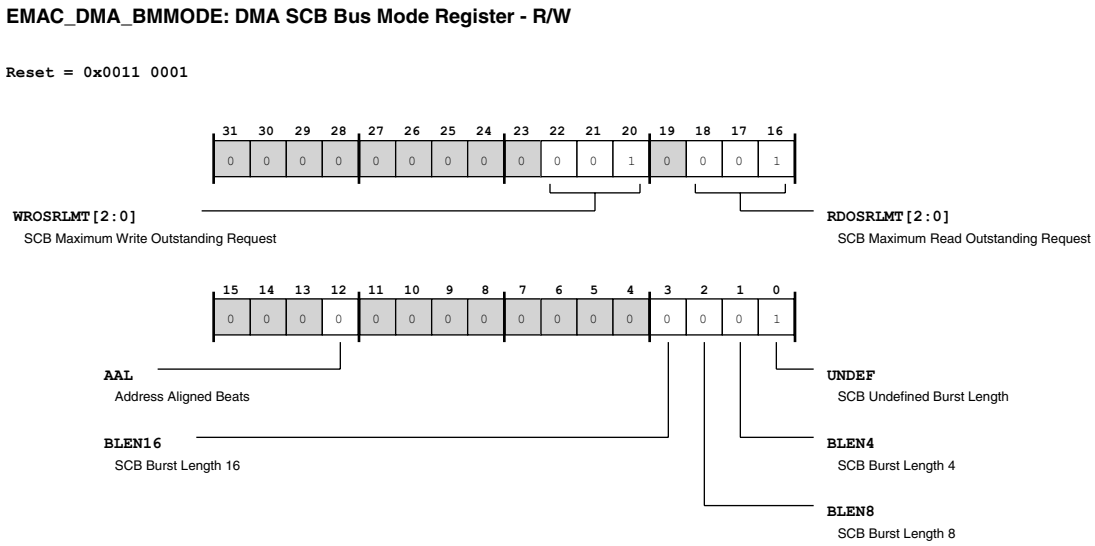


Figure 23-139: EMAC_DMA_BMMODE Register Diagram

Table 23-170: EMAC_DMA_BMMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
22:20 (R/W)	WROSRLMT	SCB Maximum Write Outstanding Request. The EMAC_DMA_BMMODE.WROSRLMT bit field's value limits the maximum outstanding request on the SCB write interface. Maximum outstanding requests = WR_OSR_LMT+1. EMAC-SCB supports up to 4 outstanding write requests.
18:16 (R/W)	RDOSRLMT	SCB Maximum Read Outstanding Request. The EMAC_DMA_BMMODE.RDOSRLMT bit field's value limits the maximum outstanding request on the SCB read interface. Maximum outstanding requests = RD_OSR_LMT+1. EMAC-SCB supports up to 4 outstanding read requests.
12 (R/NW)	AAL	Address Aligned Beats. The EMAC_DMA_BMMODE.AAL bit (read-only) reflects the state of the EMAC_DMA_BUSMODE.AAL bit. When this bit is set to 1, EMAC-SCB performs address-aligned burst transfers on both read and write channels.
3 (R/W)	BLEN16	SCB Burst Length 16. The EMAC_DMA_BMMODE.BLEN16 bit, when set (or when EMAC_DMA_BMMODE.UNDEF is set to 1), directs the EMAC-SCB to select a burst length of 16 on the SCB master interface.
2 (R/W)	BLEN8	SCB Burst Length 8. The EMAC_DMA_BMMODE.BLEN8 bit, when set (or when EMAC_DMA_BMMODE.UNDEF is set to 1), directs the EMAC-SCB to select a burst length of 8 on the SCB master interface.
1 (R/W)	BLEN4	SCB Burst Length 4. The EMAC_DMA_BMMODE.BLEN4 bit, when set (or when EMAC_DMA_BMMODE.UNDEF is set to 1), directs the EMAC-SCB to select a burst length of 4 on the SCB master interface.
0 (R/NW)	UNDEF	SCB Undefined Burst Length. The EMAC_DMA_BMMODE.UNDEF bit (read-only) indicates the complement (invert) value of EMAC_DMA_BUSMODE.FB bit . When this bit is set to 1, the EMAC-SCB is allowed to perform any burst length equal to or below the maximum allowed burst length as programmed in bits[3:1]. When this bit is set to 0, the EMAC-SCB is allowed to perform only fixed burst lengths as indicated by 16/8/4, or a burst length of 1.

DMA SCB Status Register

The EMAC_DMA_BMSTAT register indicates EMAC DMA system cross bar status.

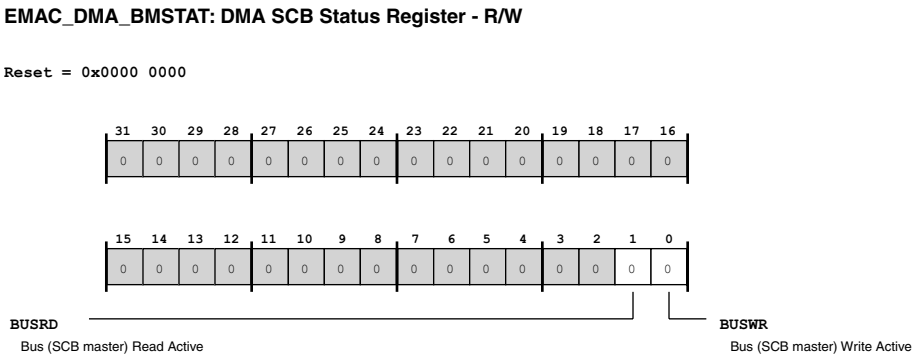


Figure 23-140: EMAC_DMA_BMSTAT Register Diagram

Table 23-171: EMAC_DMA_BMSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/NW)	BUSRD	Bus (SCB master) Read Active. The EMAC_DMA_BMSTAT.BUSRD bit, when high, indicates that SCB Master's read channel is active and transferring data.
0 (R/NW)	BUSWR	Bus (SCB master) Write Active. The EMAC_DMA_BMSTAT.BUSWR bit, when high, indicates that SCB Master's write channel is active and transferring data.

DMA Tx Descriptor Current Register

The EMAC_DMA_TXDSC_CUR register contains the current DMA transmit descriptor.

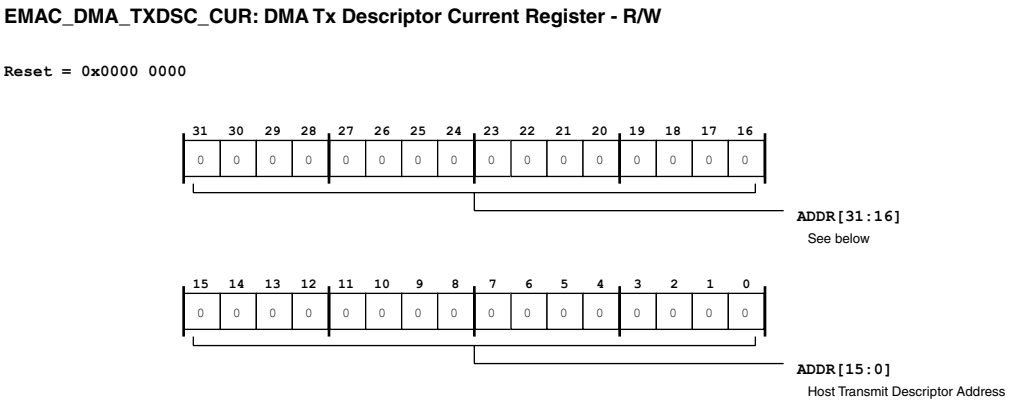


Figure 23-141: EMAC_DMA_TXDSC_CUR Register Diagram

Table 23-172: EMAC_DMA_TXDSC_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Transmit Descriptor Address. The EMAC_DMA_TXDSC_CUR.ADDR bit field points to the start address of the current Transmit Descriptor read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Rx Descriptor Current Register

The EMAC_DMA_RXDSC_CUR register contains the current DMA receive descriptor.

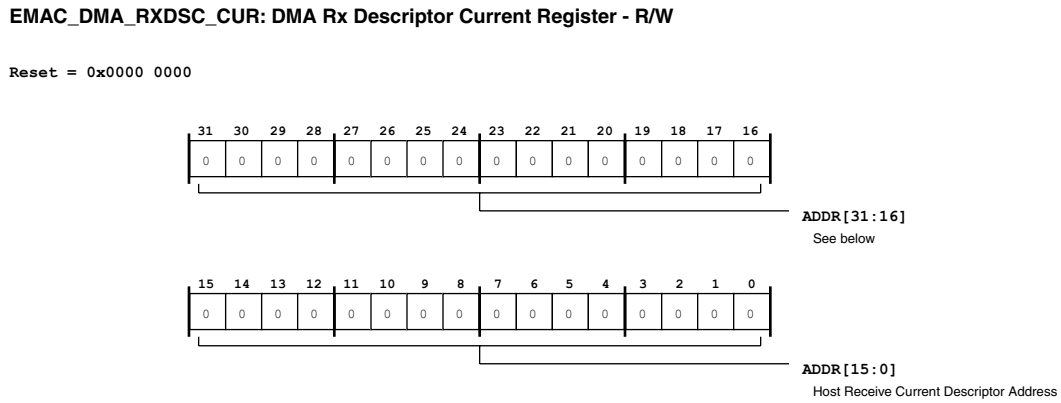


Figure 23-142: EMAC_DMA_RXDSC_CUR Register Diagram

Table 23-173: EMAC_DMA_RXDSC_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Receive Current Descriptor Address. The EMAC_DMA_RXDSC_CUR.ADDR bit field points to the start address of the current Receive Descriptor read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Tx Buffer Current Register

The EMAC_DMA_TXBUF_CUR register holds the pointer to the current transmit DMA buffer.

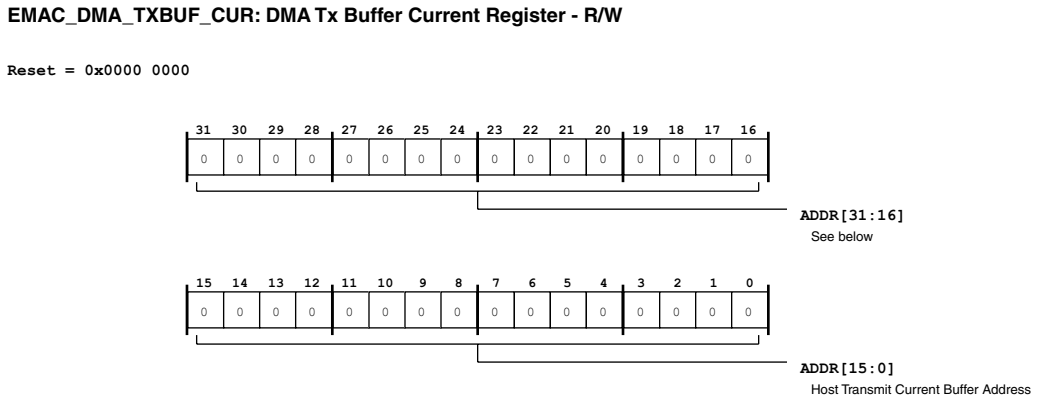


Figure 23-143: EMAC_DMA_TXBUF_CUR Register Diagram

Table 23-174: EMAC_DMA_TXBUF_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Transmit Current Buffer Address. The EMAC_DMA_TXBUF_CUR.ADDR bit field points to the current Transmit Buffer Address being read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

DMA Rx Buffer Current Register

The EMAC_DMA_RXBUF_CUR register holds the pointer to the current receive DMA buffer.

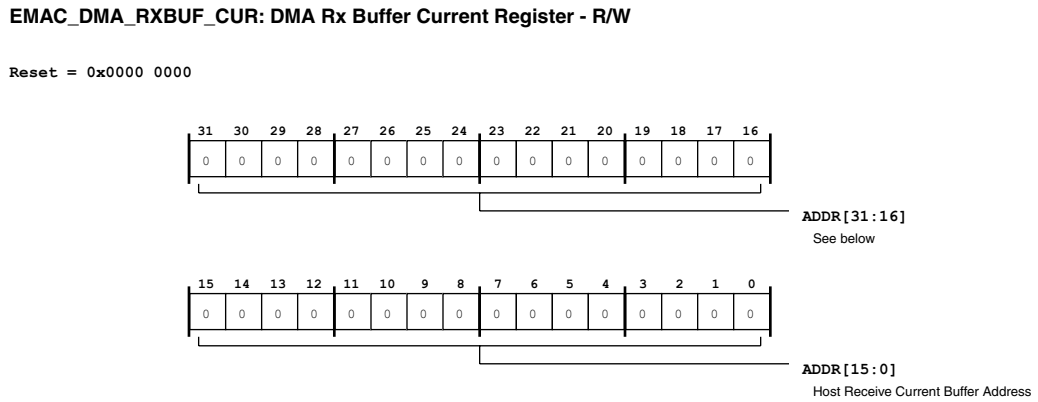


Figure 23-144: EMAC_DMA_RXBUF_CUR Register Diagram

Table 23-175: EMAC_DMA_RXBUF_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Host Receive Current Buffer Address. The EMAC_DMA_RXBUF_CUR.ADDR bit field points to the current Receive Buffer address being read by the DMA. Pointer updated by DMA during operation. Cleared on Reset.

24 Removable Storage Interface (RSI)

The removable storage interface (RSI) controller is a fast, synchronous peripheral that uses various protocols to communicate with MMC, SD, and SDIO cards. The RSI is compatible with the following protocols.

- MMC (Multimedia Card) bus protocol
- SD (Secure Digital) bus protocol
- SDIO (Secure Digital Input Output) bus protocol

All of these storage solutions use similar interface protocols. The main difference between MMC and SD support is the initialization sequence. The main difference between SD and SDIO support is the use of interrupt and read wait signals for SDIO.

NOTE: The RSI does not support the SPI bus protocol

RSI Features

The RSI includes the following features.

- Support for a single SD or SDIO card
- Support for a single MMC device (removable or embedded)
- Support for 1- and 4-bit SD modes (SPI mode is not supported)
- Support for 1-, 4-, and 8-bit MMC modes (SPI mode is not supported)
- Programmable clock frequency generated from SCLK
- Card detection capabilities
- SDIO interrupt and read wait features
- High-capacity card support such as SDHC implemented within software
- 512-bit transmit/receive FIFO
- DMA channel with 32-bit DMA access bus

Table 24-1: RSI Specifications

Feature	Availability
Protocol	

Table 24-1: RSI Specifications (Continued)

Feature	Availability
Master-Capable	Yes
Slave-Capable	No
Transmission Simplex	Yes
Transmission Half-Duplex	Yes
Transmission Full-Duplex	No
Access Type	
Data Buffer	Yes
Core Data Access	Yes
DMA Data Access	Yes
DMA Channels	1
DMA Descriptor	Yes
Boot Capable	Yes
Local Memory	No
Clock Operation	SCLK/2

RSI Functional Description

The following sections provide details on the RSI functionality.

ADSP-BF60x RSI Register List

The removable storage interface (RSI) provides an interface to multimedia cards (MMC), secure digital memory cards (SD), secure digital input/output cards (SDIO). A set of registers govern RSI operations. For more information on RSI functionality, see the RSI register descriptions.

Table 24-2: ADSP-BF60x RSI Register List

Name	Description
RSI_CTL	Control Register
RSI_ARG	Argument Register
RSI_CMD	Command Register
RSI_RESP_CMD	Response Command Register

Table 24-2: ADSP-BF60x RSI Register List (Continued)

Name	Description
RSI_RESP0	Response 0 Register
RSI_RESP1	Response 1 Register
RSI_RESP2	Response 2 Register
RSI_RESP3	Response 3 Register
RSI_DATA_TMR	Data Timer Register
RSI_DATA_LEN	Data Length Register
RSI_DATA_CTL	Data Control Register
RSI_DATA_CNT	Data Count Register
RSI_XFRSTAT	Transfer Status Register
RSI_XFRSTAT_CLR	Transfer Status Clear Register
RSI_XFR_IMSK0	Transfer Interrupt 0 Mask Register
RSI_XFR_IMSK1	Transfer Interrupt 1 Mask Register
RSI_FIFO_CNT	FIFO Counter Register
RSI_BOOT_TCNTR	Boot Timing Counter Register
RSI_BACK_TOUT	Boot Acknowledge Timeout Register
RSI_SLP_WKUP_TOUT	Sleep Wakeup Timeout Register
RSI_BLKSZ	Block Size Register
RSI_FIFO	Data FIFO Register
RSI_STAT0	Exception Status Register
RSI_IMSK0	Exception Mask Register
RSI_CFG	Configuration Register
RSI_RD_WAIT	Read Wait Enable Register

Table 24-2: ADSP-BF60x RSI Register List (Continued)

Name	Description
RSI_PID0	Peripheral ID 0 Register
RSI_PID1	Peripheral ID 1 Register
RSI_PID2	Peripheral ID 2 Register
RSI_PID3	Peripheral ID 3 Register

ADSP-BF60x RSI Interrupt List

Table 24-3: ADSP-BF60x RSI Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
RSI0 DMA Channel	61	10	LEVEL
RSI0 Interrupt 0	62		LEVEL
RSI0 Interrupt 1	63		LEVEL

ADSP-BF60x RSI Trigger List

Table 24-4: ADSP-BF60x RSI Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
RSI0 DMA Channel	30	PULSE/EDGE

Table 24-5: ADSP-BF60x RSI Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
RSI0 DMA Channel	30	

RSI Block Diagram

The figure shows the functional blocks within the RSI.

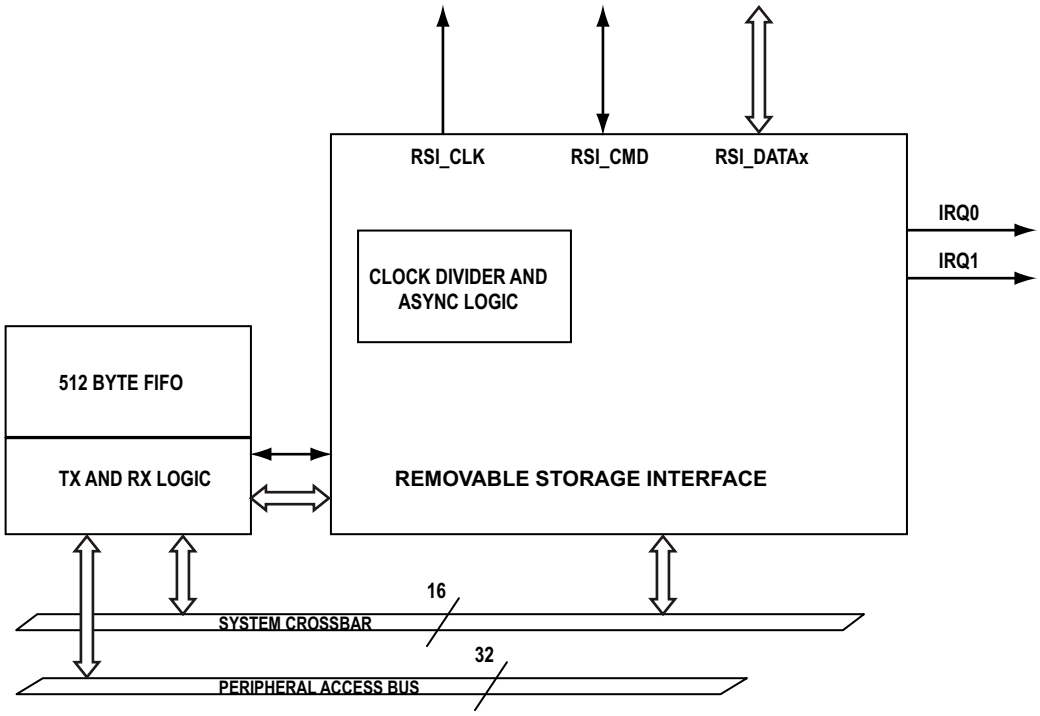


Figure 24-1: RSI Block Diagram

RSI Architectural Concepts

The following sections describe the functions and features of the RSI controller as well as the MMC, SD, and SDIO protocols. For detailed information on timing parameters and protocol requirements, refer to the corresponding processor data sheet and the following standards and specifications.

- MMCA System Specification
- JESD84 series of JEDEC standards
- SD Specifications Part 1 Physical Layer Specification
- SD Specifications Part 1 Physical Layer Simplified Specification
- SD Specifications Part E1 SDIO Specification

Communication is through a master and slave configuration, where the RSI is the master and the card is the slave device. The RSI communicates with the device using a message-based bus protocol in which the host sends commands serially using the RSI_CMD signal. Some commands require that the card provide a response back to the host. This response is also sent serially by the RSI_CMD signal.

Data transfers, both to and from the card, occur using the data signals. The number of data lines used for the data transfer can be configured to 1, 4, or 8 using RSI_D0, RSI_D3 – RSI_D0, or RSI_D7 – RSI_D0. All RSI_CMD and RSI_D7 – RSI_D0 transfers are synchronous with RSI_CLK.

Cyclic redundancy codes (CRC) are used to protect commands, responses and data transfers from transmission errors. A CRC7 code is generated for every command sent by the host and for almost every response returned by the card on the RSI_CMD signal. A CRC16 code is used on the RSI_D7 – RSI_D0 signals to protect block data transfers. In 4- and 8-bit bus configurations, CRC16 is calculated for each individual data signal.

Signal Descriptions

The RSI is a 10 pin interface out of which one pin is used for clock, one pin for command and the rest of the 8 pins are used for data. The table **RSI Signal Descriptions (General)** shows a general functional description of various RSI signals.

Table 24-6: RSI Signal Descriptions (General)

Signal Name	Signal Description
RSI_CLK	The clock signal applied to the card from the RSI. All command and data signal transfers are synchronous to this clock. The frequency can vary between zero and the maximum clock frequency. Refer to the respective processor data sheet for the maximum supported clock frequency.
RSI_CMD	A bi-directional command signal used for command transfer and card initialization. The RSI uses this signal to send commands to the cards, and the card uses the signal to send responses back to the RSI. This signal can be configured for both push-pull mode and open-drain mode, but only MMC cards support the open-drain mode. The open-drain mode allows for multiple MMC cards to share data and command signals on the RSI interface and allows for the initialization sequence to take place on all cards[1].
RSI_D7 – RSI_D0	These are the configurable bi-directional data channels used for all data transfers both to and from the card. The data bus width can be configured as 1-, 4-, or 8-bit.

Notes:

1. Although earlier revisions of the MMC specifications allowed for multiple MMC cards to be bused to a single RSI interface, it is strongly recommended that only a single device be interfaced to any given RSI interface. Multiple devices sharing a single command and data bus is now actively discouraged by the device specifications.

The table **RSI MMC Signal Descriptions** and describes the functionality of these signals for the MMC protocol and the table **RSI Signal Descriptions (SD and SDIO)** describes the functionality of these signals for SD and SDIO protocols.

Table 24-7: RSI MMC Signal Descriptions

Signal Name	MMC (1- bit)	MMC (4 - bit)	MMC (8 - bit)	Direction
RSI_D7	Not Used	Not Used	Data7	Bi-directional
RSI_D6	Not Used	Not Used	Data6	Bi-directional
RSI_D5	Not Used	Not Used	Data5	Bi-directional
RSI_D4	Not Used	Not Used	Data4	Bi-directional
RSI_D3	Not Used	Data3	Data3	Bi-directional
RSI_D2	Not Used	Data2	Data2	Bi-directional
RSI_D1	Not Used	Data1	Data1	Bi-directional
RSI_D0	Data0	Data0	Data0	Bi-directional
RSI_CMD	Command/ Response	Command/ Response	Command/ Response	Bi-directional
RSI_CLK	Clock	Clock	Clock	Output

Table 24-8: RSI SD and SDIO Signal Descriptions

Signal Name	SD (1- bit)	SD (4 - bit)	SDIO (1 - bit)	SDIO (4 - bit)	Direction
RSI_D7	Not Used	Not Used	Not Used	Not Used	Bi-directional
RSI_D6	Not Used	Not Used	Not Used	Not Used	Bi-directional
RSI_D5	Not Used	Not Used	Not Used	Not Used	Bi-directional
RSI_D4	Not Used	Not Used	Not Used	Not Used	Bi-directional
RSI_D3	Not Used/ Card Detect	Data3/ Card Detect	Not Used/ Card Detect	Data3/ Card Detect	Bi-directional
RSI_D2	Not Used	Data2	Read Wait	Data2/ Read Wait	Bi-directional
RSI_D1	Not Used	Data1	Interrupt	Data1/ Interrupt	Bi-directional
RSI_D0	Data0	Data0	Data0	Data0	Bi-directional
RSI_CMD	Command/ Response	Command/ Response	Command/ Response	Command/ Response/ CCS/CCSD	Bi-directional
RSI_CLK	Clock	Clock	Clock	Clock	Output

Clock Configuration

The RSI is a fast, synchronous peripheral with a programmable clock frequency that is supplied by the RSI_CLK signal. The interface between the RSI and the internal buses operates at *SCLK* frequency. Communication between the clock domain that is supplied externally from the RSI on the RSI_CLK signal and the RSI access to the internal buses is accomplished using synchronizers in the RSI module. The RSI_CLK frequency is configured by the 8-bit RSI_CTL.CLKDIV field and the RSI_CTL.BYPASS bit.

If RSI_CTL.BYPASS is set, the clock frequency driven on the RSI_CLK signal is derived directly from *SCLK*. If RSI_CTL.BYPASS is cleared, the clock divider logic provides an RSI_CLK frequency as shown below, where RSI_CTL.CLKDIV is an 8-bit value ranging between 0 and 255.

RSI_CLK=*SCLK*/2x(RSI_CTL.CLKDIV+1)

The RSI_CLK output is enabled or disabled by the RSI_CTL.CLKEN bit and a power save feature is implemented by setting RSI_CTL.PWRSAVE, which disables the RSI_CLK output when there are no transfers taking place on the RSI interface.

Interface Configuration

The RSI supports multiple card types under various protocols. Different card types may require slightly different interface configurations.

The command signal on MMC cards operates in two different modes depending upon the operating mode of the card. During the card identification mode, the command signal operates in open-drain configuration. When the card enters data transfer mode, the signal is configured to push-pull mode.

NOTE: The internal pull-up resistor of the RSI_CMD signal is only intended to keep the signal from floating. The internal pull-up resistor is not sufficient during the card identification phase when the MMC card RSI_CMD signal is operating in open-drain mode. If support for MMC devices is required, an external pull-up resistor should be added to the RSI_CMD signal as detailed in the JEDEC standard.

The bus width used for the data transfers is configurable to 1-bit, 4-bits, or 8-bits using the RSI_CTL.BUSWID bit field.

To stop signals from floating when no card is inserted or during times when all card drivers are in a high-impedance state, various pull-up and pull-down resistor configurations can be enabled on the RSI_D0 – RSI_D7 signals. The RSI_CFG register provides the following options.

- Enable or disable a pull-up resistor on the RSI_D3 signal.
- Enable or disable pull-up resistors on the RSI_D7 through RSI_D4 and RSI_D2 through RSI_D0 signals.

Card Detection

The RSI allows for software to detect when a card is inserted into its slot. There are a number of ways that this card detection can be performed.

- **Using the data 3 signal.** SD and SDIO cards use an internal pull-up resistor on the RSI_D3 line as a card detect signal to indicate to the host that a card is present. In order to use the RSI_D3 signal for card detection, an external pull down resistor should be added to the pin to pull the pin low during the time a card is not inserted.

  When a card is inserted into the slot, a rising edge is detected on the RSI_D3 signal and the RSI_STAT0.SDCARD bit is set. Once the card has been correctly identified, the RSI_STAT0.SDCARD interrupt should be cleared and disabled. Also disable the pull-up resistor in the SD card by issuing the SET_CLR_CARD_DETECT command. When using the RSI_D3 signal for card detection with an external pull-down resistor, do not enable the internal pull-up resistor (the RSI_CFG.DAT3PUP bit=0).

- **Using the SD/MMC socket and GPIO interrupt.** The recommended method of detecting the insertion of a card is to use the card detect feature that is made available through most sockets. Sockets supporting this feature can have the card detect pin de-bounced and connected to a GPIO pin in order to allow not only interrupt driven card detection but also interrupt driven card removal.

  This is the most reliable and efficient method of detecting the insertion and removal of a card as some MMC devices may not implement the card detect pull-up resistor on the RSI_D3 signal. Once a card is detected, the GPIO pin can have the interrupt level inverted to then generate an interrupt on card removal.

- **Using software polling.** Software polls the slot periodically using the card identification commands for the supported card types. Once a card is inserted, valid responses are sent back to the host. When the card is removed, command and data timeout errors occur.

The following figure shows the circuits required for the card detection schemes discussed above.

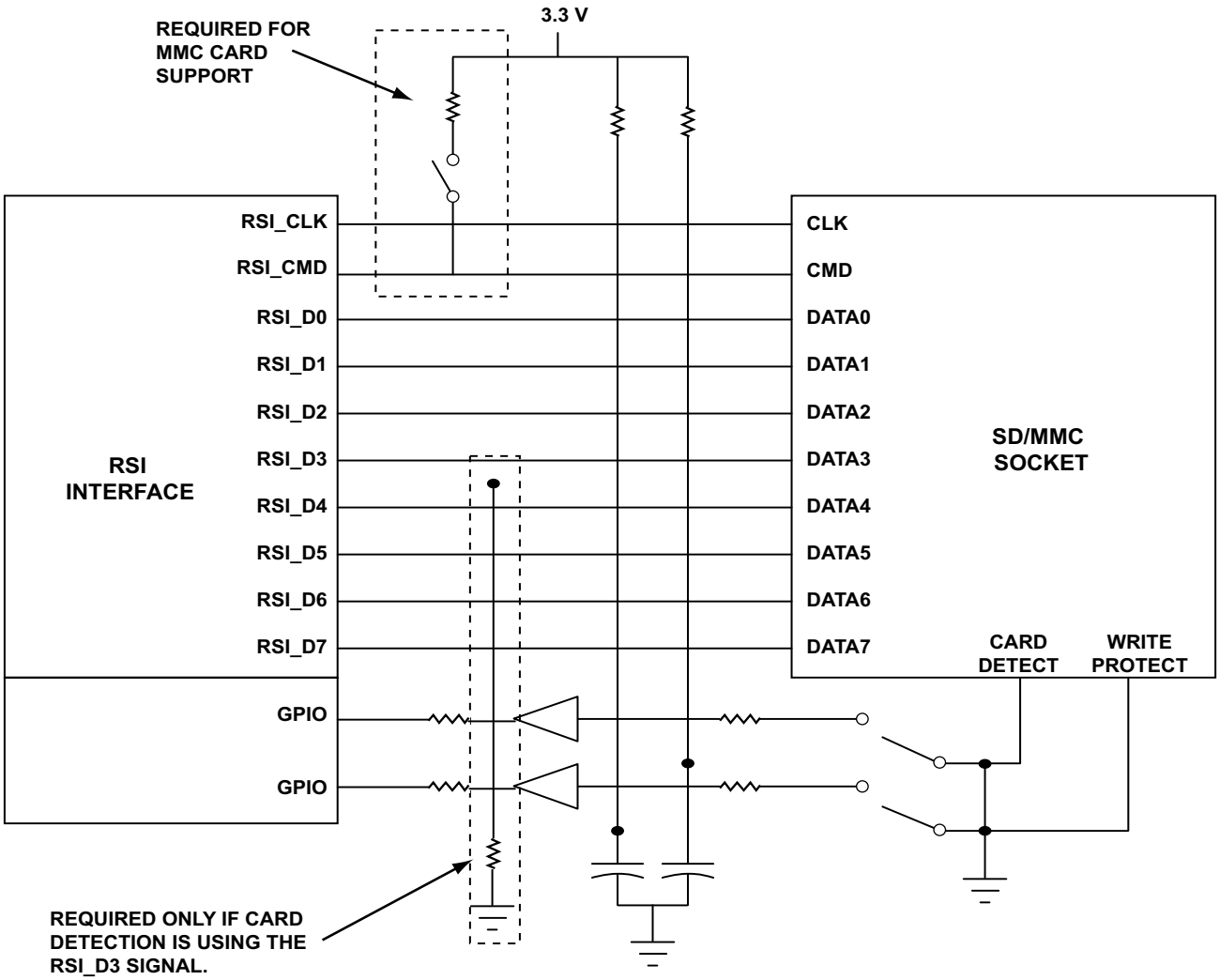


Figure 24-2: RSI Socket Interface

Power Saving Configuration

The RSI requires two internal clock signals that are derived directly from *SCLK*. One of these clock signals is routed to the clock divider and generates the `RSI_CLK` clock.

These clocks must be enabled by setting the `RSI_CFG.CLKSEN` bit for the RSI to function. Clearing this bit disables the RSI regardless of other RSI clock configurations. The `RSI_CLK` signal can be enabled or disabled using the `RSI_CTL.CLKEN` bit.

Additional power saving options can be implemented by setting the `RSI_CTL.PWRSAVE` bit which disables the `RSI_CLK` output when there are no transfers taking place on the RSI interface. These configurations are shown in **RSI Power Saving Configurations**.

Table 24-9: RSI Power Saving Configurations

RSI_CFG.CLKSEN	RSI_CTL.CLKEN	RSI_CTL.PWRSAVE	RSI State	RSI_CLK Output
0	0	0	Disabled	No clock
0	0	1	Disabled	No clock
0	1	0	Disabled	No clock
0	1	1	Disabled	No clock
1	0	0	Enabled	No clock
1	0	1	Enabled	No clock
1	1	0	Enabled	Continuous clock[1]
1	1	1	Enabled	Clock only driven during transfers[1]

1 The PWRON field of the RSI_CFG register must be set to 0x3. If PWRON is cleared, the clock will not output.

RSI Command-Response Interface

The RSI sends commands to and receives responses from the card using the RSI_CMD signal. The command to be sent to the card is issued by writing to the RSI_CMD register. This register contains a 6-bit RSI_CMD.IDX field that contains the command index to be sent to the card. The command index provides support for a total of 64 commands—0 (CMD0) to 63 (CMD63).

Some commands require an argument to be sent along with the command, such as an address for a read transaction. An argument is always sent with the command and it is the responsibility of the card to either ignore or use the argument field based on the command that is received. The argument sent with the command is provided using the RSI_ARG register.

All command transfers are protected by a 7-bit cyclic redundancy check (CRC) code, more commonly referred to as a CRC7 checksum. This allows for transmission errors to be detected and for the command to be re-issued to the card in the event of an error. All commands sent to the card are composed of 48-bits as shown in the table below.

Table 24-10: RSI Command Format

Bit Position	Width	Value	Description
47	1	0	Start bit
46	1	1	Transmitter bit
[45:40]	6	—	Command index
[39:8]	32	—	Argument
[7:1]	7	—	CRC7 checksum

Table 24-10: RSI Command Format (Continued)

Bit Position	Width	Value	Description
0	1	1	End bit

The RSI_CMD register also provides configuration information about whether a response is to be expected back from the card and the type of response.

The RSI can be configured using the RSI_CMD.RSP and RSI_CMD.LRSP bit fields to expect no response, a short response or a long response type as shown in the following tables.

Table 24-11: RSI Short Response Format

Bit Position	Width	Value	Description
47	1	0	Start bit
46	1	0	Transmitter bit
[45:40]	6	—	Command index or check bits[1]
[39:8]	32	—	Card status, register contents or argument field
[7:1]	7	—	CRC7 checksum or check bits[2]
0	1	1	End bit

1.Responses that do not contain the command index have b#111111 in the check bits field.
2.Responses that do not contain a CRC7 check sum have b#111111 in the check bits field.

Table 24-12: RSI Long Response Format

Bit Position	Width	Value	Description
135	1	0	Start bit
134	1	0	Transmitter bit
133:128	6	111111	Check bits[1]
127:1	127	—	Register contents including internal CRC7[2]
0	1	1	End bit

1.Responses that do not contain the command index have b#111111 in t he check bits field.
2.Responses that do not contain a CRC7 check sum have b#111111 in the check bits field.

Like the commands, all responses are sent on the RSI_CMD signal. A response always has a 0 start bit followed by a 0 transmission bit to indicate the transfer is from card to the RSI. Unlike the commands issued by the RSI, not all responses are protected by a CRC7 checksum. Refer to the appropriate specification for full details on the response formats for a specific device and whether they are protected by a CRC7 checksum.

When a short response is received, the response is broken down by the RSI. The 32-bit field containing bits [39:8]is stored to RSI_RESP0, where bit 39 of the response corresponds to bit 31 of RSI_RESP0 and bit 8 of the response to bit 0 of RSI_RESP0. Bits [45:40] of the response are stored to the RSI_RESP_CMD register.

For a long response, bits [127:1] of the response are stored in the RSI_RESP0, RSI_RESP1, RSI_RESP2 and RSI_RESP3 registers. Bit 31 of RSI_RESP0 contains the most significant bit (bit 127) of the response and bit 0 of RSI_RESP3 contains bit 1 of the response. Bit 31 of RSI_RESP3 is always zero.

The following figure shows the command path state machine. For the state machine to be active, the RSI must be enabled through the RSI_CTL.CLKEN bit. Disabling the clocks to the RSI results in the state machine returning to the IDLE state.

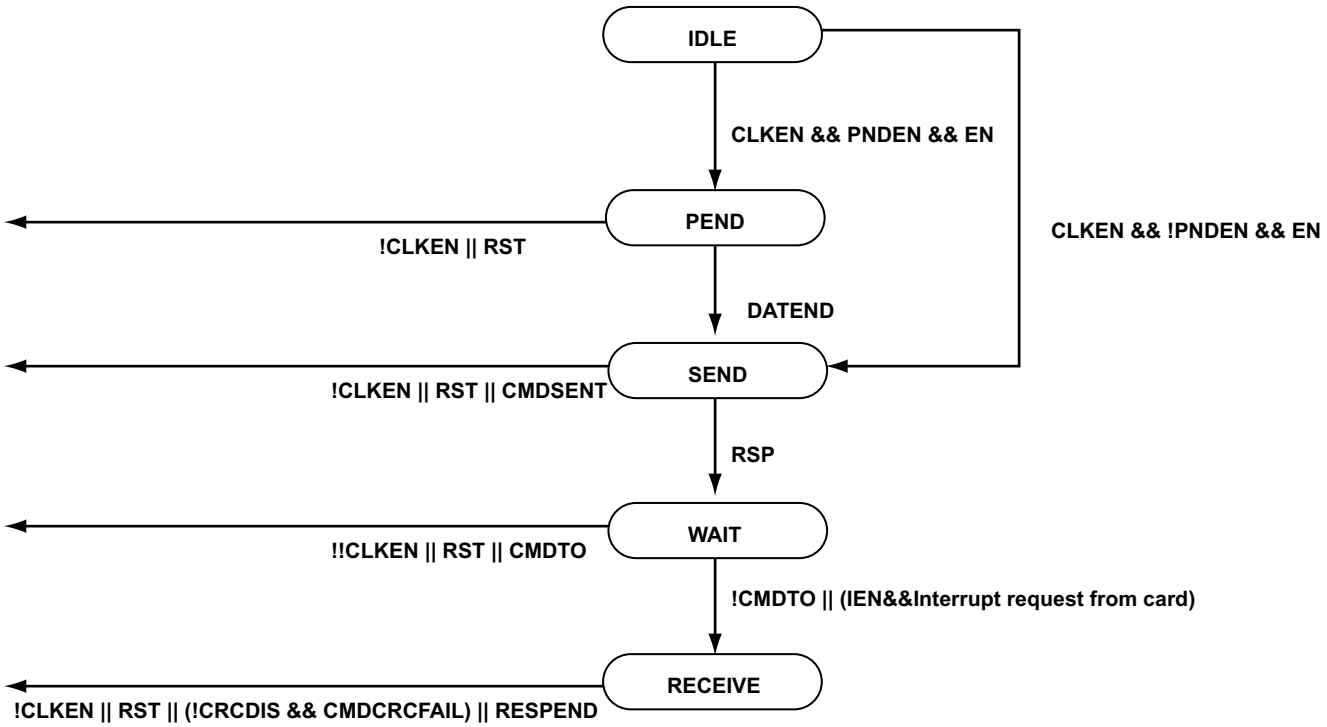


Figure 24-3: RSI Command Path State Machine

The command path state machine is responsible for setting and clearing a number of status flags in the RSI_XFRSTAT register. The following table lists the status flags and exception flags that are affected by the command path state machine.

Table 24-13: RSI Command Path Status Flags

RSI_XFRSTAT/RSI_STAT0 Flag	Description	State Flag Set in
RSI_XFRSTAT.CMDACT	Command transfer is in progress	WAIT_S
RSI_XFRSTAT.CMDSENT	Command without response sent successfully	SEND

Table 24-13: RSI Command Path Status Flags (Continued)

RSI_XFRSTAT/RSI_STAT0 Flag	Description	State Flag Set in
RSI_XFRSTAT.CMDTO	Response timeout occurred (64 RSI_CLK cycles)	WAIT_S
RSI_XFRSTAT.CMDCRCFAIL	Response CRC failure	RECEIVE
RSI_XFRSTAT.RESPEND	Response received successfully (and CRC check passed if the CRCDIS bit is cleared)	RECEIVE

The command path operates in a half-duplex mode, so that commands and responses can either be sent or received. If the state machine is not in the SEND state, the RSI_CMD output is in high impedance state.

The following figure describes a typical command and response transfer, the RSI_CMD signal is sampled by the card and the host on the rising edge of RSI_CLK.

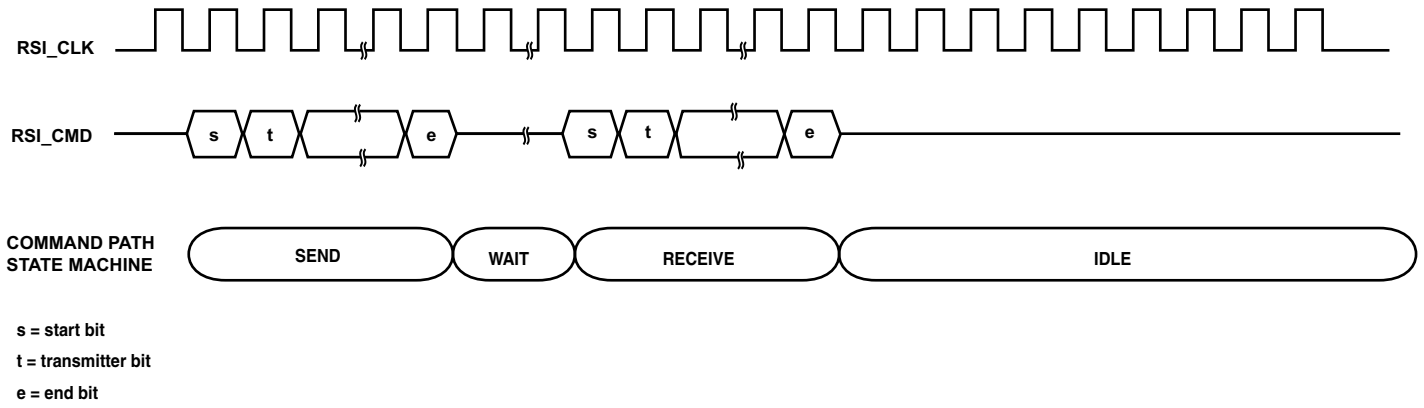


Figure 24-4: RSI Command Transfer

The following sections describes the RSI command path states.

IDLE State

The command path state machine remains in the IDLE state when it is not active and leaves the IDLE state when the RSI_CMD.EN bit is set. The state transitions to the PEND state if the RSI_CMD.PNDEN bit set, otherwise it enters the SEND state.

The state machine remains in the IDLE state for at least eight RSI_CLK cycles when the command path state machine returns to the IDLE state from another state, and the return is not because the RSI was disabled or reset. During this time, the RSI continues to drive the RSI_CLK signal even if the RSI_CTL.PWRSAVE bit is set. This allows the card to complete the current operation. If enabled again, the state machine leaves the IDLE state only after the eight RSI_CLK cycles have passed.

PEND State

The RSI enters the PEND state if the RSI_CMD.PNDEN bit is set. The state machine remains in the PEND state until it is notified by the data path sub block that a data transfer has completed. This is indicated by the RSI_XFRSTAT.DATEND flag being set when RSI_DATA_CNT decrements to zero. This mode allows for the automatic transmission of the STOP_TRANSMISSION command after reading or writing the required amount of data for stream-based transactions.

The RSI_CMD.PNDEN feature is not available for block-based transfers and cannot be used to automatically issue the STOP_TRANSMISSION command for MULTIPLE_BLOCK_READ or MULTIPLE_BLOCK_WRITE operations.

SEND State

During the SEND state, the RSI sets the RSI_XFRSTAT.CMDACT flag to indicate a transfer is in progress. The behavior of the state machine after the command is sent depends upon whether the command expects a response back from the card.

If no response is expected, the RSI clears the RSI_XFRSTAT.CMDACT flag and sets the RSI_XFRSTAT.CMDSENT flag to indicate that a command operation without a response has been completed. The state then goes to IDLE.

If a response is expected, the RSI enters the WAIT state.

WAIT State

In the WAIT state, the RSI waits to receive a response on the RSI_CMD signal. Upon entering this state, an internal timer starts. If the response is not received within 64 RSI_CLK cycles, the RSI_XFRSTAT.CMDTO flag is set and the RSI_XFRSTAT.CMDACT flag is cleared. The state machine then enters the IDLE state, awaiting the next action.

A response, sent back from the card and indicated by the 0 start bit on the RSI_CMD signal, transitions the RSI to the RECEIVE state to receive a 48-bit or 136-bit response.

The WAIT state can also detect card interrupts. This is an optional feature that applies only to MMC cards. This feature is enabled by setting the RSI_CMD.IEN bit. When this bit is set, the timeout timer that is normally started upon entry to the WAIT state is disabled. The RSI remains in this state until a card interrupt is detected.

Cards that implement this feature may have functions with a delayed response that is triggered by an internal event in the card. Once the event is triggered the card sends the response. The RSI then detects this start bit of the response and proceeds to the RECEIVE state.

RECEIVE State

In the RECEIVE state the RSI reads the response on the RSI_CMD signal from the card. If the response (short or long) passes the CRC check, the RSI_XFRSTAT.CMDACT flag is cleared and the RSI_XFRSTAT.

RESPEND flag is set. If the CRC check fails, the RSI_XFRSTAT.CMDCRCFAIL flag is set. In either case, the state machine then transitions to the IDLE state.

Command Path CRC

The command CRC generator calculates the 7-bit CRC check-sum for all 40 bits preceding the CRC code for both 48-bit commands and 48-bit responses. This includes the start bit, transmitter bit, command index, and command argument (or card status).

The 7-bit CRC checksum is calculated for the first 120 bits of the register contents field for the long response format. Note that the start bit, transmitter bit, and the six check bits are not used in the CRC calculation for the long response. The command and response CRC checksum is a 7-bit value that is calculated as follows.

CRC[6:0] = Remainder $(x_7 \times M(x))/G(x)$

With: $G(x) = x_7 + x_3 + 1$

and for a short response:

$M(x) = x_{39} \times (\text{start bit}) + ... + x_0 \times (\text{last bit before CRC})$

or for a short response:

$M(x) = x_{19} \times (\text{start bit}) + ... + x_0 \times (\text{last bit before CRC})$

RSI Data Interface

Data transfers both to and from the RSI take place over the RSI data bus signals RSI_D0 –RSI_D7. The RSI data bus width is configured by the RSI_CTL.BUSWID bit field. The default is 1-bit bus mode, where the data is transferred over the RSI_D0 signal. 4-bit mode or 8-bit mode can be enabled after configuring the card for 4-bit or 8-bit mode of operation, respectively.

The RSI data path state machine operates at RSI_CLK frequency. The state machine leaves the IDLE state when the RSI_DATA_CTL.DATEN bit is set, enabling the data transfer. The state entered upon leaving the IDLE state is determined by the RSI_DATA_CTL.DATDIR bit. The data path state machine is shown in the following figure.

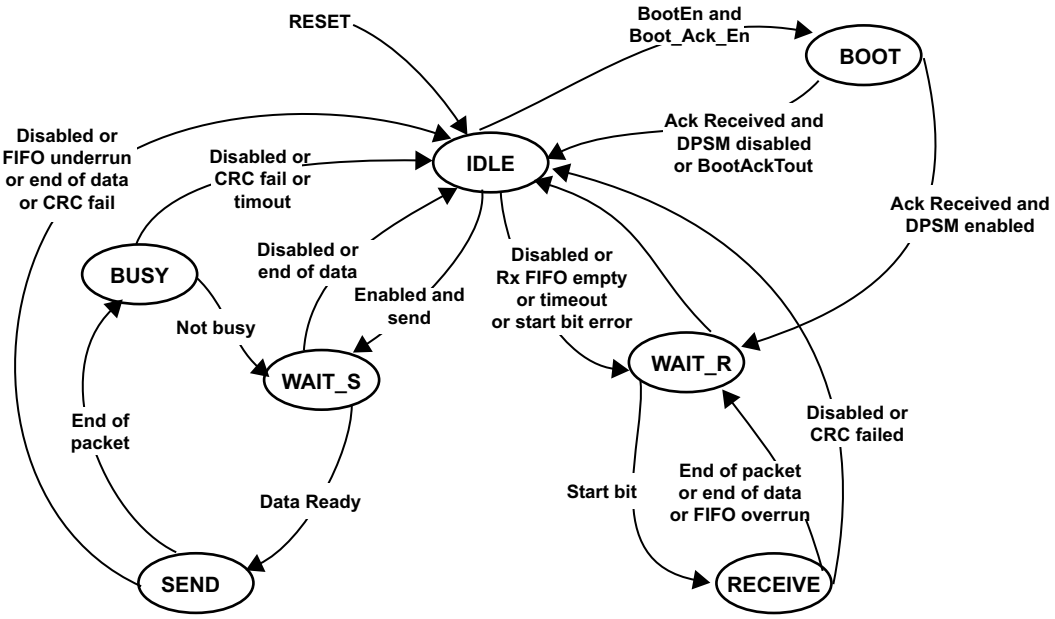


Figure 24-5: RSI Data Path State Machine

The data path status flags are shown in the following table.

Table 24-14: RSI_STATUS Flags

RSI_XFRSTAT Flag	Description	Status Flag Set In
TXACT	Data transmit in progress	WAIT_S
RSI_XFRSTAT.RXACT	Data receive in progress	WAIT_R
RSI_XFRSTAT.DATBLKEND	Data Block sent successfully and CRC pass token received Data Bock received correctly and CRC passed	BUSY (block transfer mode only) RECEIVE (block transfer mode only)
RSI_XFRSTAT.DATCRCFAIL	Data block CRC failed on transmit Data block CRC failed on receive	SEND is transmitted data is not a multiple of DATA_BLK_ LGTH BUSY if CRC token indicates failure
RSI_XFRSTAT.DATTO	Transmit timeout occurred before card de-asserted busy signal on RSI_DATA0 Receive timeout error occurred before start bit of data detected	BUSY WAIT_R
RSI_XFRSTAT.DATEND	All data sent All data received	SEND RECEIVE
RSI_XFRSTAT.SBITERR	Start bit not detected on all RSI_DATAx signals	WAIT_R

Table 24-14: RSI_STATUS Flags (Continued)

RSI_XFRSTAT Flag	Description	Status Flag Set In
RSI_XFRSTAT.TXFIFOSTAT	Transmit FIFO is half empty	SEND
RSI_XFRSTAT.TXFIFOFULL	Transmit FIFO is full	SEND
RSI_XFRSTAT.TXFIFOZERO	Transmit FIFO is empty	SEND
RSI_XFRSTAT.TXUNDR	Transmit FIFO under run error	SEND
RSI_XFRSTAT.TXFIFORDY	Valid data available in the transmit FIFO	SEND
RSI_XFRSTAT.RXFIFOSTAT	Receive FIFO is half empty	RECEIVE
RSI_XFRSTAT.RXFIFOFULL	Receive FIRO is full	RECEIVE
RSI_XFRSTAT.RXFIFOZERO	Receive FIFO is empty	RECEIVE
RSI_XFRSTAT.RXOVER	Receive FIFO over run error	RECEIVE
RSI_XFRSTAT.RXFIFORDY	Valid data is available in the receive FIFO	RECEIVE

RSI Data Transmit Path

The transmit path consists of the WAIT_S, SEND, and *BUSY* states. The RSI_BLKSZ, RSI_DATA_LEN and RSI_DATA_TMR registers must be configured before enabling the data path state machine using the RSI_DATA_CTL register. Upon leaving the IDLE state and entering the WAIT_S state, the RSI sets the RSI_XFRSTAT.TXACT flag and copies the RSI_DATA_LEN register contents into the RSI_DATA_CNT register.

The behavior of the SEND state depends on which transfer mode is configured.

- **Stream transfer mode** - If the RSI is configured for stream transfer mode, it sends data to the card until RSI_DATA_CNT expires, at which point the RSI_XFRSTAT.DATEND flag is set and the state machine returns to the IDLE state.

  Additionally, the transition of RSI_DATA_CNT to zero activates the command path state machine if it is currently in the PEND state. If at any point during the stream transfer the transmit FIFO becomes empty and data is not available in the FIFO by the time the next transfer is due to take place, the RSI_XFRSTAT.TXUNDR flag is set before returning to the IDLE state.

- **Block transfer mode** - The RSI_BLKSZ bytes, as specified during the write to RSI_DATA_CTL, are transmitted. Each byte transferred also decrements RSI_DATA_CNT.

  On completion of the block transfer, the RSI appends an internally generated 16-bit CRC code and an end bit to the data transferred over the RSI_D0 – RSI_D7 signals.

  The RSI then waits for the card token response on the RSI_D0 line to indicate whether the data was received correctly by the card or not. If the CRC response token sent by the card indicates the data was received correctly, the RSI_XFRSTAT.DATBLKEND flag is set before moving to the BUSY state. If the data was not received correctly, the RSI_XFRSTAT.DATCRCFAIL flag is set before returning to the IDLE state.

  When RSI_DATA_CNT decrements to zero, the RSI_XFRSTAT.DATEND flag is set. If the total number of bytes transmitted for the current block results in the RSI_DATA_CNT decrementing to zero and the

number of bytes transferred is not equal to RSI_BLKSZ, the transmission stops and the RSI_XFRSTAT.DATCRCFAIL flag is set.

The data path returns to the IDLE state. If at any point during the block transfer the transmit FIFO becomes empty and data is not available in the FIFO by the time the next transfer is due to take place, the RSI_XFRSTAT.TXUNDR flag is set before returning to the IDLE state.

During the BUSY state, the RSI continuously samples RSI_D0 which at this point is driven low by the card to indicate that the card is busy. When a logic high state is detected, indicating that the card is no longer busy, the state machine returns to the WAIT_S state. It then either returns to IDLE if all data has been sent or moves back to the SEND state to start another block transfer.

On entry to the BUSY state, the RSI started decrementing the timeout value specified in the RSI_DATA_TMR register. If the RSI timeout counter expires before the RSI_D0 signal is detected high, the RSI sets the RSI_XFRSTAT.DATTO flag and returns to the IDLE state.

RSI Data Receive Path

The receive path consists of the WAIT_R and the RECEIVE states. RSI_BLKSZ, RSI_DATA_LEN and RSI_DATA_TMR must be configured, before enabling the data path state machine withRSI_DATA_CTL. Upon leaving the IDLE state and entering the WAIT_R state, the RSI sets the RSI_XFRSTAT.RXACT flag and copies RSI_DATA_LEN into RSI_DATA_CNT. The behavior of the RECEIVE state is influenced by the transfer mode.

Once the receive path has entered the WAIT_R state after being enabled for a receive transaction, the RSI starts decrementing the timeout value supplied by the RSI_DATA_TMR.

If the RSI is configured for a 1-bit data bus, the RSI_XFRSTAT.DATTO flag is set if a start bit is not detected on the RSI_D0 signal before the timeout counter reaches zero. The state machine then returns to the IDLE state.

If the RSI is configured for 4-bit bus mode and the start bit is not detected on all four RSI_D0 – RSI_D3 signals before the timeout counter expires—the RSI_XFRSTAT.DATTO flag is set. The state machine returns to the IDLE state. If a start bit is detected on some, but not all, of the RSI_D0 – RSI_D7 signals on the same sampled clock cycle, then the RSI_XFRSTAT.SBITERR flag is set and the state machine returns to the IDLE state. Upon correct detection of the start bit, the state machine moves into the RECEIVE state.

The behavior of the RECEIVE state differs for stream and block transfers.

- **Stream transfer mode** - For stream transfers, received data is packed into bytes and written to the data FIFO. Data is continuously received and written to the data FIFO until RSI_DATA_CNT decrements to zero.

  When the counter reaches zero, the remaining data in the shift register is written into the FIFO, the RSI_XFRSTAT.DATEND flag is set and the state machine transitions to the WAIT_R state.

  When the receive FIFO is detected empty, the RSI_XFRSTAT.RXFIFOZERO flag is set and the state goes to IDLE. If the data FIFO becomes full and data has not been read from the FIFO prior to the next byte

being written to the FIFO, then the RSI_XFRSTAT.RXOVER flag is set. The state then transitions to WAIT_R then to IDLE.

- **Block transfer mode** - In block transfer mode, the received data is packed into bytes and written to the data FIFO.

  When RSI_BLKSZ bytes have been received, the RSI reads the 16-bit CRC check bits. If the received CRC matches the internally calculated CRC, the RSI_XFRSTAT.DATBLKEND flag is set and the state transitions to WAIT_R.

  If the RSI_DATA_CNT counter expires in alignment with the end of a RSI_BLKSZ block, the RSI_XFRSTAT.DATEND and RSI_XFRSTAT.DATBLKEND flags are set, and the state transitions to WAIT_R.

  When the receive FIFO is detected empty, the RSI_XFRSTAT.RXFIFOZERO flag is set and the state transitions to IDLE. If RSI_DATA_CNT expires before the end of a RSI_BLKSZ block, the RSI_XFRSTAT.DATCRCFAIL flag is set. The state transitions to IDLE.

Data Path CRC

The data CRC generator calculates the 16-bit CRC checksum for all bits sent or received for a given block transaction (stream based data transfers are not available). For a 1-bit bus configuration, the 16-bit CRC is calculated for all data sent on the RSI_D0 signal. For a 4-bit-wide data bus, the 16-bit CRC is calculated separately for each RSI_D0 – RSI_D7 signal. The data path CRC checksum is a 16-bit value calculated as follows.

$CRC[15:0] = \text{Remainder } (x_{16} \times M(x))/G(x)$

with:

$G(x) = x_{16} + x_{12} + x_5 + 1$

where:

$M(x) = x_{((8 \times \text{DTX\_BLK\_LGTH}) - 1)} \times (\text{first data bit}) + ... + x_0 \times (\text{last data bit})$

RSI Data FIFO

The data FIFO is a 32-bit wide, 16-word deep data buffer with transmit and receive logic. The FIFO is configuration depends on the state of the RSI_XFRSTAT.TXACT and RSI_XFRSTAT.RXACT flags. If the RSI_XFRSTAT.TXACT is set, the FIFO operates as a transmit FIFO supplying data to the RSI for transfer to the card. If the RSI_XFRSTAT.RXACT flag is set, the FIFO operates as a receive FIFO, where the RSI writes data received from the card. If neither flags are set, then the FIFO is disabled.

When the transmit FIFO is disabled, all the transmit status flags are de-asserted and the transmit read and write pointers are reset. The RSI asserts the RSI_XFRSTAT.TXACT flag upon starting a data transfer. During the data transfer the transmit logic maintains the transmit FIFO status flags shown in the following table.

Table 24-15: RSI Transmit FIFO Status Flags

RSI_XFRSTAT Flag	Description
RSI_XFRSTAT.TXFIFOSTAT	Transmit FIFO is half empty
RSI_XFRSTAT.TXFIFOFULL	Transmit FIFO is full
RSI_XFRSTAT.TXFIFOZERO	Transmit FIFO is empty
RSI_XFRSTAT.TXUNDR	Transmit FIFO under run error
RSI_XFRSTAT.TXFIFORDY	Valid data available in the transmit FIFO

When the receive FIFO is disabled, all receive status flags are de-asserted and the receive read and write pointers are reset. The RSI asserts the RXACT flag upon starting a data read transaction. During the data transfer, the receive logic maintains the receive FIFO status flags shown in the following table.

Table 24-16: RSI Receive FIFO Status Flags

RSI_XFRSTAT Flag	Description
RSI_XFRSTAT.RXFIFOSTAT	Receive FIFO is half empty
RSI_XFRSTAT.RXFIFOFULL	Receive FIFO is full
RSI_XFRSTAT.RXFIFOZERO	Receive FIFO is empty
RSI_XFRSTAT.RXOVER	Receive FIFO over run error
RSI_XFRSTAT.RXFIFORDY	Valid data available in the receive FIFO

Card Busy/Ready Detection

Some commands, like CMD6, may assert the busy signal by driving the RSI_D0 signal low two cycles after the end bit of the command. Setting the RSI_CMD.CHKBUSY bit configures the RSI to check if the line is busy. Note that the RSI_D0 – RSI_D7 lines are driven by the card though their values are not relevant.

If the RSI_CMD.CHKBUSY bit is enabled for a particular command, on the third cycle from the command's end bit, the RSI_D0 signal is checked to see if it is low. If RSI_D0 is low, the RSI_STAT0.BUSYMODE bit is set. When the RSI_D0 line goes high, the RSI_STAT0.CARDRDY bit (W1C) is set, the RSI_STAT0.BUSYMODE bit is cleared, and the card ready interrupt is generated.

If the RSI_D0 doesn't go low at the third cycle from the command's end bit, the card is considered not busy. The RSI_STAT0.BUSYMODE bit of RSI exception status register isn't set, the card ready interrupt is generated and the RSI_STAT0.CARDRDY bit is set. If the RSI_CMD.CHKBUSY bit is not enabled, the RSI_D0 signal is not checked for the busy condition.

The RSI_STAT0.BUSYMODE bit is also updated during a data write operation if the RSI_D0 signal is pulled low while the card is programming the data. The RSI_STAT0.CARDRDY bit and the card ready interrupt are not updated in this case. The card busy timing is shown in the following figure.

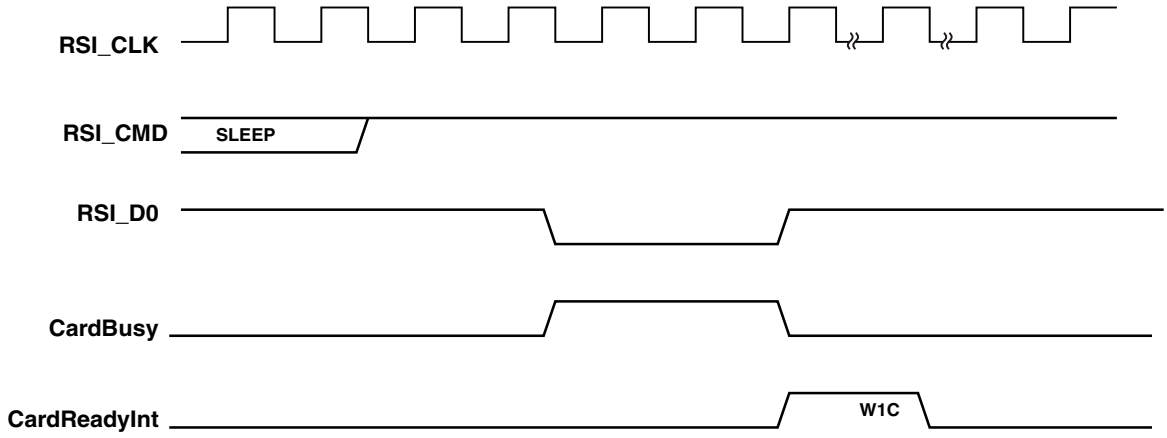


Figure 24-6: Card Busy Operation

SDIO Support

Two additional RSI features implement SDIO functionality.

- Hardware interrupt support over the RSI_D0 pin
- Read wait request over the RSI_D2 pin

SDIO devices may have multiple interrupt sources that are mapped to a single interrupt line. The interrupt is level-sensitive, allowing multiple functions to generate an interrupt simultaneously. Thus the interrupt request will continually be asserted until all sources generating an interrupt are determined and cleared by the RSI.

The sources of the interrupts are found by interrogating the SDIO device. The interrupts are cleared through operations unique to each function.

The SDIO device sends an interrupt request to the RSI by asserting the RSI_D1 signal low. The interrupt status is indicated by the RSI_STAT0.SDIOINT bit. The status can be configured to interrupt the processor through the RSI_IMSK0.SDIOINT bit.

When the RSI is configured for 1-bit bus width, the interrupt is generated by the SDIO with no timing constraints because the RSI_D1 signal acts as a dedicated IRQ signal. The RSI should be configured using the RSI_CFG.DATPUP bits such that pull-up are enabled on all RSI_D0 – RSI_D7 signals. When the RSI sampling RSI_D1 low, the RSI asserts the RSI_STAT0.SDIOINT flag. This flag is asserted until the RSI_D1 signal is sampled high again.

When the RSI is configured for 4-bit bus widths, the RSI_D1 signal is shared between the IRQ signal and the RSI_D1 signal. In this configuration, the interrupt is only be recognized by the RSI within a specific interrupt period.

RSI Operating Modes

The following sections describe the functionality of the RSI module in various modes of operation.

Card Identification Mode

When a device connected to the RSI is first powered and detected by the host or has been reset, the device must first be identified and initialized by the host. The software determines whether the device is compatible with the RSI controller and the implemented software drivers. This phase in the procedure is known commonly as the card identification mode.

When a device is in card identification mode, the host may be required to perform the following actions.

- Reset the device
- Validate the device operating voltage range
- Identify the device type
- Assign/request a relative card address (RCA)

All communications between the host and card during the card identification phase occur using the `RSI_CMD` signal. The maximum clock frequency during the identification phase is typically far lower than the maximum data transfer frequency for the card.

Data Transfer Mode

The card enters stand-by state, known as the data transfer mode, when it has been assigned an RCA. Data transfers can only take place when the device has entered the data transfer mode.

Once the device is in data transfer mode, communication takes place through the `RSI_CMD` and the `RSI_D0` – `RSI_D7` signals. The card is further interrogated to identify bus widths, maximum clock frequency, and the device capacity. At this point the bus width can be altered and the clock frequency can be increased.

Data may be written to the device or read from the device using the following two methods.

- Stream reads and writes. Stream transfers produce a continuous stream of data until the RSI stops the transfer by setting a specific command. For stream read and write operations, additional maximum operating frequency limitations may be imposed by the device. Stream write operations may also have restrictions that are dependent upon writable block boundaries.
- Block reads and writes. Block based transfers result in a block of a pre-configured size being transferred. The block size depends on the device and is obtained by reading registers contained on the device during the device detection procedure.

The data transfer between the RSI block and the processor's internal memory can be performed in two ways as described below.

DMA Data Transfers

The RSI block has a dedicated DMA channel assigned. This DMA channel can be configured either to transmit/receive the data from/to the RSI data FIFO without core intervention.

Core Data Transfers

If the RSI DMA channel is disabled, the RSI transmit/receive FIFO may be written or read by the processor core as a memory-mapped register (RSI_FIFO). In order to avoid FIFO overflow or underflow, the core should access the FIFO registers in one of the two following ways.

1. Unmask the transmit FIFO empty (RSI_XFR_IMSK0.TXFIFOZERO, RSI_XFR_IMSK1.TXFIFOZERO) or receive FIFO full (RSI_XFR_IMSK0.RXFIFOFULL, RSI_XFR_IMSK1.RXFIFOFULL) interrupts. Write the word to be sent to the transmit FIFO or read the received word from the receive FIFO inside the interrupt service routine.
2. Poll the transmit FIFO empty (RSI_XFRSTAT.TXFIFOZERO) or receive FIFO full (RSI_XFRSTAT.RXFIFOFULL) status bits. Write the word to be sent to the transmit FIFO or read the received word from the receive FIFO after the corresponding status bit is detected as set.

Boot Mode

MultiMedia Cards (MMC) based on the MMCA specifications version 4.3 or later support a special mode known as boot mode. In this mode, the MMC host can read boot data from the slave MMC device by keeping RSI_CMD line low after power-on, or sending CMD0 with argument 0xFFFFFFFA (optional for slave), before issuing CMD1. The data can be read from either a dedicated boot area or user area.

Normal Boot Mode

Before enabling normal boot mode, the boot timing counter bits (RSI_BOOT_TCNTR.SETUP and RSI_BOOT_TCNTR.HOLD) and boot mode type bit (RSI_CFG.MMCBMODE=0) should be configured. Otherwise the RSI uses the default values of 74 and 56 RSI_CLK cycles.

Program the RSI_DATA_TMR register to act as boot data timeout register. If the boot acknowledge bit (RSI_CFG.BACKEN) is set, the boot acknowledge timeout register (RSI_BACK_TOUT) should also be configured.

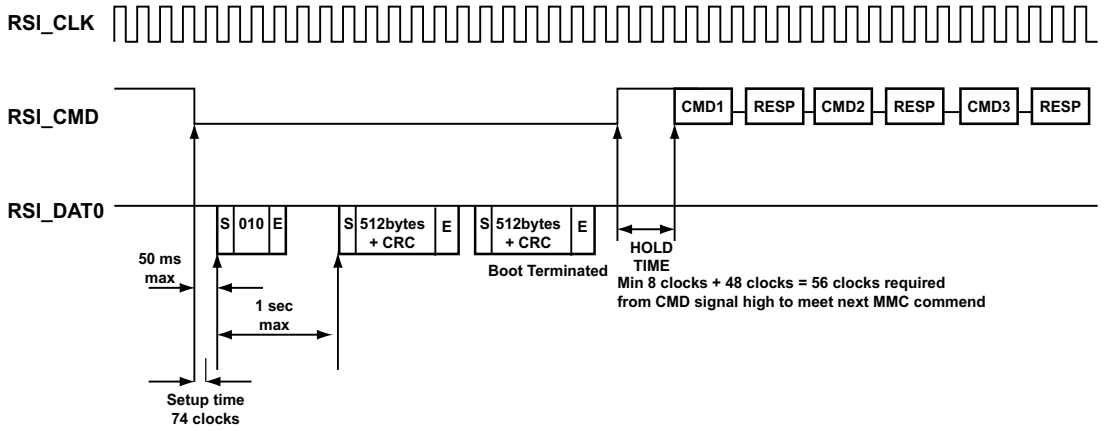


Figure 24-7: Normal Boot Mode Timing

When the RSI block is enabled for normal boot mode by setting the RSI_CFG.MMCBEN bit, the following tasks are performed in the same cycle.

1. The RSI_CMD line is pulled low.
2. The Boot setup counter register (RSI_BOOT_TCNTR) starts decrementing.
3. The Boot acknowledge timeout counter register (RSI_BACK_TOUT) starts incrementing if the RSI_CFG.BACKEN bit is set.
4. The boot data timeout counter register (RSI_DATA_TMR) starts incrementing.

When the boot setup counter expires, the boot setup time expire interrupt is generated, setting the RSI_STAT0.BSETUPEXP bit. This interrupt indicates that card has entered the boot state. Clearing this bit also clears this interrupt line.

NOTE: All bits in the RSI_STAT0 register are W1C (write-1-to-clear) bits.

If the boot acknowledge bit (RSI_CFG.BACKEN) is set, the RSI goes to the boot state and expects an acknowledgment to be returned by the slave within the interval programmed in the boot acknowledge counter register (RSI_BACK_TOUT). If the acknowledge is not received before the counter reaches this value, the boot acknowledge timeout interrupt is generated and the boot acknowledge timeout bit (RSI_STAT0.BACKTO) is set.

If the acknowledge is received, but its value is not 010, the boot acknowledge timeout interrupt (RSI_STAT0.BACKTO) is generated and the boot ack received is corrupted bit (RSI_STAT0.BACKBAD) is set.

In either case the RSI enters the IDLE state. The core should terminate the boot operation by pulling the RSI_CMD signal high by writing 0 to the MMC boot enable bit (RSI_CFG.MMCBEN). Now the boot hold counter register (RSI_BOOT_TCNTR) starts decrementing. When the counter expires, the boot hold time expired interrupt is generated and the RSI_STAT0.BHOLDEXP bit is set. The RSI can now start normal non boot operation (even though the boot operation failed).

When the data transfer is enabled by writing to the RSI_DATA_CTL register, if the acknowledge is received before the RSI_BACK_TOUT counter expires, the RSI enters the WAIT_R state and waits for the start bit of the data.

When the data transfer is enabled by writing to the RSI_DATA_CTL register, if the RSI_CFG.BACKEN bit is cleared, then no acknowledgment is expected and the RSI enters the WAIT_R state.

The slave should start sending the boot data before boot data timeout counter register (RSI_DATA_TMR) expires. If the counter expires while state machine is still in the WAIT_R state, the boot data timeout interrupt is generated and its corresponding RSI_STAT0.BDATTO bit is set. After a timeout the RSI enters the IDLE state.

When the entire boot data is received, the core can write 0 to the RSI_CFG.MMCBEN bit to the terminate the boot operation. The boot hold counter register (RSI_BOOT_TCNTR) now starts decrementing. When the counter expires, the boot hold time expired interrupt is generated and the RSI_STAT0.BHOLDEXP bit is set. The RSI can now start normal non boot operation (even though the boot operation failed).

The slave is now ready for CMD1 operation. The master must start a normal MMC initialization sequence by sending CMD1.

Alternate Boot Mode

Before enabling the alternate boot mode, program the boot timing counter bits (RSI_BOOT_TCNTR.SETUP and RSI_BOOT_TCNTR.HOLD) and boot mode type bit (RSI_CFG.MMCBMODE=1). Otherwise the RSI uses the default values of 74 and 56 RSI_CLK cycles. Program the RSI_DATA_TMR register to act as the boot data timeout register. If the RSI_CFG.BACKEN bit is set, program the boot acknowledge timeout (RSI_BACK_TOUT) register.

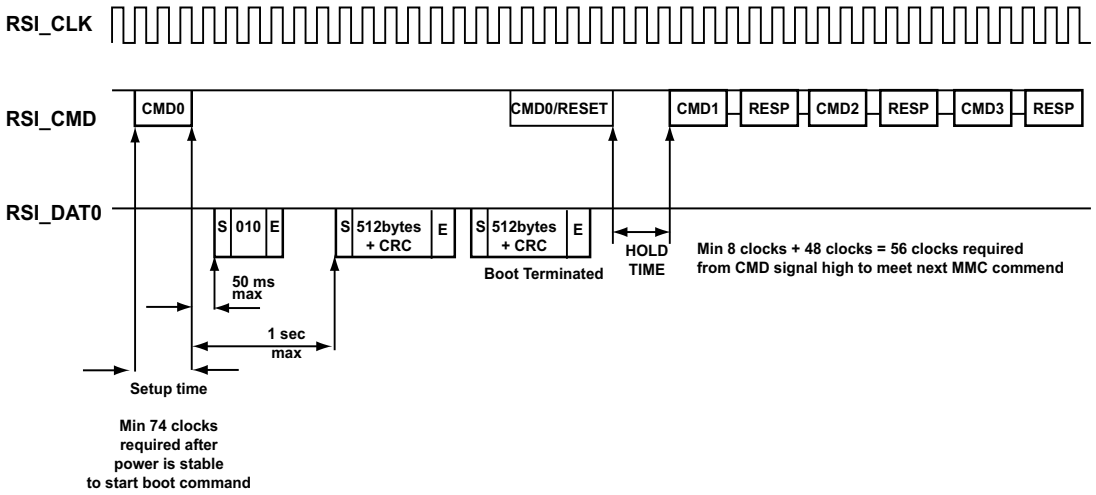


Figure 24-8: Alternate Boot Mode Timing

NOTE: All bits in the RSI_STAT0 register are W1C (write-1-to-clear) bits.

When the RSI block is enabled for alternate boot mode by setting the MMC boot enable bit (RSI_CFG.MMCBEN=0), the boot setup counter (RSI_BOOT_TCNTR) starts decrementing. When this counter expires, the boot setup time expired interrupt is generated and the RSI_STAT0.BSETUPEXP bit is set.

After receiving the boot setup time expired interrupt, send CMD0 with the argument 0xFFFFFFFA to the slave to start the boot operation. When the end bit of CMD0 is reached on the CMD line, the boot acknowledge timeout counter register (RSI_BACK_TOUT) and the boot data timeout counter register (RSI_DATA_TMR) start incrementing from zero.

If the boot acknowledge bit (RSI_CFG.BACKEN) is set, the RSI enters the boot state and expects an acknowledgment to be returned by the slave within the interval programmed in the boot acknowledge counter register (RSI_BACK_TOUT). If the acknowledge is not received before the counter expires, the boot acknowledge interrupt is generated and the boot acknowledge timeout bit (RSI_STAT0.BACKTO) is set. If the acknowledge is received, but its value is not 010, the boot acknowledge received is corrupted interrupt is generated and the boot ack received is corrupted bit (RSI_STAT0.BACKBAD) is set. In both cases the RSI goes back to the IDLE state.

The core can terminate the boot operation by first clearing the RSI_CFG.MMCBEN bit and then sending the CMD0 command. At this point the boot hold counter register (RSI_BOOT_TCNTR) starts decrementing. When the counter expires, the boot hold time expired interrupt is generated and the RSI_STAT0.BHOLDEXP bit is set. The RSI can now start normal operation (non boot operation) even though boot operation failed.

If the acknowledge is received before the RSI_BACK_TOUT counter expires, the RSI enters the WAIT_R state once the data transfer is enabled by writing to the RSI_DATA_CTL register and the RSI waits for the start bit of the data.

If the RSI_CFG.BACKEN bit is 0, then no acknowledgment is expected and RSI is in WAIT_R state once the data transfer is enabled by writing to the RSI_DATA_CTL register.

The slave should start sending the boot data before the RSI_DATA_TMR expires. If the counter expires while the state machine is in the WAIT_R state, the boot data timeout interrupt is generated and the RSI_STAT0.BDATTO flag is set. The RSI goes to the IDLE state after a timeout.

When the entire boot data is received, the core can terminate the boot operation by clearing RSI_CFG.MMCBEN bit and sending the CMD0 command and the RSI_BOOT_TCNTR starts decrementing. Once the counter expires, the boot hold time expired interrupt is generated and the RSI_STAT0.BHOLDEXP bit is set. The RSI can now start normal operation (non boot operation) even though the boot operation failed.

The slave is now ready for the CMD1 operation and the master should initiate a normal MMC initialization sequence by sending CMD1.

NOTE: If booting is abruptly terminated for any reason, booting cannot be retried without powering off and powering on the MMC card.

Sleep Mode

MultiMedia cards (MMC) based on the MMCA specification version 4.3 or later support sleep mode. A card may be switched between the sleep state and a standby state using the SLEEP/WAKEUP command (CMD5).

In the sleep state, power consumption of the memory device is minimized. The SLEEP command is used to initiate the state transition from the standby state to the sleep state. The memory device indicates the transition phase is busy by pulling the RSI_D0 line low. The sleep state is reached when the memory device stops pulling down the RSI_D0 line low (RSI_D0 goes high).

The WAKEUP command is used to initiate the state transition from sleep to standby. The memory device indicates the transition phase busy by pulling down the RSI_D0 line low. The standby state is reached when the memory device stops pulling down the RSI_D0 line low (RSI_D0 goes high).

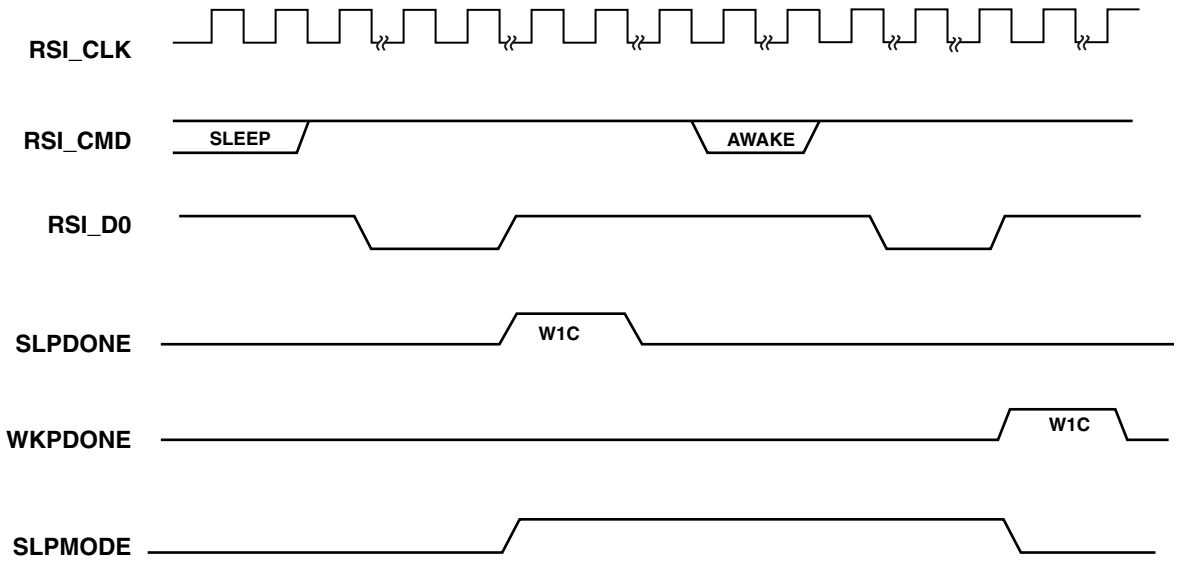


Figure 24-9: Sleep Walk-up Operation

When in standby state, the card is allowed to go to sleep mode by sending the CMD5 with bit 15 set in the argument. The RSI_SLP_WKUP_TOUT counter starts decrementing at the end bit of this command. The RSI block now waits for a rising edge on the RSI_D0 line. If the rising edge is not detected on the RSI_D0 line before the counter expires, the sleep walk-up timer expired interrupt is generated and the corresponding RSI_STAT0.SLPWKPTOUT bit is set.

If a rising edge is detected on the RSI_D0 pin before the RSI_SLP_WKUP_TOUT counter expires, The card entered sleep state interrupt is generated and the RSI_STAT0.SLPDONE and RSI_STAT0.SLPMODE bits are set. This indicates that the card has successfully entered the sleep state. The RSI_STAT0.SLPDONE bit is a W1C bit while the RSI_STAT0.SLPMODE bit is a read-only bit and can not be cleared by writing 1. At this point the power supply for the card may be switched off. Ramp the power supply back up prior to initiating the state transition (by sending the WAKEUP command) from sleep to standby.

When the WAKEUP command is issued by sending CMD5 with bit 15 as 0 in the argument, the RSI_SLP_WKUP_TOUT counter starts decrementing. If a rising edge on the RSI_D0 line is not detected before the counter expires, the sleep walk-up timer expired interrupt is generated and the corresponding RSI_STAT0.SLPWKPTOUT bit is set.

If a rising edge is detected on RSI_D0 pin before the counter expires, the card entered standby state interrupt is generated and the corresponding RSI_STAT0.WKPDONE bit is set and the RSI_STAT0.SLPMODE register is cleared. This indicates that the card has successfully entered the standby state and normal operation can now be resumed.

A SLEEP command is recognized only if the card is in the standby state and the WAKEUP command is recognized only when the card is in the sleep state.

RSI Event Control

This section provides details about various RSI interrupt, status, and error signals.

RSI Interrupt Signals

The RSI has interrupt signals that connect the module to the System Event Controller (SEC). For more information, see the "ADSP-BF60x Interrupt List" section in the *System Event Controller (SEC)* chapter.

RSI Status and Error Signals

The RSI block has 22 individual status bits in the RSI_XFRSTAT register that can be configured to generate an interrupt. The status bits can be mapped to either of the two interrupts RSI_INT0 or RSI_INT1. This allows for greater flexibility in system configuration. To generate an interrupt on RSI_INT0, the interrupt should be enabled by setting the corresponding bit in the RSI_XFR_IMSK0 register. Interrupts that are required to be generated on RSI_INT1 are enabled by setting the corresponding bit in the RSI_XFR_IMSK1 register.

In addition the RSI_XFRSTAT register, each of the flags in the RSI_STAT0 register are also capable of generating an interrupt. Interrupts for the RSI_STAT0 flags are enabled by setting the corresponding bit in the RSI_IMSK0 register. These interrupts are sent to the SEC through RSI_INT0 only.

RSI Programming Model

The following sections describe the RSI programming model for various RSI operating modes.

Card Identification

The following sections describe SD and MMC card identification.

SD Card Identification

Use the following procedure to identify a SD card.

1. Issue the IDLE command to the card using the RSI_CMD register.
2. Issue the SEND_IF command through the RSI_CMD register, supplying the host supply voltage and a check pattern via the RSI_ARG register.
   - If a valid response with a compatible voltage range and matching check pattern is received, the card is compliant with SD veSDHon 2.00 or later.
   - If a response is received with an incompatible voltage range the card cannot be used.
   - If no response is received (indicated by the RSI_XFRSTAT.CMDTO bit), go to step 5.

   STEP RESULT: The command expects an R7 response type.
3. Issue the RSI_SEND_OP_COND command through the RSI_CMD register, supplying the voltage window supported and whether the host supports high capacity cards using the RSI_ARG register.
   - If the card remains busy, or no valid responses have been received within one second, the card is rejected.
   - If the card returns a response indicating that it is busy, resend the RSI_SEND_OP_CMD until the card indicates it is ready.
   - If the host does not support the high capacity mode (as indicated by the HCS bit=0 of the argument), the busy status bit is never cleared.

   STEP RESULT: The RSI expects an R3 response to this command. The RSI can reject the card if the voltage range is not compatible.
4. If the host supports high-capacity cards, verify whether the response in the RSI_RESP0 register indicates if the card capacity status (CCS) bit is set.
   - If CCS is set, an SD VeSDHon 2.00 or later high capacity SD memory card is present—proceed to step 5.
   - If the CCS bit is cleared, then the card is an SD VeSDHon 2.00 or later standard capacity memory card— proceed to step 5.

   a. Issue the RSI_SEND_OP_COND command via the RSI_CMD register, supplying the voltage window supported and with the high-capacity support (HCS) bit set to 0 via the RSI_ARG register.

   STEP RESULT: The RSI expects an R3 response to this command, at which time the card can be rejected if the voltage range is not compatible.

   b. If the card returns a response indicating that it is busy, resend the RSI_SEND_OP_CMD until the card indicates that it is ready.

STEP RESULT: The card should be identified within 1 second. If in that time frame the card is still busy or no valid responses have been received, the card should be rejected. Once the response indicates that the card is ready, the card type has been identified as an SD Version 1.x standard-capacity memory card.

5. Issue the ALL_SEND_CID command.

   STEP RESULT: An R2 response type is expected. This results in the card sending the 128-bit card identification (CID) register and transitioning from ready to identification mode.

6. Issue the SEND_RELATIVE_ADDR command.

   STEP RESULT: An R6 response type is expected. This results in the card issuing a new relative address which must be used to select the card for future data transfers.

RESULT:

The card then moves into standby mode, completing the identification procedure.

MMC Card Identification Procedure

Use the following procedure to identify a MMC card.

1. Issue the IDLE command to the card using the `RSI_CMD` register.
2. Issue the SEND_OP_COND command to the card using the `RSI_CMD` register, supplying the operating voltage window that the host is compatible with and the access mode that the host supports (byte or sector) using the `RSI_ARG` register.

   STEP RESULT: The RSI expects an R3 type response. This allows the host to reject the card if it is not compatible with the supply voltage or if the access mode is not supported by the host software. If the card returns an indication that it is busy, repeat this step until the card is either rejected or not busy.

3. Issue the ALL_SEND_CID command using the `RSI_CMD` register.

   STEP RESULT: The RSI expects an R2 response to this command. This results in the card sending the 128-bit card identification (CID) register and transitioning from ready to identification mode.

4. Issue the SET_RELATIVE_ADDR command, providing a 16-bit relative card address (RCA) using the `RSI_ARG` register that is assigned to the card.

   STEP RESULT: An R1 response type is expected for this command. This results in the card being assigned with the RCA provided, which must be used to select the card for future data transfers.

RESULT:

The card moves into standby mode, completing the identification procedure.

Data Transfer

The following section describe how to program the RSI block for various data transfer scenarios in both core and DMA modes.

Single Block Writes

Block write operations typically consist of 512 bytes of data per block. If the card is found to support other block lengths or the default block length as specified in the CID register is not 512, the block length of the RSI must be configured accordingly. The block length of the card and the block length of the RSI must be configured for the same block size at all times. The block length of the RSI is configured via the RSI_BLKSZ register.

CAUTION: It is important to know when the data path state machine is enabled and when data is written to the FIFO for transfer to the card. Write transactions require that data be written after the response has completed for the WRITE_BLOCK command. If the data path state machine is enabled before sending the WRITE_BLOCK command, data must not be written to the transmit FIFO until after the response has been received as indicated by the RSI_XFRSTAT.RESPEND bit. Failure to adhere to this procedure results in data being written to the card in violation of the block write timing parameters, causing a data CRC failure.

Single Block Core Write

Use the following procedure to perform a single block core write.

1. Write the card's RCA to the upper 16-bits of the RSI_ARG register.
2. Write the RSI_CMD register with the SELECT/DESELECT_CARD command and configure the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

   STEP RESULT: The response type is R1b.
3. Wait for the RSI_XFRSTAT.RESPEND bit to be set then W1C the RSI_XFRSTAT_CLR.RESPEND bit.
4. Use the RSI_RESP0 register to ensure that the device is not busy and no errors occurred.
5. Configure the number of bytes to be transferred to the RSI_DATA_LEN register.

   ADDITIONAL INFORMATION: This is 512 bytes for a single block.
6. Write the appropriate timeout value for a write operation to the RSI_DATA_TMR register.
7. Write the destination start address to the RSI_ARG register.

   ADDITIONAL INFORMATION: The address must be aligned to a 512 byte boundary if misaligned accesses are not enabled and the card is not a high-capacity SD card or sector-addressable MMC card.

8. Write the WRITE_BLOCK command to the RSI_CMD register, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

   STEP RESULT: The response type is R1.

9. Wait for the command response end indication in the RSI_XFRSTAT.RESPEND bit. When detected, W1C the RSI_XFRSTAT_CLR.RESPEND bit.

10. Configure the RSI_BLKSZ register with the value set to 512 for a 512-byte block.

    ADDITIONAL INFORMATION: All other fields of the RSI_DATA_CTL register should be zero.

11. Set the RSI_DATA_CTL.DATEN bit to enable the data path state machine.

12. Write data to the RSI_FIFO register until the FIFO becomes full as indicated by the RSI_XFRSTAT.TXFIFOFULL bit.

    a. Continue to write data to the FIFO as long as the FIFO is not full. Optionally programs can write data in blocks of eight 32-bit words while the RSI_XFRSTAT.TXFIFOSTAT bit =1 (transmit FIFO is half empty).

    b. Continue until all 128 32-bit words (512 bytes) have been transferred.

13. Wait for the card to respond with the CRC token, indicated when the RSI_XFRSTAT.DATBLKEND bit =1.

    ADDITIONAL INFORMATION: The RSI_XFRSTAT.DATEND bit is also set at this time if the RSI_DATA_LEN register was configured for 512 bytes in step 5.

14. W1C the RSI_XFRSTAT_CLR.DATBLKEND and RSI_XFRSTAT_CLR.DATEND bits to clear the RSI_XFRSTAT.DATBLKEND and RSI_XFRSTAT.DATEND bits.

Single Block DMA Writes

Use the following procedure to perform a single block DMA write.

1. Write the card's RCA to the upper 16-bits of the RSI_ARG register.

2. Write the RSI_CMD register with the SELECT/DESELECT_CARD command, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

   STEP RESULT: The response type is R1b.

3. Wait for the RSI_XFRSTAT.RESPEND bit to be set then W1C the RSI_XFRSTAT_CLR.RESPEND bit.

4. Ensure that the device is not busy and no errors occurred by verifying the response contained in the RSI_RESP0 register.

5. Configure the DMA channel assigned to the RSI controller.
   a. Configure the DMA_ADDRSTART register with the address of the first byte of data to be written to the card.
   b. Configure the DMA_XCNT register to 128, and the DMA_XMOD register to 4.
   c. Set the DMA_CFG.EN=1 (DMA enable) and DMA_CFG.MSIZE=2 (word size of 32-bits).
6. Once the DMA channel has been configured and enabled, write the number of bytes to be transferred to the RSI_DATA_LEN register.

   ADDITIONAL INFORMATION: This is 512 bytes for a single block.
7. Write the appropriate timeout value for a write operation to the RSI_DATA_TMR register.
8. Write the destination start address to the RSI_ARG register.

   ADDITIONAL INFORMATION: The address must be aligned to a 512-byte boundary if misaligned accesses are not enabled and the card is not a high-capacity SD card or sector-addressable MMC card.
9. Write the WRITE_BLOCK command to the RSI_CMD register, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

   STEP RESULT: The response type is R1.
10. Wait for the command response end indication in the RSI_XFRSTAT.RESPEND bit and W1C the RSI_XFRSTAT_CLR.RESPEND bit.
11. Configure the RSI_BLKSZ register with the value set to 512 for a 512-byte block.

    ADDITIONAL INFORMATION: All other fields of the RSI_DATA_CTL register should be zero.
12. Set RSI_DATA_CTL.DATEN=1 (enable data path state machine), and RSI_DATA_CTL.DMAEN=1 (DMA enabled).
13. Wait for the card to respond with the CRC token, indicated when the RSI_XFRSTAT.DATBLKEND bit =1.

    ADDITIONAL INFORMATION: The RSI_XFRSTAT.DATEND bit is also set at this point if the RSI_DATA_LEN register was set to 512 bytes in step 5.
14. W1C the RSI_XFRSTAT_CLR.DATBLKEND and RSI_XFRSTAT_CLR.DATEND to clear the RSI_XFRSTAT.DATBLKEND and RSI_XFRSTAT.DATEND bits. Also clear the DMA_STAT.IRQDONE bit if applicable.

Single Block Reads

Block read operations typically consist of 512 bytes of data per block. If the card is found to support other block lengths or the default block length as specified in the CID register is not 512, the block length of the RSI must be configured accordingly. The block length of the card and the block length of the RSI must be configured for the same block size at all times. The block length of the RSI is configured using the RSI_BLKSZ register.

NOTE: For data transfers from the card to the RSI, it is important to know when the data path state machine is enabled and when data is read from the receive FIFO. This is because read transactions can occur on the RSI_D0 – RSI_D7 signals prior to the response of the command being received. Therefore the data path state machine and DMA controller (if used) should be enabled either:

- Prior to issuing a command that involves a data read packet
- Immediately after the command has been issued but prior to pending on the RSI_XFRSTAT.RESPEND flag

NOTE: If the core is being used to read the receive FIFO, it is advised not to depend on the command response end (RSI_XFRSTAT.RESPEND) flag to determine in the command response end. This is because data can be driven on the RSI_D0 – RSI_D7 signals two RSI_CLK cycles after the end bit of the command. At a minimum, an additional 48 RSI_CLK cycles pass before the response is received, during which time the receive buffer may potentially have received 24 bytes of data on a 4-bit bus and approaches the half full state. Software should ensure that the receive buffer does not become full prior to data being read from the receive FIFO.

Single Block Core Reads

Use the following procedure to perform a single block core read.

1. Write the card's RCA to the upper 16-bits of the RSI_ARG register.
2. Write the RSI_CMD register with the SELECT/DESELECT_CARD command, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

   STEP RESULT: The response type is R1b.
3. Wait for the RSI_XFRSTAT.RESPEND bit to be set then W1C the RSI_XFRSTAT_CLR.RESPEND bit.
4. Ensure that the device is not busy and no errors occurred by verifying the response contained in the RSI_RESP0 register.
5. Write the number of bytes to be transferred to the RSI_DATA_LEN register.

   ADDITIONAL INFORMATION: This is 512 bytes for a single block.
6. Configure the RSI_DATA_TMR register with the appropriate timeout value for a read operation.
7. Write the destination start address to the RSI_ARG register.

   ADDITIONAL INFORMATION: The address supplied must be aligned to a 512-byte boundary if misaligned accesses are not enabled and the card is not a high-capacity SD card or sector-addressable MMC card.
8. Configure the RSI_DATA_CTL register with the RSI_BLKSZ register value configured to 512 for a 512-byte block.

   ADDITIONAL INFORMATION: All other fields of the RSI_DATA_CTL register should be zero.

9. Set the RSI_DATA_CTL.DATEN bit to enable the data path state machine and the RSI_DATA_CTL.DATDIR bit to configure the transfer direction from the card to the controller.
10. Write the READ_SINGLE_BLOCK command to the RSI_CMD register, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

    STEP RESULT: The response type is R1.

NOTE: In order to meet some timing restrictions related to block read operations, do not wait for the RSI_XFRSTAT.RESPEND bit to be set—move immediately to the next step. Because the card can send data before a response can be completed on the RSI_CMD signal, moving immediately to step 11 ensures a receive FIFO overflow does not occur.

11. Poll the RSI_XFRSTAT.RXFIFORDY bit or the RSI_XFRSTAT.RXFIFOZERO bit which indicate the receive FIFO has data available, or the receive FIFO is empty. As long as the receive FIFO is not empty, read data from the RSI_FIFO register until all 512 bytes have been read.
12. Once all bytes have been read, wait for the RSI_XFRSTAT.DATBLKEND bit to indicate that the data was received correctly and passed the CRC check. The RSI_XFRSTAT.DATEND bit may also be set, depending on the value written to the RSI_DATA_LEN register.
13. W1C the RSI_XFRSTAT_CLR.DATBLKEND and RSI_XFRSTAT_CLR.DATEND bits to clear the RSI_XFRSTAT.DATBLKEND and RSI_XFRSTAT.DATEND bits.

Single Block DMA Reads

Use the following procedure to perform a single block DMA read.

1. Write the card's RCA to the upper 16-bits of the RSI_ARG register.
2. Write the RSI_CMD register with the SELECT/DESELECT_CARD command, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

    STEP RESULT: The response type is R1b.
3. Wait for the RSI_XFRSTAT.RESPEND bit to be set then W1C the RSI_XFRSTAT_CLR.RESPEND bit.
4. Ensure that the device is not busy and no errors occurred by verifying the response contained in the RSI_RESP0 register.

5. Configure the DMA channel assigned to the RSI controller.
   a. Configure the DMA_ADDRSTART register with the address of the first byte of where the received data is to be stored.
   b. Configure the DMA_XCNT register to 128 and the DMA_XMOD register to 4.
   c. Set DMA_CFG.EN=1 (DMA enable), DMA_CFG.MSIZE=2 (word size of 32-bits) and DMA_CFG.WNR=0 (memory write).
6. Write the number of bytes to be transferred to the RSI_DATA_LEN register.

   ADDITIONAL INFORMATION: This is 512 bytes for a single block.
7. Write the appropriate timeout value for a read operation to the RSI_DATA_TMR register.
8. Write the source start address to the RSI_ARG register.

   ADDITIONAL INFORMATION: The address must be aligned to a 512-byte boundary if misaligned accesses are not enabled and the card is not a high-capacity SD card or sector-addressable MMC card.
9. Configure the RSI_BLKSZ register value to 512 for a 512-byte block.

   ADDITIONAL INFORMATION: All other fields of the RSI_DATA_CTL register should be zero.
10. Set RSI_DATA_CTL.DATEN=1 (enable data path state machine), RSI_DATA_CTL.DATDIR=1 (transfer direction from the card to the controller), and RSI_DATA_CTL.DMAEN=1 (DMA enabled).
11. Write the READ_SINGLE_BLOCK command to the RSI_CMD register, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

    STEP RESULT: The response type is R1.
12. Poll the RSI_XFRSTAT.RESPEND bit and W1C the RSI_XFRSTAT_CLR.RESPEND register when the command response end is detected.

    ADDITIONAL INFORMATION: Unlike core accesses, it is safe to perform this step. The DMA controller (enabled in step 5) ensures that any data sent to the receive FIFO prior to the command response end flag being set is received correctly.
13. Wait for the data block end flag to indicate that the data was received correctly and passed the CRC check. The data end flag may also be set, depending on the value written to the RSI_DATA_LEN register.
14. W1C the RSI_XFRSTAT_CLR.DATBLKEND and RSI_XFRSTAT_CLR.DATEND bits to clear the RSI_XFRSTAT.DATBLKEND and RSI_XFRSTAT.DATEND bits. Also clear the DMA_STAT.IRQDONE bit if applicable.

Multiple Block Writes

Block write operations typically consist of 512 bytes of data per block. If the card is found to support other block lengths or the default block length as specified in the CID register is not 512, the block length of the RSI must be configured accordingly. The block length of the card and the block length of the RSI must be configured for the same block size at all times. The block length of the RSI is configured using the RSI_BLKSZ register.

Multiple Block Core Write

1. Write the card's RCA to the upper 16-bits of the RSI_ARG register.
2. Write the RSI_CMD register with the SELECT/DESELECT_CARD command, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

   STEP RESULT: The response type is R1b.
3. Wait for the RSI_XFRSTAT.RESPEND bit to be set then W1C the RSI_XFRSTAT_CLR.RESPEND bit.
4. Ensure that the device is not busy and no errors occurred by verifying the response contained in the RSI_RESP0 register.
5. Write the number of bytes to be transferred to the RSI_DATA_LEN register.

   ADDITIONAL INFORMATION: For example, write 4096 to write eight blocks of 512 bytes.
6. Write the appropriate timeout value for a write operation to the RSI_DATA_TMR register.
7. Write the destination start address to the RSI_ARG register.

   ADDITIONAL INFORMATION: The supplied address must be aligned to a 512-byte boundary if misaligned accesses are not enabled and the card is not a high-capacity SD card or a sector-addressable MMC card.
8. Write the WRITE_MULTIPLE_BLOCK command to the RSI_CMD register, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

   STEP RESULT: The response type is R1.
9. Wait for the command response end indication in the RSI_XFRSTAT.RESPEND bit. When detected, W1C the RSI_XFRSTAT_CLR.RESPEND bit.
10. Configure the RSI_BLKSZ register with the value set to 512 for a 512-byte block.

    ADDITIONAL INFORMATION: All other fields of the RSI_DATA_CTL register should be zero.
11. Set the RSI_DATA_CTL.DATEN bit to enable the data path state machine.

12. Write data to the RSI_FIFO register until the FIFO becomes full as indicated by the RSI_XFRSTAT.TXFIFOFULL bit.
    a. Continue to write data to the FIFO as long as the FIFO is not full. Optionally programs can write data in blocks of eight 32-bit words while the RSI_XFRSTAT.TXFIFOSTAT bit =1 (transmit FIFO is half empty).
    b. Continue until all 128 32-bit words (512 bytes) have been transferred.
13. Wait for the card to respond with the CRC token, indicated when the RSI_XFRSTAT.DATBLKEND bit =1.
14. W1C the RSI_XFRSTAT_CLR.DATBLKEND bit to clear the RSI_XFRSTAT.DATBLKEND bit.
15. Repeat steps 11 to 13 for the number of blocks to be transferred or until the RSI_XFRSTAT.DATEND bit is set.
16. Write the RSI_CMD register with the STOP_TRANSMISSION command, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

    STEP RESULT: The response type is R1. 16.
17. W1C the RSI_XFRSTAT_CLR.DATBLKEND and RSI_XFRSTAT_CLR.DATEND bits to clear the RSI_XFRSTAT.DATBLKEND and RSI_XFRSTAT.DATEND bits.

Multiple Block DMA Writes

Use the following procedure to perform a multiple block DMA write.

1. Write the card's RCA to the upper 16-bits of the RSI_ARG register.
2. Write the RSI_CMD register with the SELECT/DESELECT_CARD command, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

   STEP RESULT: The response type is R1b.
3. Wait for the RSI_XFRSTAT.RESPEND bit to be set then W1C the RSI_XFRSTAT_CLR.RESPEND bit.
4. Ensure that the device is not busy and no errors occurred by verifying the response contained in the RSI_RESP0 register.

5. Configure the DMA channel assigned to the RSI controller.
   a. Configure the DMA_ADDRSTART register with the address of the first byte of data to be written to the card.
   b. Configure the DMA_XCNT register to the overall number of 32-bit words to be written. For example, write 1024 to transfer 4096 bytes.
   c. Set the DMA_XMOD register to 4.
   d. Set the DMA_CFG.EN=1 (DMA enable) and DMA_CFG.MSIZE=2 (word size of 32-bits).
6. Once the DMA channel has been configured and enabled, write the number of bytes to be transferred to the RSI_DATA_LEN register.

   ADDITIONAL INFORMATION: For example, write 4096 to write eight blocks of 512 bytes.
7. Write the appropriate timeout value for a write operation to the RSI_DATA_TMR register.
8. Write the destination start address to the RSI_ARG register.

   ADDITIONAL INFORMATION: The supplied address must be aligned to a 512-byte boundary if misaligned accesses are not enabled and the card is not a high-capacity SD card or sector addressable MMC card.
9. Write the WRITE_MULTIPLE_BLOCK command to the RSI_CMD register, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

   STEP RESULT: The response type is R1.
10. Wait for the command response end indication in the RSI_XFRSTAT.RESPEND bit and W1C the RSI_XFRSTAT_CLR.RESPEND bit.
11. Configure the RSI_BLKSZ register with the value set to 512 for a 512-byte block.

    ADDITIONAL INFORMATION: All other fields of the RSI_DATA_CTL register should be zero.
12. Set RSI_DATA_CTL.DATEN=1 (enable data path state machine), and RSI_DATA_CTL.DMAEN=1 (DMA enabled).
13. Poll the RSI_XFRSTAT.DATEND bit or alternatively poll for each instance of the RSI_XFRSTAT.DATBLKEND bit that is set on successful completion of each block transfer.

    ADDITIONAL INFORMATION: For a 4096 byte transfer, RSI_XFRSTAT.DATBLKEND is set eight times and should be cleared after it is detected using the RSI_XFRSTAT_CLR.DATBLKEND bit.

14. Write the RSI_CMD register with the STOP_TRANSMISSION command, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

    STEP RESULT: The response type is R1.

15. W1C the RSI_XFRSTAT_CLR.DATBLKEND and RSI_XFRSTAT_CLR.DATEND to clear the RSI_XFRSTAT.DATBLKEND and RSI_XFRSTAT.DATEND bits. Also clear the DMA_STAT.IRQDONE bit if applicable.

Multiple Block Read

Block read operations typically consist of 512 bytes of data per block. If the card is found to support other block lengths or the default block length as specified in the CID register is not 512, the block length of the RSI must be configured accordingly. The block length of the card and the block length of the RSI must be configured for the same block size at all times. The block length of the RSI is configured in the RSI_BLKSZ register.

Multiple Block Core Reads

Use the following procedure to perform a multiple block core read.

1. Write the card's RCA to the upper 16-bits of the RSI_ARG register.
2. Write the RSI_CMD register with the SELECT/DESELECT_CARD command, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

   STEP RESULT: The response type is R1b.

3. Wait for the RSI_XFRSTAT.RESPEND bit to be set then W1C the RSI_XFRSTAT_CLR.RESPEND bit.
4. Ensure that the device is not busy and no errors occurred by verifying the response contained in the RSI_RESP0 register.
5. Write the number of bytes to be transferred to the RSI_DATA_LEN register.

   ADDITIONAL INFORMATION: This is 512 bytes for a single block.

6. Write the appropriate timeout value for a read operation to the RSI_DATA_TMR register.
7. Write the destination start address to the RSI_ARG register.

   ADDITIONAL INFORMATION: The address must be aligned to a 512-byte boundary if misaligned accesses are not enabled and the card is not a high-capacity SD card or sector-addressable MMC card.

8. Configure the RSI_DATA_CTL register with the RSI_BLKSZ register value configured to 512 for a 512-byte block.

   ADDITIONAL INFORMATION: All other fields of the RSI_DATA_CTL register should be zero.

9. Set the RSI_DATA_CTL.DATEN bit to enable the data path state machine and the RSI_DATA_CTL.DATDIR bit to configure the transfer direction from the card to the controller.
10. Write the READ_MULTIPLE_BLOCK command to the RSI_CMD register, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

    STEP RESULT: The response type is R1.

NOTE: In order to meet some timing restrictions related to block read operations, do not wait for the RSI_XFRSTAT.RESPEND bit to be set—move immediately to the next step. Because the card can send data before a response can be completed on the RSI_CMD signal, moving immediately to step 11 ensures a receive FIFO overflow does not occur.

11. Poll the RSI_XFRSTAT.RXFIFORDY bit or the RSI_XFRSTAT.RXFIFOZERO bit which indicate the receive FIFO has data available, or the receive FIFO is empty. As long as the receive FIFO is not empty, read data from the RSI_FIFO register until all 512 bytes have been read.
12. Once the block has been read, wait for the RSI_XFRSTAT.DATBLKEND bit to be set indicating that the data was received correctly and passed the CRC check the W1C the RSI_XFRSTAT_CLR.DATBLKEND bit.
13. Repeat steps 11 and 12 until the required number of blocks have been read or until the RSI_XFRSTAT.DATEND bit has been set.
14. Write the RSI_CMD register with the STOP_TRANSMISSION command, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

    STEP RESULT: The response type is R1.
15. W1C the RSI_XFRSTAT_CLR.DATBLKEND and RSI_XFRSTAT_CLR.DATEND bits to clear the RSI_XFRSTAT.DATBLKEND and RSI_XFRSTAT.DATEND bits.

Multiple Block DMA Reads

Use the following procedure to perform a multiple block DMA read.

1. Write the card's RCA to the upper 16-bits of the RSI_ARG register.
2. Write the RSI_CMD register with the SELECT/DESELECT_CARD command, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

    STEP RESULT: The response type is R1b.
3. Wait for the RSI_XFRSTAT.RESPEND bit to be set then W1Cthe RSI_XFRSTAT_CLR.RESPEND bit.
4. Ensure that the device is not busy and no errors occurred by verifying the response contained in the RSI_RESP0 register.

5. Configure the DMA channel assigned to the RSI controller.
   a. Write the DMA_ADDRSTART register with the address of the first byte of where the received data is to be stored.
   b. Configure the DMA_XCNT register to the number of 32-bit words to be read, which is 1024 for a 4096 byte read transfer. Configure the DMA_XMOD register to 4.
   c. Set DMA_CFG.EN=1 (DMA enable), DMA_CFG.MSIZE=2 (word size of 32-bits) and DMA_CFG.WNR=0 (memory write).
6. Write the number of bytes to be transferred to the RSI_DATA_LEN register.

   ADDITIONAL INFORMATION: This is 4096 for eight blocks of 512 bytes.
7. Write the appropriate timeout value for a read operation to the RSI_DATA_TMR register.
8. Write the source start address to the RSI_ARG register.

   ADDITIONAL INFORMATION: The address must be aligned to a 512-byte boundary if misaligned accesses are not enabled and the card is not a high-capacity SD card or a sector-addressable MMC card.
9. Configure the RSI_BLKSZ register value to 512 for a 512-byte block.

   ADDITIONAL INFORMATION: All other fields of the RSI_DATA_CTL register should be zero.
10. Set RSI_DATA_CTL.DATEN=1 (enable data path state machine), RSI_DATA_CTL.DATDIR=1 (transfer direction from the card to the controller), and RSI_DATA_CTL.DMAEN=1 (DMA enabled).
11. Write the READ_MULTIPLE_BLOCK command to the RSI_CMD register, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

    STEP RESULT: The response type is R1.

NOTE: Unlike core accesses, it is safe to poll on RSI_XFRSTAT.RESPEND indication and W1C the RSI_XFRSTAT_CLR.RESPEND bit. The DMA controller ensures any data sent to the receive FIFO prior to the RSI_XFRSTAT.RESPEND flag being set is received correctly.

12. Poll for the RSI_XFRSTAT.DATEND bit or alternatively poll for each instance of the RSI_XFRSTAT.DATBLKEND bit that is set on successful completion of each block transfer. For a 4096-byte transfer, RSI_XFRSTAT.DATBLKEND is set eight times and should be cleared after it is detected using a W1C operation in the RSI_XFRSTAT_CLR.DATBLKEND bit.

13. Write the RSI_CMD register with the STOP_TRANSMISSION command, configuring the command path state machine to expect a short response by setting the RSI_CMD.RSP bit and clearing the RSI_CMD.LRSP bit.

    STEP RESULT: The response type is R1.

14. W1C the RSI_XFRSTAT_CLR.DATBLKEND and RSI_XFRSTAT_CLR.DATEND bits to clear the RSI_XFRSTAT.DATBLKEND and RSI_XFRSTAT.DATEND bits. Also clear the DMA_STAT.IRQDONE bit if applicable.

RSI Programming Concepts

This section provides details about some special programming considerations to effectively use the RSI module.

Disabling CRC check

Some memory cards, such as eMMC and SDIO, (and others), do not send CRC status for some commands. In earlier Blackfin processors the RSI took the stuffed bits coming from the card as CRC and generated a CRC check failure interrupt. To disable CRC checking, set RSI_CMD.CRCDIS bit.

Data End Interrupt

In older RSI versions, the data end interrupt (RSI_XFRSTAT.DATEND bit is set) is generated when the Data Counter reaches 0 (RSI_DATA_TMR register). In the new version of RSI this interrupt is delayed in data write operations if the card indicates a busy condition by pulling the RSI_D0 line low. This interrupt is now generated once the RSI_D0 line goes high.

Miscellaneous Programming Guidelines

- The PORTx_FER bit for the RSI_CLK should be set before the RSI_D3 pin is enabled.
- The RSI_CMD and RSI_D0 signals should always be used together. Therefore only one pull-up enable is used for these two signals.
- Write 1 to clear the SD card detect interrupt status bit (RSI_STAT0.SDCARD) after a SDMMC reset is issued.
- There is a 2.5 system clock cycle latency involved with a write to the RSI registers. Any action that needs to be taken after the register write should be timed accordingly. This is especially important in situations where the system clock on which RSI operates is very slow compared to the core domain clock.

ADSP-BF60x RSI Register Descriptions

Removable Storage Interface (RSI) contains the following registers.

Table 24-17: ADSP-BF60x RSI Register List

Name	Description
RSI_CTL	Control Register
RSI_ARG	Argument Register
RSI_CMD	Command Register
RSI_RESP_CMD	Response Command Register
RSI_RESP0	Response 0 Register
RSI_RESP1	Response 1 Register
RSI_RESP2	Response 2 Register
RSI_RESP3	Response 3 Register
RSI_DATA_TMR	Data Timer Register
RSI_DATA_LEN	Data Length Register
RSI_DATA_CTL	Data Control Register
RSI_DATA_CNT	Data Count Register
RSI_XFRSTAT	Transfer Status Register
RSI_XFRSTAT_CLR	Transfer Status Clear Register
RSI_XFR_IMSK0	Transfer Interrupt 0 Mask Register
RSI_XFR_IMSK1	Transfer Interrupt 1 Mask Register
RSI_FIFO_CNT	FIFO Counter Register
RSI_BOOT_TCNTR	Boot Timing Counter Register
RSI_BACK_TOUT	Boot Acknowledge Timeout Register
RSI_SLP_WKUP_TOUT	Sleep Wakeup Timeout Register
RSI_BLKSZ	Block Size Register
RSI_FIFO	Data FIFO Register

Table 24-17: ADSP-BF60x RSI Register List (Continued)

Name	Description
RSI_STAT0	Exception Status Register
RSI_IMSK0	Exception Mask Register
RSI_CFG	Configuration Register
RSI_RD_WAIT	Read Wait Enable Register
RSI_PID0	Peripheral ID 0 Register
RSI_PID1	Peripheral ID 1 Register
RSI_PID2	Peripheral ID 2 Register
RSI_PID3	Peripheral ID 3 Register

Control Register

The RSI_CTL register provides control functionality for the RSI_CLK (RSI clock pin). The RSI_CLK can be derived directly from the SCLK signal by enabling RSI_CTL.BYPASS; otherwise, the RSI_CLK frequency is determined from the current SCLK frequency and the RSI_CTL.CLKDIV field according to the formula:

RSI_CLK = (SCLK) / (2 x (CLKDIV +1))

In order to conserve power, the RSI_CLK can be disabled without disabling the entire RSI interface using the RSI_CTL.CLKEN bit; additionally the RSI_CTL.PWRSAVE bit, when set, results in the RSI_CLK signal only been driven when the RSI is performing a transfer either to or from the card. In addition to clock control functionality, the data bus width of the RSI interface is also controlled from this register as well as the type of device that has been identified as being interfaced to the RSI. Bits [13:15] are added to this register. These bits indicate the type of card that is connected to the RSI. The RSI_CTL.CARDTYPE field can be initialized by software to indicate the type of device identified. As a number of commands (such as Sleep, Wakeup) use the RSI_CTL.CARDTYPE information, this field is required to be programmed before the Sleep or Wakeup command is issued to the device.

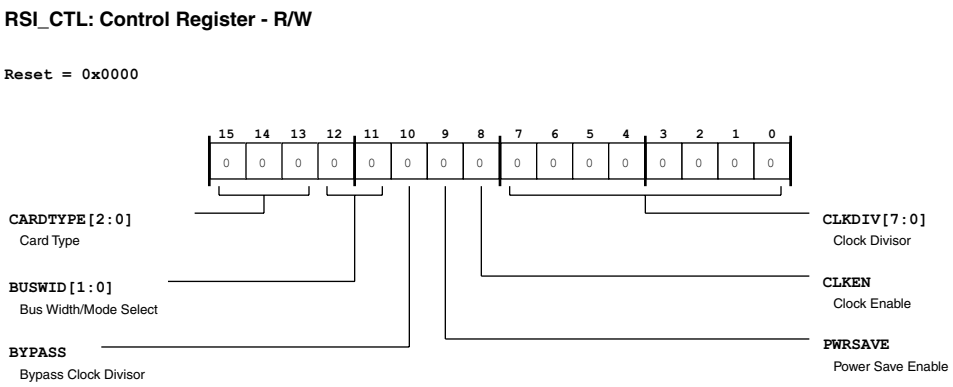


Figure 24-10: RSI_CTL Register Diagram

Table 24-18: RSI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:13 (R/W)	CARDTYPE	Card Type. The RSI_CTL.CARDTYPE bits indicate the type of card connected to the RSI.	
		0	SDIO
		1	eMMC
		2	SD Card
		3	Reserved
12:11 (R/W)	BUSWID	Bus Width/Mode Select. The RSI_CTL.BUSWID bits select the RSI bus width and mode.	
		0	1-bit data bus (Std Bus Mode, uses RSI_D0)
		1	4-bit data bus (Wide Bus Mode)
		2	8-bit data bus (Byte Bus Mode)
		3	Reserved
10 (R/W)	BYPASS	Bypass Clock Divisor. The RSI_CTL.BYPASS bit enables bypass of the RSI clock divisor.	
		0	Disable Bypass
		1	Enable Bypass (SCLK drives RSI_CLK)

Table 24-18: RSI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	PWRSAVE	Power Save Enable. The RSI_CTL.PWRSAVE bit enables power save operation, which enables RSI_CLK only when the bus is active. If this bit is disabled, the RSI_CLK is always enabled when RSI_CTL.CLKEN is set, whether or not the bus is active.	
		0	Disable Power Save
		1	Enable Power Save
8 (R/W)	CLKEN	Clock Enable. The RSI_CTL.CLKEN bit enables the RSI_CLK bus clock.	
		0	Disable Clock
		1	Enable Clock
7:0 (R/W)	CLKDIV	Clock Divisor. The RSI_CTL.CLKDIV bits apply a clock divisor to the RSI_CLK frequency when bypass is disabled (RSI_CTL.BYPASS =0).	

Argument Register

The RSI_ARG register contains the 32-bit argument that is sent on the RSI_CMD pin as part of a command message. If a command requires an argument, the argument must first be loaded into the RSI_ARG register prior to writing and enabling the command in the RSI_CMD register. For more information about RSI commands and responses, see the RSI functional description.

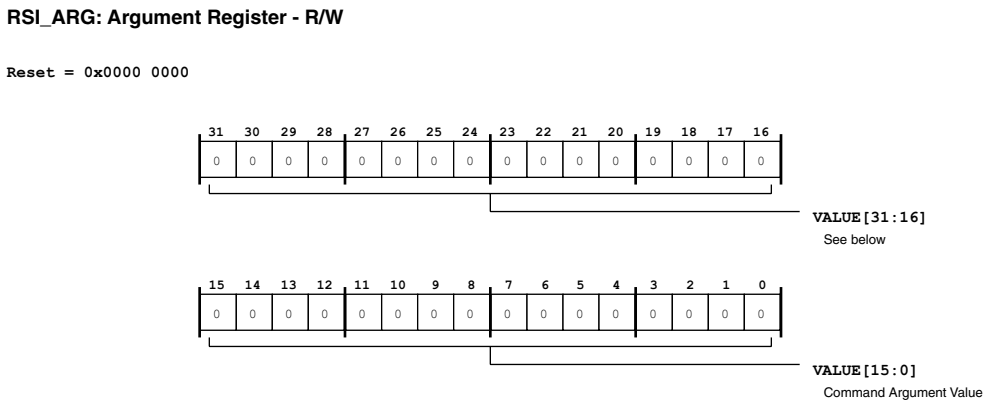


Figure 24-11: RSI_ARG Register Diagram

Table 24-19: RSI_ARG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Command Argument Value. The RSI_ARG.VALUE bits contain the 32-bit argument value that is sent on the RSI_CMD pin as part of a command message.

Command Register

The RSI_CMD register controls the command path state machine. The RSI_CMD.IDX field contains the index of the command to be issued through the RSI as part of the command message. If the command requires a response, this is indicated with the RSI_CMD.RSP bit.

The length of the response (short or long) is selected with the RSI_CMD.LRSP bit. The command path state machine becomes active when the RSI_CMD.EN bit is set and is disabled if this bit is cleared.

NOTE: After a data write, data cannot be written to this register for three SYSCLK periods plus two BCLK periods.

It is not required to manually clear the RSI_CMD.EN bit after the command sequence has completed. The command path state machine automatically terminates and becomes idle after the operation has completed.

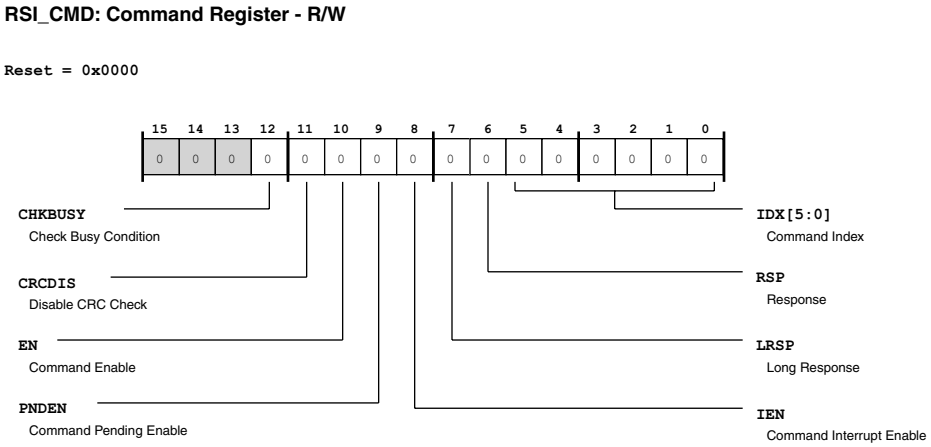


Figure 24-12: RSI_CMD Register Diagram

Table 24-20: RSI_CMD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	CHKBUSY	Check Busy Condition. The RSI_CMD.CHKBUSY bit directs the RSI to check for busy state on the RSI_DO pin.	
		0	No Check for Busy
		1	Check for Busy
11 (R/W)	CRCDIS	Disable CRC Check. The RSI_CMD.CRCDIS bit disables the RSI CRC check operation.	
		0	Enable CRC Check
		1	Disable CRC Check
10 (R/W)	EN	Command Enable. The RSI_CMD.EN bit enables the RSI command state machine operation.	
		0	Disable Command Operation
		1	Enable Command Operation
9 (R/W)	PNDEN	Command Pending Enable. The RSI_CMD.PNDEN bit directs the command state machine to waits for command pending status (RSI_XFRSTAT.DATEND =1) before the RSI starts sending a command. This feature is used only during MMC Stream Mode.	
		0	Disable Command Pending
		1	Enable Command Pending
8 (R/W)	IEN	Command Interrupt Enable. The RSI_CMD.IEN bit disables the timeout mechanism when waiting for a response to be received from an MMC device and instead waits until a card interrupt is detected.	
		0	Disable Command Interrupt
		1	Enable Command Interrupt
7 (R/W)	LRSP	Long Response. The RSI_CMD.LRSP bit directs the command state machine to wait for a 136-bit long response if a response is enabled (RSI_CMD.RSP =1).	
		0	No Wait for Long Response
		1	Wait for Long Response

Table 24-20: RSI_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	RSP	Response. The RSI_CMD.RSP bit directs the command state machine to wait for a short response. When enabled, either a command response end flag (RSI_XFRSTAT.RESPEND =1) or command CRC fail flag (RSI_XFRSTAT.CMDCRCFAIL =1) is expected. When disabled, a command sent flag (RSI_XFRSTAT.CMDSENT =1) is expected.	
		0	No Wait for Short Response
		1	Wait for Short Response
5:0 (R/W)	IDX	Command Index. The RSI_CMD.IDX bits contain the command index (the command number to be issued). For more information about RSI commands and responses, see the RSI functional description.	

Response Command Register

The RSI_RESP_CMD register contains the command index field of the last response received. If the command response does not contain doe not contain a command index field (as is the case with a long response), the contents of the RSI_RESP_CMD register would typically be ignored. In this situation, the register likely contains the value 0x3F, which is the value of the reserved field of the response.

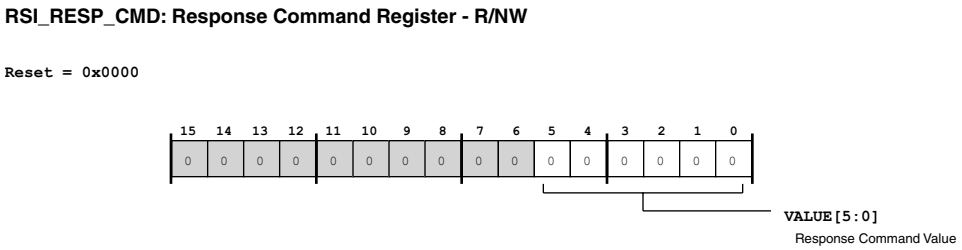


Figure 24-13: RSI_RESP_CMD Register Diagram

Table 24-21: RSI_RESP_CMD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5:0 (R/NW)	VALUE	Response Command Value. The RSI_RESP_CMD.VALUE bits contain the command response value from the last response received.

Response 0 Register

The response registers (RSI_RESP0, RSI_RESP1, RSI_RESP2, RSI_RESP3) contain the response information received back from a card for a given command message. The received response may be 32 or 127 bits in length, depending on whether the response type is short or long. The most significant bit of the response is received first and is located in bit 31 of the RSI_RESP0 register. Bit 0 of RSI_RESP3 is always zero. The table show two example responses (short versus long).

Response Register	Short Response	Long Response
RSI_RESP0	Response bits [31:0]	Response bits [127:96]
RSI_RESP1	Not used	Response bits [95:64]
RSI_RESP2	Not used	Response bits [63:32]
RSI_RESP3	Not used	Response bits [31:1]

Note that bits 31:1 of the long response are stored in bits 30:0 of the RSI_RESP3 register.Bit 31 of the RSI_RESP3 register is not used and is always zero.

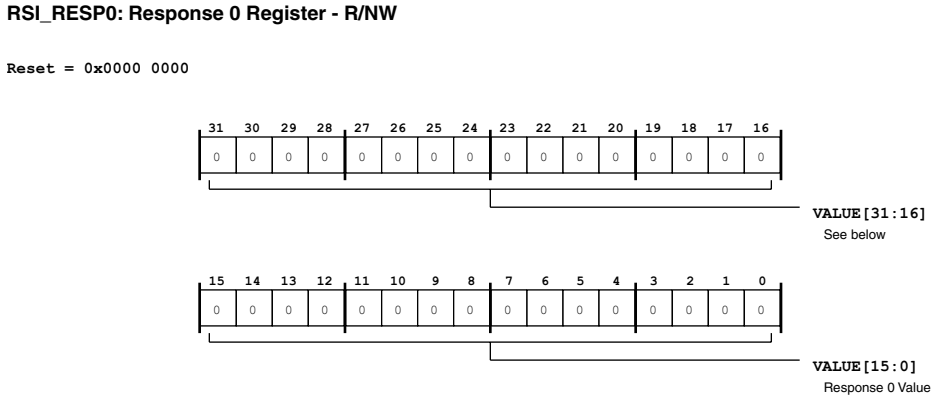


Figure 24-14: RSI_RESP0 Register Diagram

Table 24-22: RSI_RESP0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Response 0 Value. The RSI_RESP0.VALUE bits contain card status bits [31:0] in a short response and contain card status bits [127:96] in a long response.

Response 1 Register

The response registers (RSI_RESP0, RSI_RESP1, RSI_RESP2, RSI_RESP3) contain the response information received back from a card for a given command message. For more information, see the RSI_RESP0 register description.

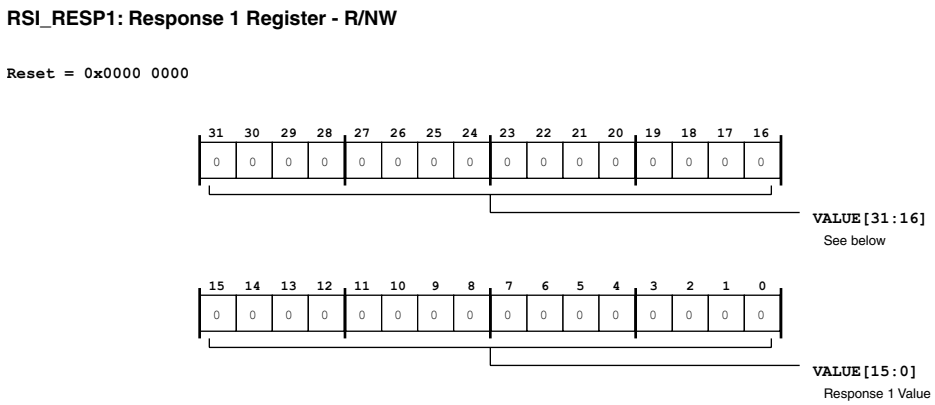


Figure 24-15: RSI_RESP1 Register Diagram

Table 24-23: RSI_RESP1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Response 1 Value. The RSI_RESP1.VALUE bits contain card status bits [95:64] in a long response.

Response 2 Register

The response registers (RSI_RESP0, RSI_RESP1, RSI_RESP2, RSI_RESP3) contain the response information received back from a card for a given command message. For more information, see the RSI_RESP0 register description.

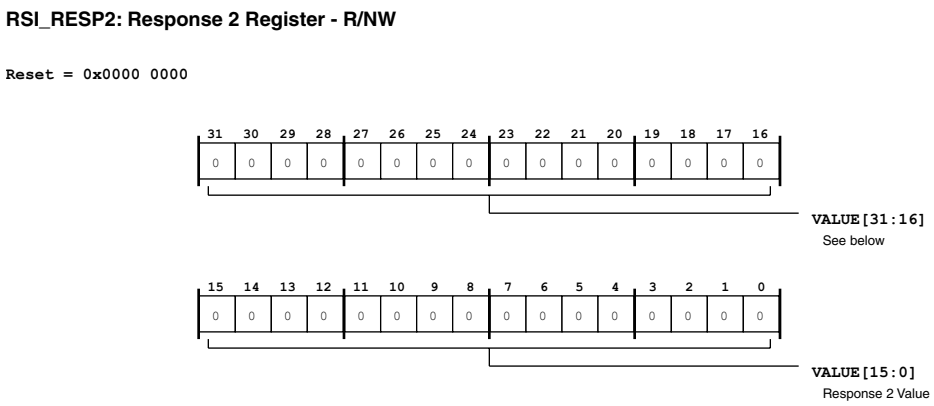


Figure 24-16: RSI_RESP2 Register Diagram

Table 24-24: RSI_RESP2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Response 2 Value. The RSI_RESP2.VALUE bits contain card status bits [63:32] in a long response.

Response 3 Register

The response registers (RSI_RESP0, RSI_RESP1, RSI_RESP2, RSI_RESP3) contain the response information received back from a card for a given command message. For more information, see the RSI_RESP0 register description.

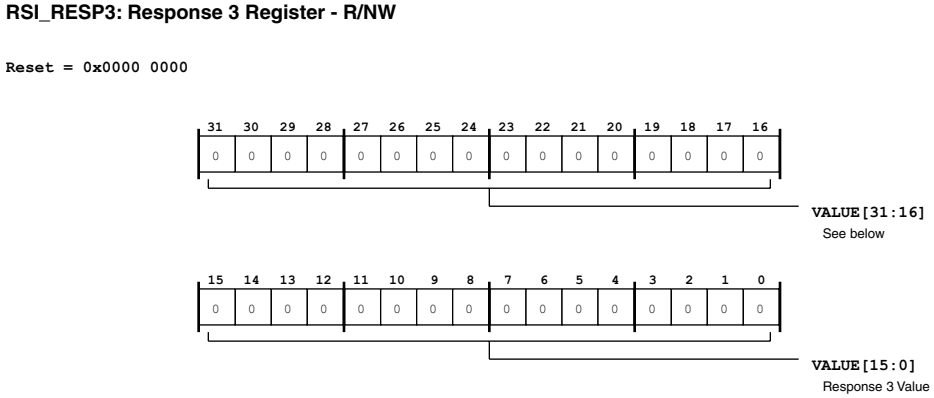


Figure 24-17: RSI_RESP3 Register Diagram

Table 24-25: RSI_RESP3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Response 3 Value. The RSI_RESP3.VALUE bits contain card status bits [31:0] in a long response. The most significant bit of the card status is received first. The most significant bit of RSI_RESP3.VALUE is always zero.

Data Timer Register

The RSI_DATA_TMR register contains a 32-bit value for the data timeout period (in RSI_CLK cycles). An internal counter loads the value from this register, and starts to decrement when the data path state machine enters the WAIT_R or the BUSY states. If the timer decrements to zero while the data path state machine is still in either of these two states, the RSI_XFRSTAT.DATTO flag is set. The RSI_DATA_TMR and the RSI_DATA_LEN registers must both be written to prior to starting a data transfer through the RSI_DATA_CTL register.

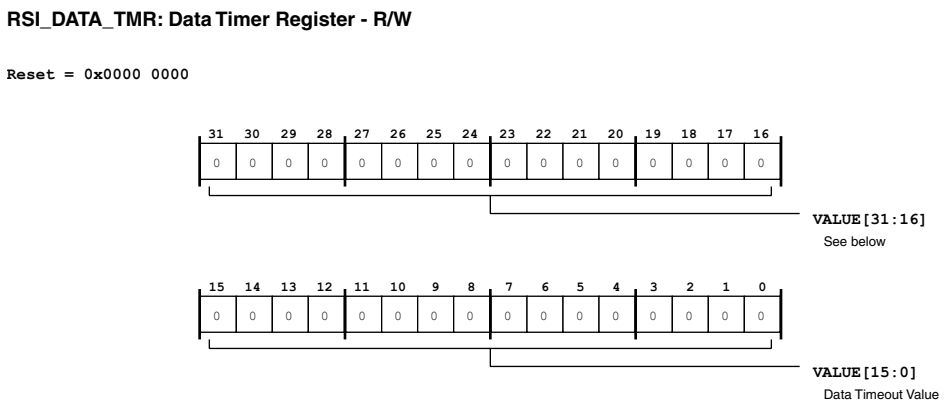


Figure 24-18: RSI_DATA_TMR Register Diagram

Table 24-26: RSI_DATA_TMR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Data Timeout Value. The RSI_DATA_TMR.VALUE bits hold the timeout value (which is used in normal mode) and hold the boot data timeout value (which is used in boot mode).

Data Length Register

The RSI_DATA_LEN register contains a 16-bit value for the number of data bytes to be transferred before setting the RSI_XFRSTAT.DATEND flag. The value loaded to this register is copied into the RSI_DATA_CNT register when the data path state machine is enabled and starts the transfer.

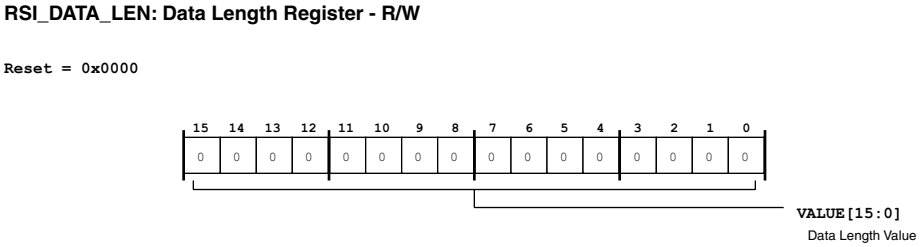


Figure 24-19: RSI_DATA_LEN Register Diagram

Table 24-27: RSI_DATA_LEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Data Length Value. The RSI_DATA_LEN.VALUE bits hold the length value for the number of data bytes to be transferred.

Data Control Register

The RSI_DATA_CTL register largely controls the data path state machine. Data transfer starts when RSI_DATA_CTL.DATEN is set (=1). Depending on the RSI_DATA_CTL.DATDIR bit, the data path state machine moves to the WAIT_S or the WAIT_R state. There is no need to clear the RSI_DATA_CTL.DATEN bit after data transfer. Note that, after a data write, data cannot be written to this register for three SYSCLK periods plus two BCLK periods.

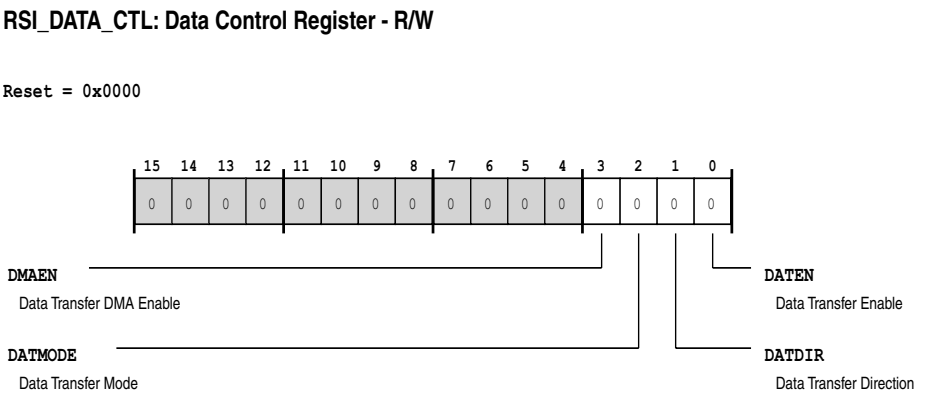


Figure 24-20: RSI_DATA_CTL Register Diagram

Table 24-28: RSI_DATA_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	DMAEN	Data Transfer DMA Enable. The RSI_DATA_CTL.DMAEN bit enables RSI DMA transfers. If not DMA is not enabled, the RSI FIFO is only accessible through the processor core.	
		0	Disable DMA
		1	Enable DMA
2 (R/W)	DATMODE	Data Transfer Mode. The RSI_DATA_CTL.DATMODE bit selects whether the RSI uses stream or block data transfer mode.	
		0	Block Bata Transfer
		1	Stream Data Transfer
1 (R/W)	DATDIR	Data Transfer Direction. The RSI_DATA_CTL.DATDIR bit selects the direction of the transfer.	
		0	Controller-to-Card Transfers
		1	Card-to-Controller Transfers
0 (R/W)	DATEN	Data Transfer Enable. The RSI_DATA_CTL.DATEN bit enables the RSI state machine, enabling RSI operation.	
		0	Disable Data Transfer
		1	Enable Data Transfer

Data Count Register

The RSI_DATA_CNT register is loaded from the RSI_DATA_LEN register when the data path state machine becomes enabled and moves from the IDLE state to the WAIT_S or WAIT_R states. As the data is transferred, the counter decrements; upon decrementing to zero, the state machine then moves back to the IDLE state and the RSI_XFRSTAT.DATEND flag is set.

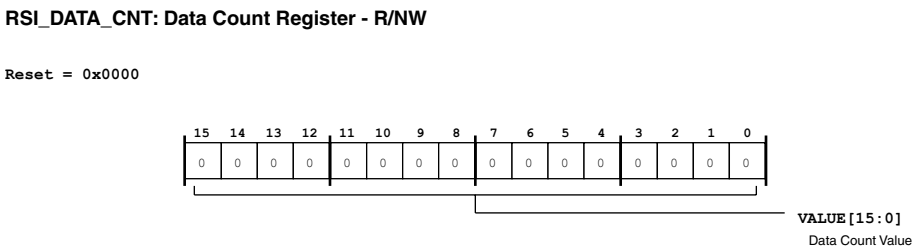


Figure 24-21: RSI_DATA_CNT Register Diagram

Table 24-29: RSI_DATA_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Data Count Value. The RSI_DATA_CNT.VALUE bits contain the current count of bytes remaining in the data transfer.

Transfer Status Register

The RSI_XFRSTAT register contains both static and dynamic flags that indicate the status of the RSI. The static flags (bits [10:0]) remain asserted and are required to be cleared by writing to the RSI_XFRSTAT_CLR register. The dynamic flags (bits [21:11]) change state, depending on the state of the underlying logic. The transmit and receive FIFO logic controls bits [21:12], which vary depending on the state of the FIFO and depending on whether the FIFO is currently enabled for a transmit or receive operation.

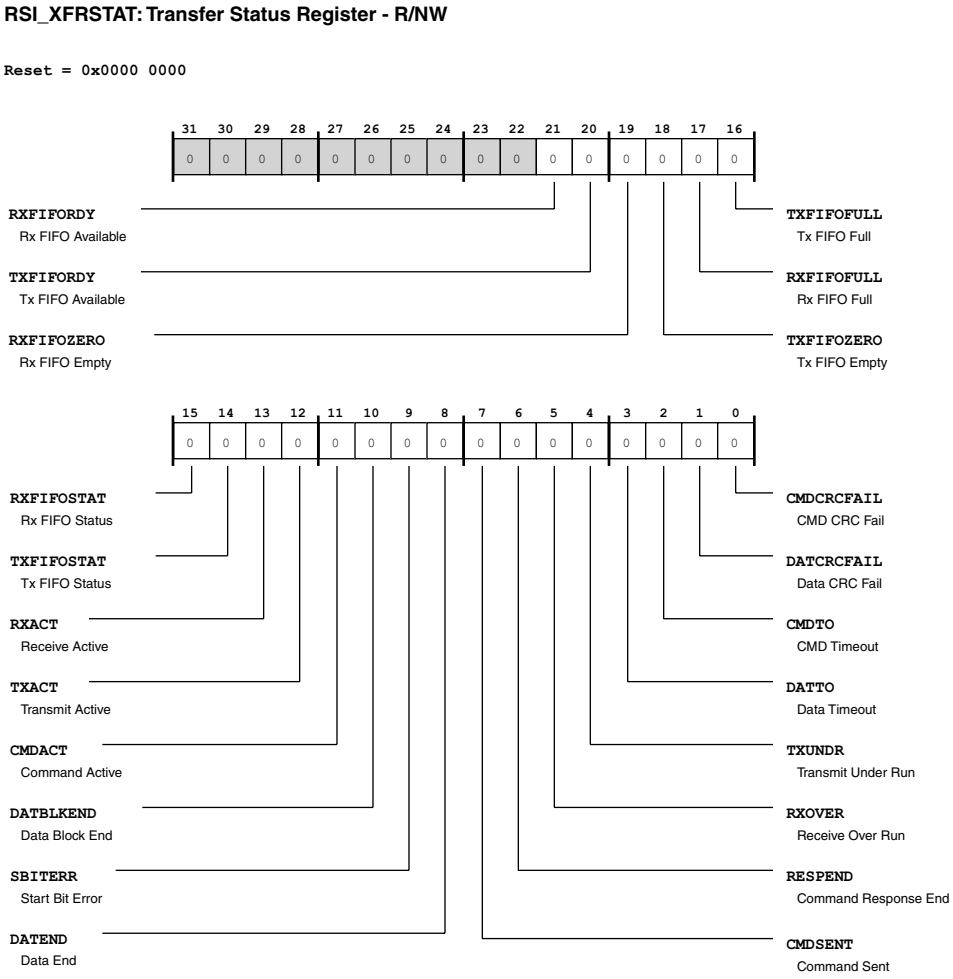


Figure 24-22: RSI_XFRSTAT Register Diagram

Table 24-30: RSI_XFRSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/NW)	RXFIFORDY	Rx FIFO Available.	
		0	No Status
		1	Data Available in Rx FIFO
20 (R/NW)	TXFIFORDY	Tx FIFO Available.	
		0	No Status
		1	Data Available in Tx FIFO
19 (R/NW)	RXFIFOZERO	Rx FIFO Empty.	
		0	No Status
		1	Rx FIFO Empty
18 (R/NW)	TXFIFOZERO	Tx FIFO Empty.	
		0	No Status
		1	Tx FIFO Empty
17 (R/NW)	RXFIFOFULL	Rx FIFO Full.	
		0	No Status
		1	Rx FIFO Full
16 (R/NW)	TXFIFOFULL	Tx FIFO Full.	
		0	No Status
		1	Tx FIFO Full
15 (R/NW)	RXFIFOSTAT	Rx FIFO Status.	
		0	No Status
		1	Rx FIFO Half Full
14 (R/NW)	TXFIFOSTAT	Tx FIFO Status.	
		0	No Status
		1	Tx FIFO Half Empty
13 (R/NW)	RXACT	Receive Active.	
		0	No Status
		1	Data Receive in Progress
12 (R/NW)	TXACT	Transmit Active.	
		0	No Status
		1	Data Transmit in Progress

Table 24-30: RSI_XFRSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/NW)	CMDACT	Command Active.	
		0	No Status
		1	Command Transfer in Progress
10 (R/NW)	DATBLKEND	Data Block End.	
		0	No Status
		1	Data Block Sent/Rcvd (CRC Check Pass)
9 (R/NW)	SBITERR	Start Bit Error.	
		0	No Status
		1	Start Bit Not Detected on All Data Signal in Wide Bus Mode
8 (R/NW)	DATEND	Data End.	
		0	No Status
		1	Data End (Data Counter is Zero)
7 (R/NW)	CMDSENT	Command Sent.	
		0	No Status
		1	Command Sent (No Response Required)
6 (R/NW)	RESPEND	Command Response End.	
		0	No Status
		1	Command Response Received (CRC Check Passed)
5 (R/NW)	RXOVER	Receive Over Run.	
		0	No Status
		1	Rx FIFO Over Run Error
4 (R/NW)	TXUNDR	Transmit Under Run.	
		0	No Status
		1	Tx FIFO Under Run Error
3 (R/NW)	DATTO	Data Timeout.	
		0	No Status
		1	Data Timeout

Table 24-30: RSI_XFRSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/NW)	CMDTO	CMD Timeout.	
		0	No Status
		1	Command Response Timeout
1 (R/NW)	DATCRCFAIL	Data CRC Fail.	
		0	No Status
		1	Data Block Sent/Rcvd (CRC Check Fail)
0 (R/NW)	CMDCRCFAIL	CMD CRC Fail.	
		0	No Status
		1	Command Response Rcvd (CRC Check Fail)

Transfer Status Clear Register

The RSI_XFRSTAT_CLR register is used to clear the static flags of the RSI_XFRSTAT register. Write a 1 to any of the bits to clear the corresponding flag in the RSI_XFRSTAT register.

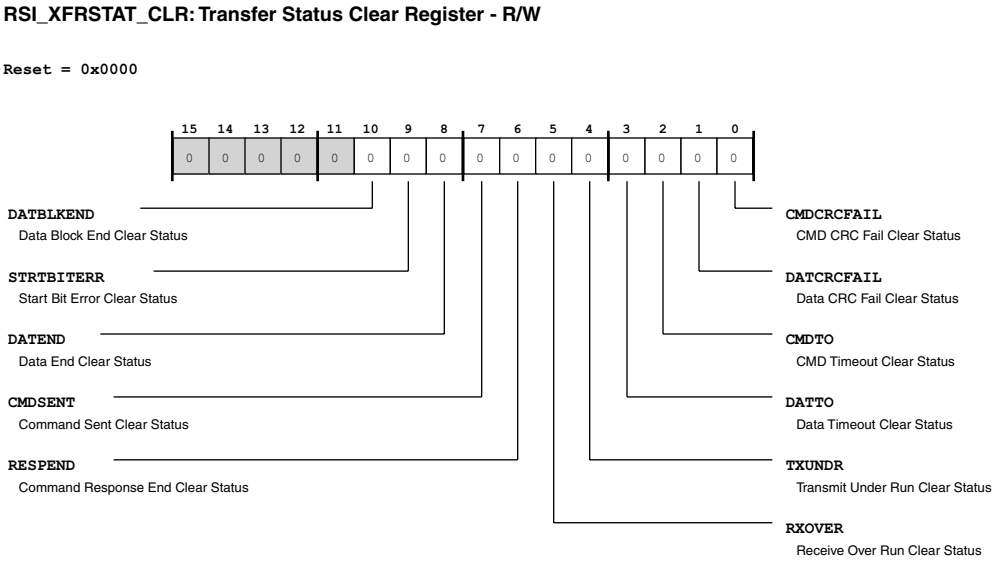


Figure 24-23: RSI_XFRSTAT_CLR Register Diagram

Table 24-31: RSI_XFRSTAT_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/W1C)	DATBLKEND	Data Block End Clear Status. Set RSI_XFRSTAT_CLR.DATBLKEND to clear the corresponding status bit in RSI_XFRSTAT.
9 (R/W1C)	STRTBITERR	Start Bit Error Clear Status. Set RSI_XFRSTAT_CLR.STRTBITERR to clear the corresponding status bit in RSI_XFRSTAT.
8 (R/W1C)	DATEND	Data End Clear Status. Set RSI_XFRSTAT_CLR.DATEND to clear the corresponding status bit in RSI_XFRSTAT.
7 (R/W1C)	CMDSENT	Command Sent Clear Status. Set RSI_XFRSTAT_CLR.CMDSENT to clear the corresponding status bit in RSI_XFRSTAT.
6 (R/W1C)	RESPEND	Command Response End Clear Status. Set RSI_XFRSTAT_CLR.RESPEND to clear the corresponding status bit in RSI_XFRSTAT.
5 (R/W1C)	RXOVER	Receive Over Run Clear Status. Set RSI_XFRSTAT_CLR.RXOVER to clear the corresponding status bit in RSI_XFRSTAT.
4 (R/W1C)	TXUNDR	Transmit Under Run Clear Status. Set RSI_XFRSTAT_CLR.TXUNDR to clear the corresponding status bit in RSI_XFRSTAT.
3 (R/W1C)	DATTO	Data Timeout Clear Status. Set RSI_XFRSTAT_CLR.DATTO to clear the corresponding status bit in RSI_XFRSTAT.
2 (R/W1C)	CMDTO	CMD Timeout Clear Status. Set RSI_XFRSTAT_CLR.CMDTO to clear the corresponding status bit in RSI_XFRSTAT.
1 (R/W1C)	DATCRCFAIL	Data CRC Fail Clear Status. Set RSI_XFRSTAT_CLR.DATCRCFAIL to clear the corresponding status bit in RSI_XFRSTAT.
0 (R/W1C)	CMDCRCFAIL	CMD CRC Fail Clear Status. Set RSI_XFRSTAT_CLR.CMDCRCFAIL to clear the corresponding status bit in RSI_XFRSTAT.

Transfer Interrupt 0 Mask Register

The interrupt mask registers (RSI_XFR_IMSK0 and RSI_XFR_IMSK1) determine which of the static and dynamic flags of the RSI_XFRSTAT register generate an interrupt request using one of the RSI interrupts. An RSI interrupt is enabled by setting the corresponding bit in the interrupt mask register to 1. Interrupts enabled in the RSI_XFR_IMSK0 register generate an interrupt using the RSI_INT0 signal of the RSI, and interrupts enabled in the RSI_XFR_IMSK1 register generate an interrupt using the RSI_INT1 signal of the RSI.

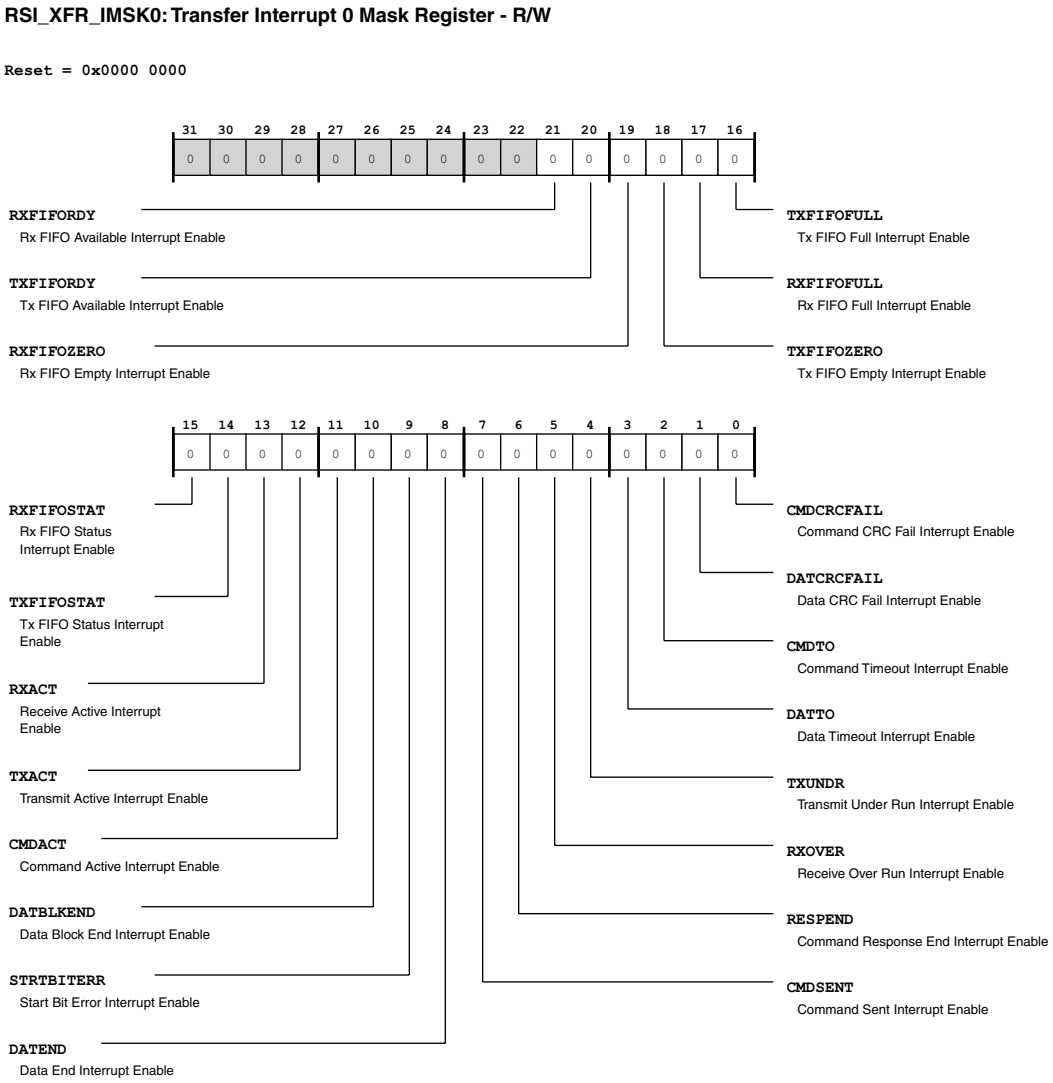


Figure 24-24: RSI_XFR_IMSK0 Register Diagram

Table 24-32: RSI_XFR_IMSK0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	RXFIFORDY	Rx FIFO Available Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
20 (R/W)	TXFIFORDY	Tx FIFO Available Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
19 (R/W)	RXFIFOZERO	Rx FIFO Empty Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
18 (R/W)	TXFIFOZERO	Tx FIFO Empty Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
17 (R/W)	RXFIFOFULL	Rx FIFO Full Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
16 (R/W)	TXFIFOFULL	Tx FIFO Full Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
15 (R/W)	RXFIFOSTAT	Rx FIFO Status Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
14 (R/W)	TXFIFOSTAT	Tx FIFO Status Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
13 (R/W)	RXACT	Receive Active Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
12 (R/W)	TXACT	Transmit Active Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Table 24-32: RSI_XFR_IMSK0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	CMDACT	Command Active Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
10 (R/W)	DATBLKEND	Data Block End Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
9 (R/W)	STRTBITERR	Start Bit Error Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
8 (R/W)	DATEND	Data End Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
7 (R/W)	CMDSENT	Command Sent Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
6 (R/W)	RESPEND	Command Response End Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
5 (R/W)	RXOVER	Receive Over Run Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
4 (R/W)	TXUNDR	Transmit Under Run Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
3 (R/W)	DATTO	Data Timeout Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
2 (R/W)	CMDTO	Command Timeout Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Table 24-32: RSI_XFR_IMSK0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	DATCRCFAIL	Data CRC Fail Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
0 (R/W)	CMDCRCFAIL	Command CRC Fail Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Transfer Interrupt 1 Mask Register

The interrupt mask registers (RSI_XFR_IMSK0 and RSI_XFR_IMSK1) determine which of the static and dynamic flags of the RSI_XFRSTAT register generate an interrupt request using one of the RSI interrupts. An RSI interrupt is enabled by setting the corresponding bit in the interrupt mask register to 1. Interrupts enabled in the RSI_XFR_IMSK0 register generate an interrupt using the RSI_INT0 signal of the RSI, and interrupts enabled in the RSI_XFR_IMSK1 register generate an interrupt using the RSI_INT1 signal of the RSI.

RSI_XFR_IMSK1: Transfer Interrupt 1 Mask Register - R/W

Reset = 0x0000 0000

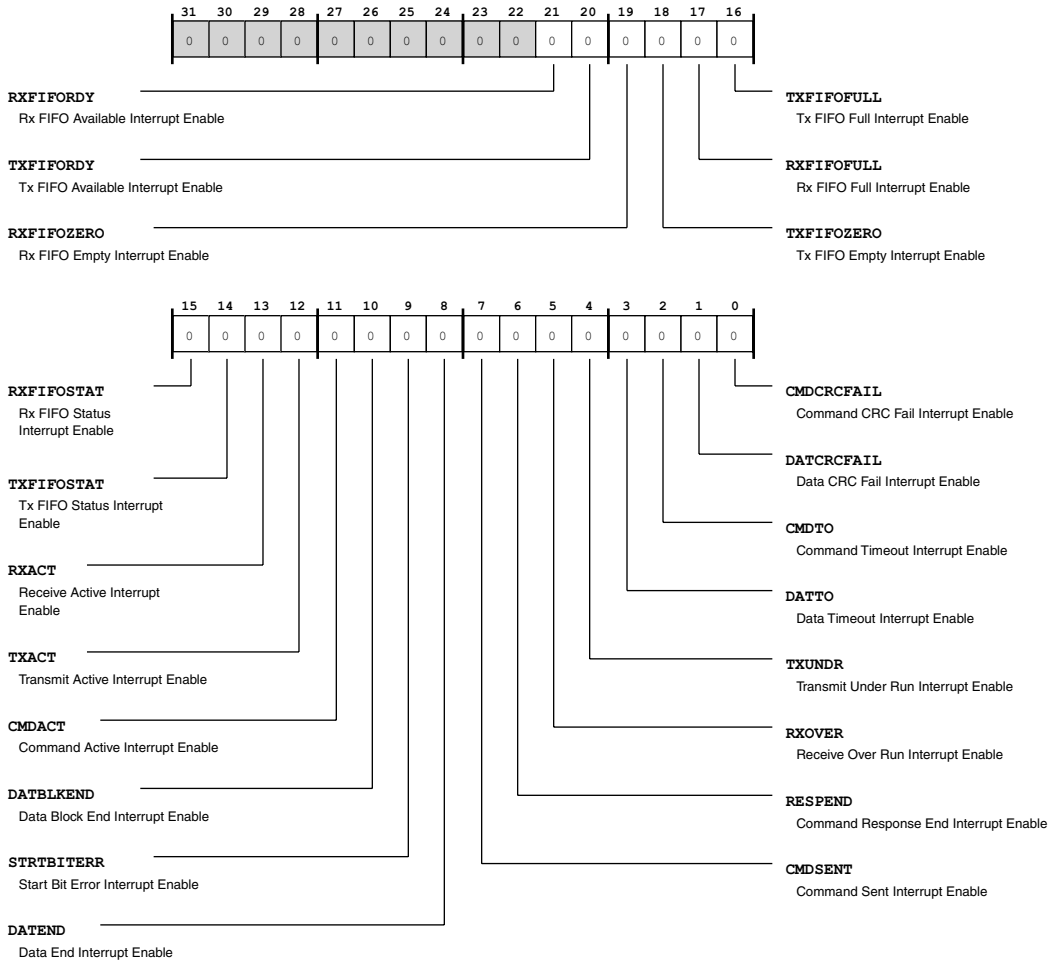


Figure 24-25: RSI_XFR_IMSK1 Register Diagram

Table 24-33: RSI_XFR_IMSK1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	RXFIFORDY	Rx FIFO Available Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
20 (R/W)	TXFIFORDY	Tx FIFO Available Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Table 24-33: RSI_XFR_IMSK1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	RXFIFOZERO	Rx FIFO Empty Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
18 (R/W)	TXFIFOZERO	Tx FIFO Empty Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
17 (R/W)	RXFIFOFULL	Rx FIFO Full Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
16 (R/W)	TXFIFOFULL	Tx FIFO Full Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
15 (R/W)	RXFIFOSTAT	Rx FIFO Status Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
14 (R/W)	TXFIFOSTAT	Tx FIFO Status Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
13 (R/W)	RXACT	Receive Active Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
12 (R/W)	TXACT	Transmit Active Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
11 (R/W)	CMDACT	Command Active Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
10 (R/W)	DATBLKEND	Data Block End Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Table 24-33: RSI_XFR_IMSK1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	STRTBITERR	Start Bit Error Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
8 (R/W)	DATEND	Data End Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
7 (R/W)	CMDSENT	Command Sent Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
6 (R/W)	RESPEND	Command Response End Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
5 (R/W)	RXOVER	Receive Over Run Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
4 (R/W)	TXUNDR	Transmit Under Run Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
3 (R/W)	DATTO	Data Timeout Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
2 (R/W)	CMDTO	Command Timeout Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
1 (R/W)	DATCRCFAIL	Data CRC Fail Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
0 (R/W)	CMDCRCFAIL	Command CRC Fail Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

FIFO Counter Register

The RSI_FIFO_CNT register contains a value indicating the number of 32-bit words still to be read from or written to the FIFO. The RSI_FIFO_CNT is loaded from the RSI_DATA_LEN register when the RSI_DATA_CTL.DATEN bit is set. If the data length is not word-aligned (multiple of 4), the remaining 1 to 3 bytes are regarded as a word.

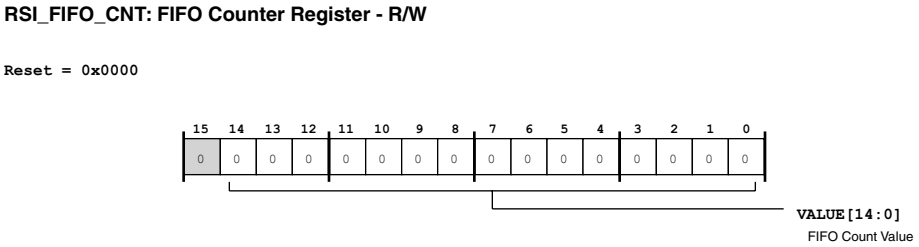


Figure 24-26: RSI_FIFO_CNT Register Diagram

Table 24-34: RSI_FIFO_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/W)	VALUE	FIFO Count Value. The RSI_FIFO_CNT.VALUE bits contain the current count of 32-bit words remaining to be read-from or written-to the FIFO.

Boot Timing Counter Register

The RSI_BOOT_TCNTR register selects the cycle count for normal and alternate boot signal timing. When the RSI_BOOT_TCNTR.HOLD and RSI_BOOT_TCNTR.SETUP counters expire, the RSI sets the RSI_STAT0.BSETUPEXP and RSI_STAT0.BHOLDEXP bits, raising the corresponding interrupts (if enabled/unmasked) in the RSI_IMSK0 register.

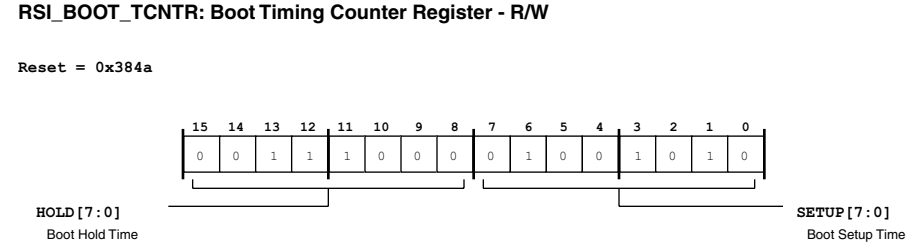


Figure 24-27: RSI_BOOT_TCNTR Register Diagram

Table 24-35: RSI_BOOT_TCNTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	HOLD	Boot Hold Time. In Normal Boot mode, The RSI_BOOT_TCNTR.HOLD bits select the minimum number of cycles (RSI_CLK periods) required after pulling RSI_CMD line high to resume normal operation. In Alternate Boot mode, the RSI_BOOT_TCNTR.HOLD bits select the minimum number of cycles (RSI_CLK periods) required after CMD0/Reset to resume normal operation.
7:0 (R/W)	SETUP	Boot Setup Time. In Normal Boot mode, the RSI_BOOT_TCNTR.SETUP bits select the minimum number of cycles (RSI_CLK periods) for which RSI_CMD line pulled low to enable boot. In Alternate Boot mode, the RSI_BOOT_TCNTR.SETUP bits select the minimum number of cycles (RSI_CLK periods) for which RSI_CMD line is high before giving CMD0 with 0xFFFFFFFA.

Boot Acknowledge Timeout Register

When the RSI_CFG.BACKEN bit =1, the value in the RSI_BACK_TOUT register is used to compute the boot acknowledge timeout interval for the ACK from a slave. The RSI increments a counter (starts at zero) from the RSI_CLK while waiting for the acknowledge. When this counter reaches the timeout value, the RSI sets the RSI_STAT0.BACKTO bit, raising the corresponding interrupts (if enabled/unmasked) in the RSI_IMSK0 register. When the RSI_CFG.BACKEN bit =0, the RSI_BACK_TOUT register contents are not used.

The value to be programmed in the RSI_BACK_TOUT register is computed with the following formula,

Boot Acknowledge timeout = (Absolute time in seconds) * (RSI_CLK frequency)

= ((Absolute time) * SCLK)/ (2(RSI_CTL.CLKDIV + 1))

Where, RSI_CTL.CLKDIV holds the value, by which the SCLK is divided to generate RSI_CLK (RSI clock).

In bypass mode (clock divisor disabled), the formula is:

Boot Acknowledge timeout = (Absolute time in seconds) * SCLK

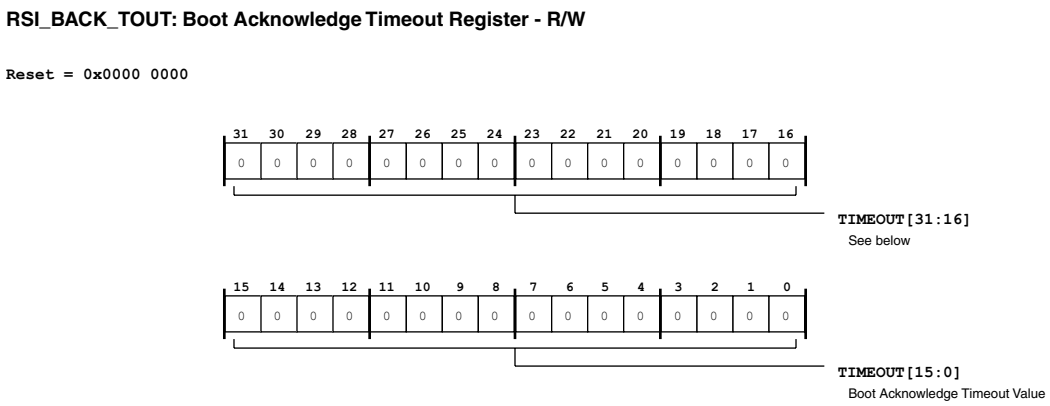


Figure 24-28: RSI_BACK_TOUT Register Diagram

Table 24-36: RSI_BACK_TOUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TIMEOUT	Boot Acknowledge Timeout Value. The RSI_BACK_TOUT.TIMEOUT bits hold the value used to compute the boot acknowledge timeout interval for the ACK from a slave. This timeout only is used when the RSI_CFG.BACKEN bit =1.

Sleep Wakeup Timeout Register

The RSI_SLP_WKUP_TOUT register defines the maximum timeout value for state transitions from Standby State to Sleep State and from Sleep State to Standby State. This value is loaded to Sleep/Wakeup Counter when a Sleep or Wakeup command is sent. When the counter expires, the RSI sets the RSI_STAT0.SLPWKPTOUT bit, raising the corresponding interrupts (if enabled/unmasked) in the RSI_IMSK0 register. The formula to calculate the maximum timeout value is:

Sleep Wakeup Timeout = (Absolute time in seconds) * (RSI_CLK frequency)

= ((Absolute time in seconds) * SCLK)/(2(RSI_CTL.CLKDIV + 1))

Where RSI_CTL.CLKDIV holds the value by which the SCLK is divided to generate the RSI_CLK (RSI clock).

In bypass mode (clock divisor disabled), the formula is:

Sleep Wakeup Timeout = (Absolute time in seconds) * SCLK

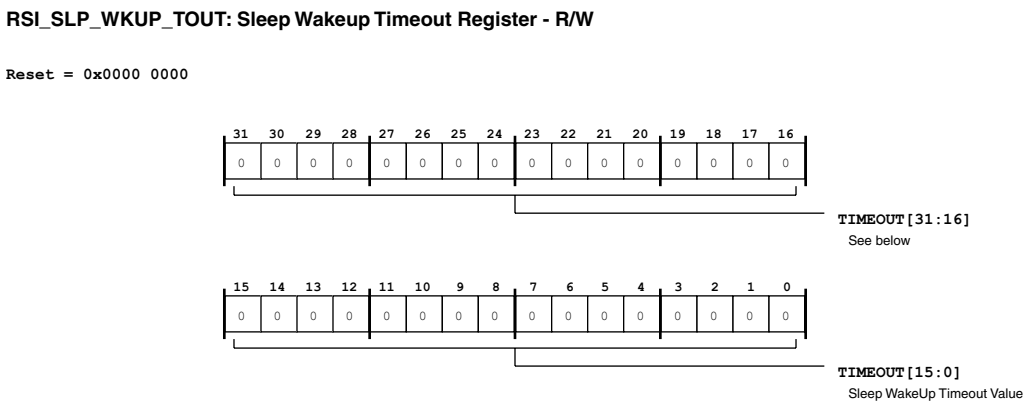


Figure 24-29: RSI_SLP_WKUP_TOUT Register Diagram

Table 24-37: RSI_SLP_WKUP_TOUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TIMEOUT	Sleep WakeUp Timeout Value. The RSI_SLP_WKUP_TOUT.TIMEOUT bits defines the maximum timeout value for state transitions from Standby State to Sleep State and from Sleep State to Standby State.

Block Size Register

The RSI_BLKSZ register holds the size of each block in bytes. Only allowed block size values should be programmed.

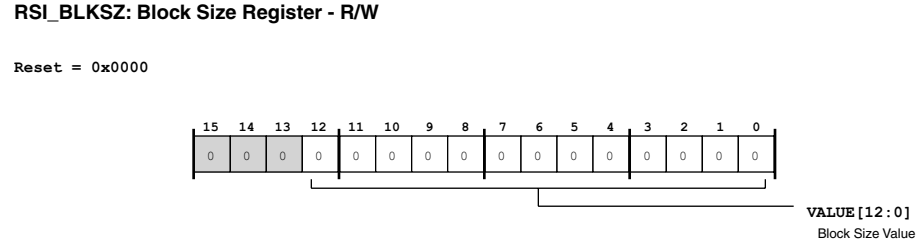


Figure 24-30: RSI_BLKSZ Register Diagram

Table 24-38: RSI_BLKSZ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
12:0 (R/W)	VALUE	Block Size Value. The RSI_BLKSZ.VALUE bits select the size of each block of data.	
		0x801 - 0x1FFF	Reserved
		0x003 - 0x7FF	3 to 2047 Bytes
		1	1 Byte
		2	2 Bytes
		2048	2048 Bytes

Data FIFO Register

The RSI_FIFO register provides access to the 16-entry transmit and receive FIFO. The register is accessed as a 32-bit word.

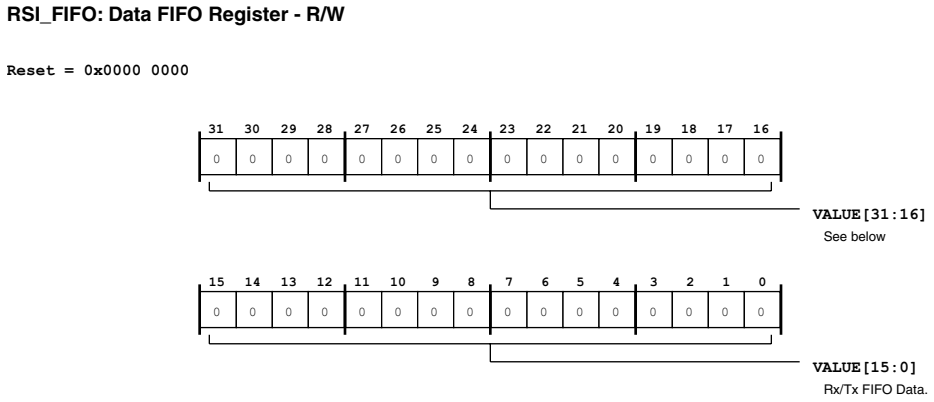


Figure 24-31: RSI_FIFO Register Diagram

Table 24-39: RSI_FIFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Rx/Tx FIFO Data.. The RSI_FIFO.VALUE bits hold FIFO data.

Exception Status Register

The RSI_STAT0 register contains exception status bits for SDIO cards and the card detection logic. This register also contains exception status bits for Boot mode, Sleep mode and Card Busy status. These excep-

tion status bits can be uses to generate an interrupt request through the RSI_INT0 signal by enabling (unmasking) the corresponding exception interrupt in the RSI_IMSK0 register. All bits in this register are write-1-to-clear bits.

RSI_STAT0: Exception Status Register - R/W

Reset = 0x0000 0000

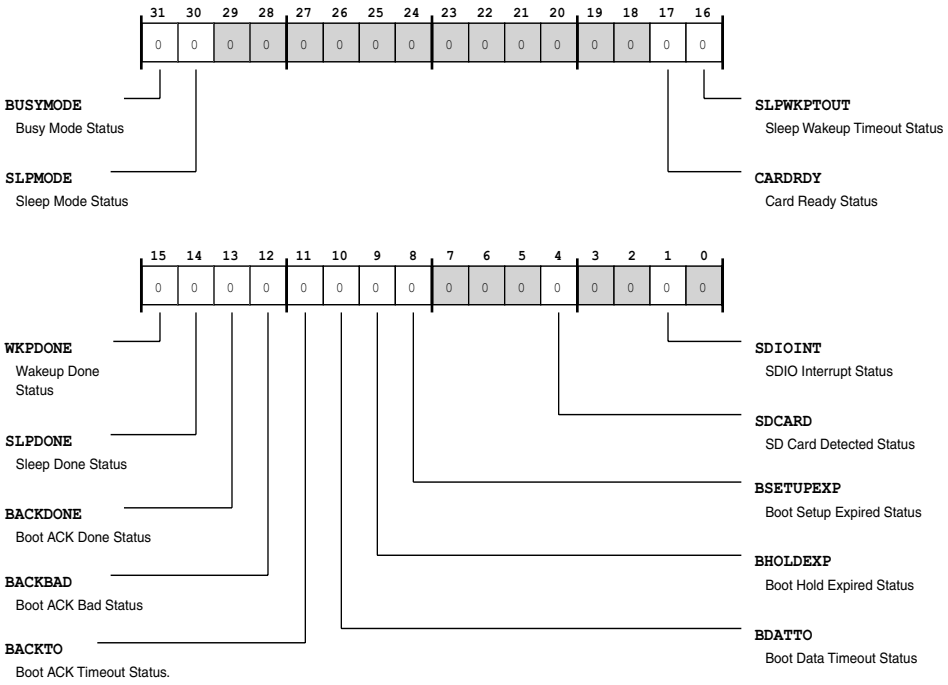


Figure 24-32: RSI_STAT0 Register Diagram

Table 24-40: RSI_STAT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	BUSYMODE	Busy Mode Status. The RSI_STAT0.BUSYMODE bit indicates when the card is in Busy mode.	
		0	No Status
		1	Busy Mode
30 (R/W)	SLPMODE	Sleep Mode Status. The RSI_STAT0.SLPMODE bit indicates when the card is in Sleep Mode.	
		0	No Status
		1	Sleep Mode

Table 24-40: RSI_STAT0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	CARDRDY	Card Ready Status. The RSI_STAT0.CARDRDY bit indicates when the RSI_CMD.CHKBUSY is set and the card comes out of busy state by pulling the RSI_D0 line high.	
		0	No Status
		1	Card Ready
16 (R/W)	SLPWKPTOUT	Sleep Wakeup Timeout Status. The RSI_STAT0.SLPWKPTOUT bit indicates when the card does not change state between Standby and Sleep State before the timeout counter expires.	
		0	No Status
		1	Timeout
15 (R/W)	WKPDONE	Wakeup Done Status. The RSI_STAT0.WKPDONE bit indicates when the card successfully entered Standby State from Sleep State.	
		0	No Status
		1	Sleep-to-Standby State Transition Success
14 (R/W)	SLPDONE	Sleep Done Status. The RSI_STAT0.SLPDONE bit indicates when the card successfully entered Sleep State from Standby State.	
		0	No Status
		1	Standby-to-Sleep State Transition Success
13 (R/W)	BACKDONE	Boot ACK Done Status. The RSI_STAT0.BACKDONE bit indicates when the expected boot ACK is received.	
		0	No Status
		1	Correct Boot ACK Received
12 (R/W)	BACKBAD	Boot ACK Bad Status. The RSI_STAT0.BACKBAD bit indicates when the received boot ACK is not =010 (bad).	
		0	No Status
		1	Corrupted Boot ACK Received

Table 24-40: RSI_STAT0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	BACKTO	Boot ACK Timeout Status.. The RSI_STAT0.BACKTO bit indicates when boot ACK is not received before the boot ACK timeout counter expires. This status is only available when the RSI_CFG.BACKEN bit is set.	
		0	No Status
		1	Timeout
10 (R/W)	BDATTO	Boot Data Timeout Status. The RSI_STAT0.BDATTO bit indicates when boot data is not received before the boot data timeout counter expires.	
		0	No Status
		1	Timeout
9 (R/W)	BHOLDEXP	Boot Hold Expired Status. The RSI_STAT0.BHOLDEXP bit indicates when the boot hold time counter expires.	
		0	No Status
		1	Counter Expired
8 (R/W)	BSETUPEXP	Boot Setup Expired Status. The RSI_STAT0.BSETUPEXP bit indicates when the boot setup time counter expires.	
		0	No Status
		1	Counter Expired
4 (R/W)	SDCARD	SD Card Detected Status. The RSI_STAT0.SDCARD bit indicates that the RSI has detected a rising edge on the RSI_D3 signal, which is intended for use with SD devices that support card detection using this signal.	
		0	No SD Card Detected
		1	SD Card Detected
1 (R/W)	SDIOINT	SDIO Interrupt Status. The RSI_STAT0.SDIOINT bit indicates that the RSI has detected an interrupt generated by SDIO cards on the RSI_D0 signal.	
		0	No interrupt detected
		1	Interrupt detected

Exception Mask Register

The interrupt mask bits in the RSI_IMSK0 register determine which of the flags of the RSI_STAT0 register generate an interrupt request using the RSI_INT0 interrupt. The RSI_IMSK0 register contains mask bits for the RSI_STAT0 status bits. Writing a "1" to the RSI_IMSK0 bit enables the interrupt for the corresponding bit in the RSI_STAT0 register.

RSI_IMSK0: Exception Mask Register - R/W

Reset = 0x0000 0000

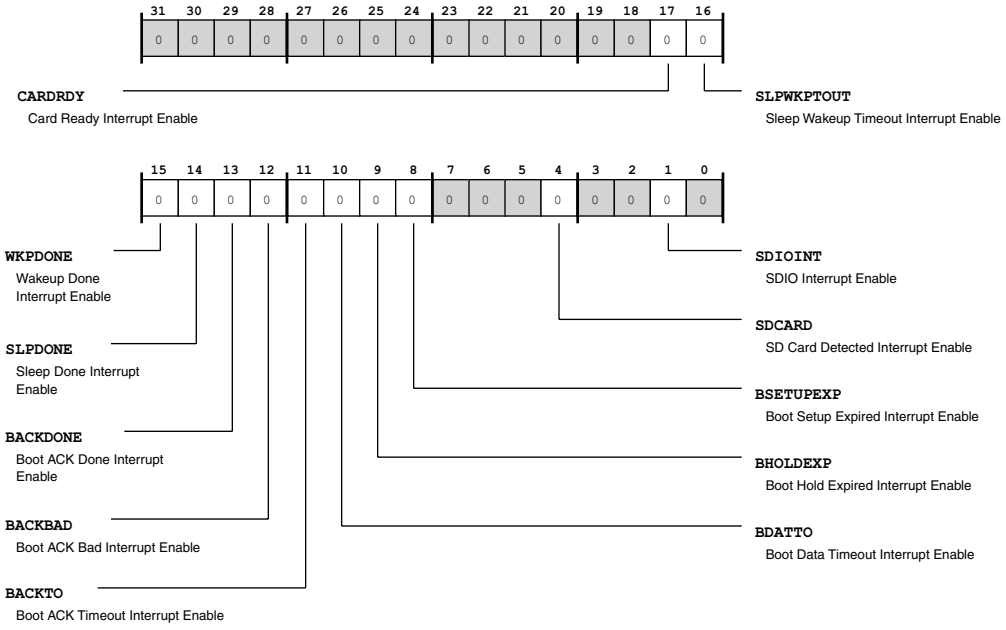


Figure 24-33: RSI_IMSK0 Register Diagram

Table 24-41: RSI_IMSK0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	CARDRDY	Card Ready Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
16 (R/W)	SLPWKPTOUT	Sleep Wakeup Timeout Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
15 (R/W)	WKPDONE	Wakeup Done Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Table 24-41: RSI_IMSK0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	SLPDONE	Sleep Done Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
13 (R/W)	BACKDONE	Boot ACK Done Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
12 (R/W)	BACKBAD	Boot ACK Bad Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
11 (R/W)	BACKTO	Boot ACK Timeout Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
10 (R/W)	BDATTO	Boot Data Timeout Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
9 (R/W)	BHOLDEXP	Boot Hold Expired Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
8 (R/W)	BSETUPEXP	Boot Setup Expired Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
4 (R/W)	SDCARD	SD Card Detected Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
1 (R/W)	SDIOINT	SDIO Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Configuration Register

The RSI_CFG register controls bits that enable and disable portions of the RSI. The RSI_CFG.CLKSEN bit must be set in order to enable the RSI for operation. If an external pull-down resistor is used for implementing card detection on the RSI_D3 signal, the RSI_CFG.DAT3PUP should not be set. The pull-up and pull-down resistors on the RSI_D0 through RSI_D7 signals (but not RSI_D3) become active only when the corresponding GPIO pins are configured for RSI functionality with the pin multiplexing. For example, if only the 4-bit data bus is enabled in the pin multiplexing, setting RSI_CFG.DATPUP enables only the pull-up resistors on the signals that are configured for RSI use. The RSI_CFG register also provides additional functionality for SDIO support. To enable SDIO 4-bit mode, in addition to setting the bus width to 4-bit with the RSI_CTL.BUSWID field, the RSI_CFG.SD4EN should be set. The RSI_CFG.MWINEN bit, when set, allows for SDIO interrupts to be detected outside the specified one-cycle window and is set when interrupt support is required during multiple block read transactions from SDIO. The RSI can also be reset with the RSI_CFG.RST bit. Writing this bit resets the RSI module and returns all registers to their default values.

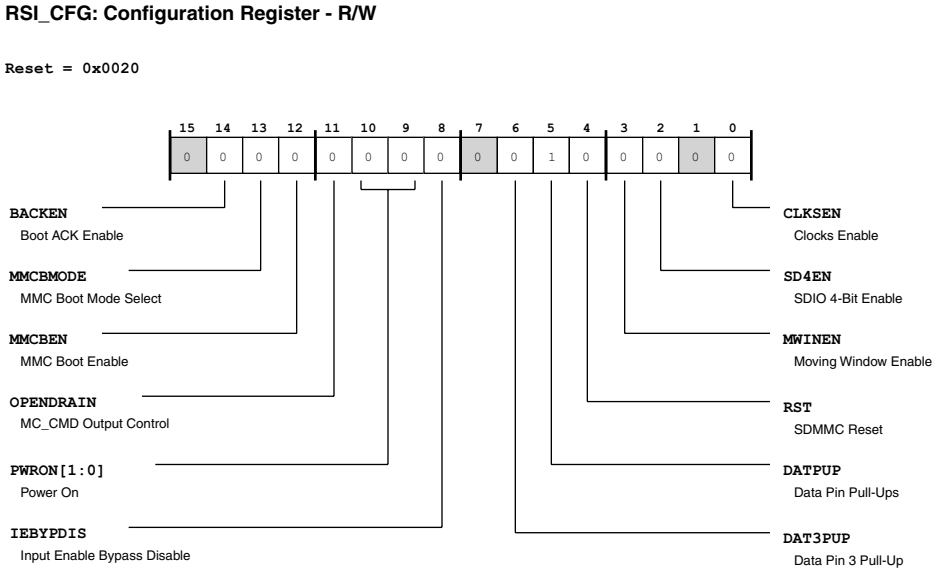


Figure 24-34: RSI_CFG Register Diagram

Table 24-42: RSI_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	BACKEN	Boot ACK Enable. The RSI_CFG.BACKEN bit directs the RSI to expect a boot ACK.	
		0	No Boot ACK Expected
		1	Expect Boot ACK

Table 24-42: RSI_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	MMCBMODE	MMC Boot Mode Select. The RSI_CFG.MMCBMODE bit selects the MCC boot mode.	
		0	Normal Boot Mode
		1	Alternate Boot Mode
12 (R/W)	MMCBEN	MMC Boot Enable. The RSI_CFG.MMCBEN bit enables MCC boot operation.	
		0	Disable MMC Boot
		1	Enable MMC Boot
11 (R/W)	OPENDRAIN	MC_CMD Output Control. The RSI_CFG.OPENDRAIN bit selects whether the MC_CMD output control is an open drain output.	
		0	Not Open Drain
		1	Open Drain
10:9 (R/W)	PWRON	Power On. The RSI_CFG.PWRON bits enable RSI operation. Field values other than those shown are reserved.	
		0	Disable RSI
		3	Enable RSI
8 (R/W)	IEBYPDIS	Input Enable Bypass Disable. The RSI_CFG.IEBYPDIS bit disables the input enable bypass.	
		0	Enable Bypass
		1	Disable Bypass
6 (R/W)	DAT3PUP	Data Pin 3 Pull-Up. The RSI_CFG.DAT3PUP bit enables a pull-up resistor on the SD_ DAT3 line (RSI_D3).	
		0	Disable Pull-Up
		1	Enable Pull-Up
5 (R/W)	DATPUP	Data Pin Pull-Ups. The RSI_CFG.DATPUP bit enables pull-up resistors on the SD_DAT lines (RSI_D0 through RSI_D7 signals---but not RSI_D3---depending on RSI_CTL.BUSWID selection).	
		0	Disable Pull-Ups
		1	Enable Pull-Ups

Table 24-42: RSI_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W1A)	RST	SDMMC Reset. The RSI_CFG.RST bit resets the SDMMC.	
		0	No Action
		1	Reset SDMMC
3 (R/W)	MWINEN	Moving Window Enable. The RSI_CFG.MWINEN bit enables moving window operation.	
		0	Disable Moving Window
		1	Enable Moving Window
2 (R/W)	SD4EN	SDIO 4-Bit Enable. The RSI_CFG.SD4EN bit enable SDIO 4-bit operation.	
		0	Disable SDIO 4-bit
		1	Enable SDIO 4-bit
0 (R/W)	CLKSEN	Clocks Enable. The RSI_CFG.CLKSEN bit enable RSI clocks (both PCLK and MCLK).	
		0	Disable Clocks
		1	Enable Clocks

Read Wait Enable Register

The RSI_RD_WAIT register contains the RSI_RD_WAIT.REQUEST bit that, when set, issues a read wait request to an SDIO card. After software is ready to resume the data transfer, this bit must be cleared. The functionality applies to both 1-bit and 4-bit SDIO modes.

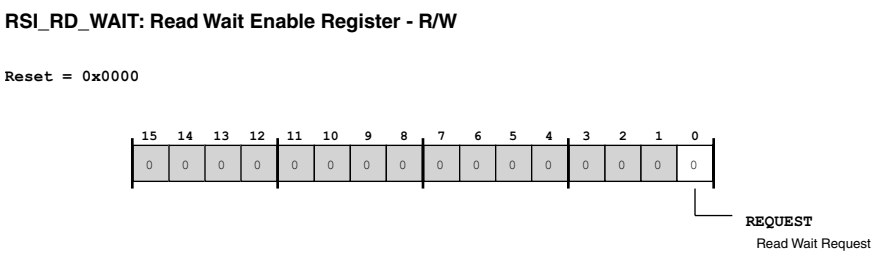


Figure 24-35: RSI_RD_WAIT Register Diagram

Table 24-43: RSI_RD_WAIT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	REQUEST	Read Wait Request. The RSI_RD_WAIT.REQUEST bit directs the RSI to issue a read wait request to the SDIO card.	
		0	Normal Operation
		1	Read Wait Request Operation

Peripheral ID 0 Register

The RSI_PIDx registers (RSI_PID0, RSI_PID1, RSI_PID2, RSI_PID3, RSI_PID4, RSI_PID5, RSI_PID6, and RSI_PID7) contain a fixed value at reset and are used to identify the peripheral revision. There are a total of four 16-bit identification registers of which the lower 8-bits are valid. The contents of these four registers are:

RSI Peripheral ID Register	RSI_PID Value
RSI_PID0	0x80
RSI_PID1	0x11
RSI_PID2	0x04
RSI_PID3	0x00

The RSI identification registers are read-only. The values of the bits can be grouped into one 32-bit word--the word comprehends RSIPID [3:0] of value 0x00041180.

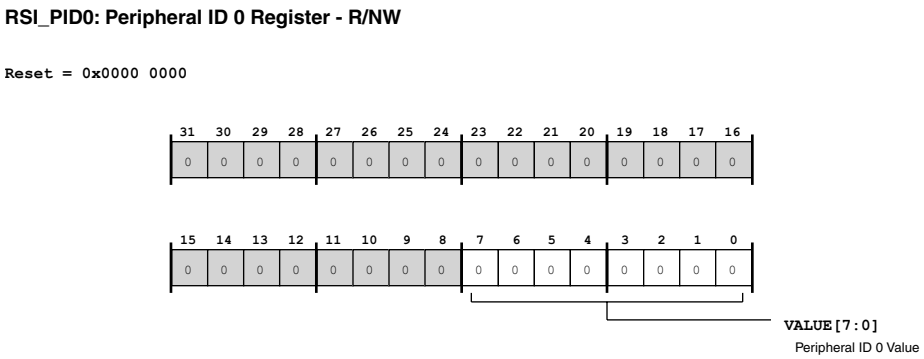


Figure 24-36: RSI_PID0 Register Diagram

Table 24-44: RSI_PID0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	VALUE	Peripheral ID 0 Value. For more information, see the RSI_PID0 register description.

Peripheral ID 1 Register

There are a total of four 16-bit identification registers of which the lower 8-bits are valid. For more information, see the RSI_PID0 register description.

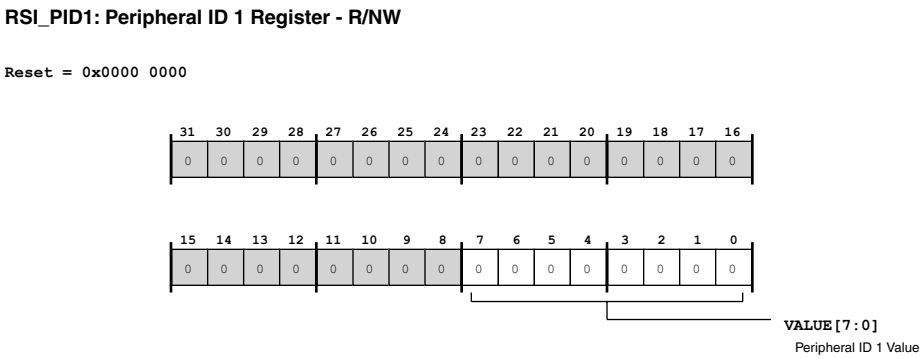


Figure 24-37: RSI_PID1 Register Diagram

Table 24-45: RSI_PID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	VALUE	Peripheral ID 1 Value. For more information, see the RSI_PID0 register description.

Peripheral ID 2 Register

There are a total of four 16-bit identification registers of which the lower 8-bits are valid. For more information, see the RSI_PID0 register description.

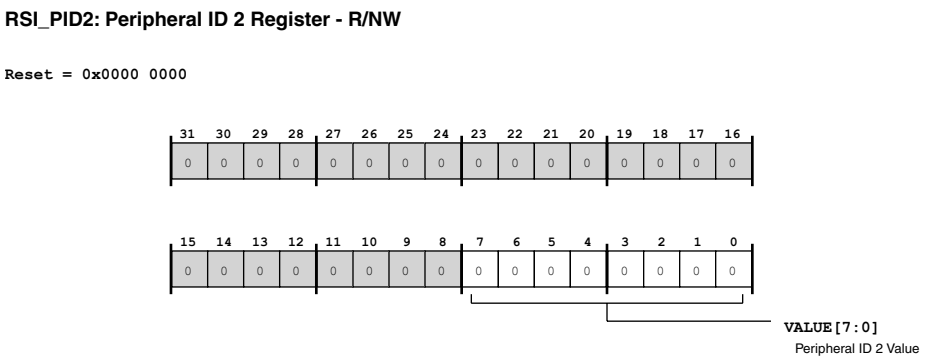


Figure 24-38: RSI_PID2 Register Diagram

Table 24-46: RSI_PID2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	VALUE	Peripheral ID 2 Value. For more information, see the RSI_PID0 register description.

Peripheral ID 3 Register

There are a total of four 16-bit identification registers of which the lower 8-bits are valid. For more information, see the RSI_PID0 register description.

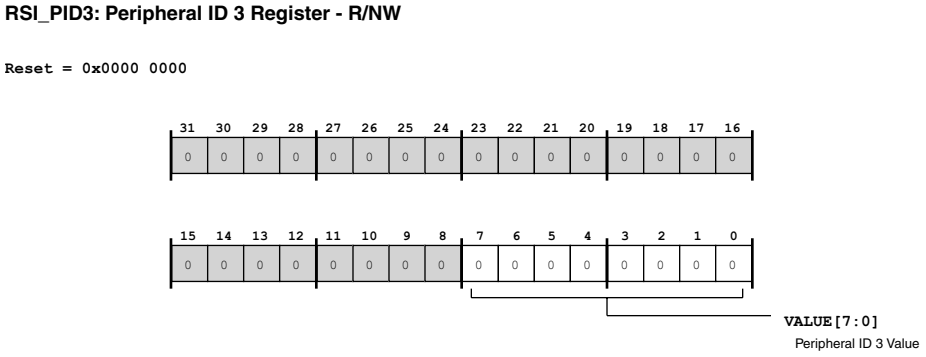


Figure 24-39: RSI_PID3 Register Diagram

Table 24-47: RSI_PID3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	VALUE	Peripheral ID 3 Value. For more information, see the RSI_PID0 register description.

25 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. With the two data pins, it allows for full-duplex operation to other SPI compatible devices. An additional two (optional) data pins are provided to support quad SPI operation. Enhanced modes of operation such as flow control, Fast Mode and dual-I/O mode (DIOM) are also supported. Moreover, a direct memory access (DMA) mode allows for transferring several words with minimal CPU interaction.

The SPI interface includes programmable baud rates, clock phase, and clock polarity. It can operate in a multi-master environment by interfacing with several other devices, acting as either a master device or a slave device. In a multi-master environment, the SPI interface uses open drain outputs to avoid data bus contention.

SPI Features

The SPI module supports the following features:

- Full-duplex, synchronous serial interface
- Supports 8, 16 and 32-bit word sizes
- Programmable baud rate, clock phase and polarity
- Programmable inter-frame latency
- Flow control
- Support for Fast, DIOM and Quad SPI enhanced modes
- Independent receive and transmit DMA channels
- Burst transfer mode for non-DMA write accesses

SPI Functional Description

The following sections provide functional descriptions of the SPI:

- *ADSP-BF60x SPI Register List*
- *ADSP-BF60x SPI Interrupt List*

- *ADSP-BF60x SPI Trigger List*
- *SPI Block Diagram*

The SPI is essentially a shift register that serially transmits and receives data bits to/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.

During a data transfer, one SPI system acts as the link master which controls the data flow, while 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 may simultaneously shift data into multiple slaves (broadcast mode). However, only one slave may drive its output to write data back to the master at any given time. This must be enforced in the broadcast mode, where several slaves can be selected to receive data from the master, but only one slave can be enabled to send data back to the master.

SPI supports enhanced modes of operation like Fast Mode, DIOM and Quad-SPI, as well as providing 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 two 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 four bits are shifted in or out on each receive or transmit edge. Flow control can be used by a slower slave to stall a faster master device.

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 may 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 slaves must be in transmit mode and driving the `SPI_MISO`line.

ADSP-BF60x 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 govern SPI operations. For more information on SPI functionality, see the SPI register descriptions.

Table 25-1: ADSP-BF60x SPI Register List

Name	Description
`SPI_CTL`	Control Register
`SPI_RXCTL`	Receive Control Register
`SPI_TXCTL`	Transmit Control Register

Table 25-1: ADSP-BF60x SPI Register List (Continued)

Name	Description
SPI_CLK	Clock Rate Register
SPI_DLY	Delay Register
SPI_SLVSEL	Slave Select Register
SPI_RWC	Received Word Count Register
SPI_RWCR	Received Word Count Reload Register
SPI_TWC	Transmitted Word Count Register
SPI_TWCR	Transmitted Word Count Reload Register
SPI_IMSK	Interrupt Mask Register
SPI_IMSK_CLR	Interrupt Mask Clear Register
SPI_IMSK_SET	Interrupt Mask Set Register
SPI_STAT	Status Register
SPI_ILAT	Masked Interrupt Condition Register
SPI_ILAT_CLR	Masked Interrupt Clear Register
SPI_RFIFO	Receive FIFO Data Register
SPI_TFIFO	Transmit FIFO Data Register

ADSP-BF60x SPI Interrupt List

Table 25-2: ADSP-BF60x SPI Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
SPI0 TX DMA Channel	55	6	LEVEL
SPI0 RX DMA Channel	56	7	LEVEL
SPI0 Status	57		LEVEL
SPI1 TX DMA Channel	58	8	LEVEL
SPI1 RX DMA Channel	59	9	LEVEL

Table 25-2: ADSP-BF60x SPI Interrupt List Interrupt List (Continued)

Description	Interrupt ID	DMA Channel	Sensitivity
SPI1 Status	60		LEVEL

ADSP-BF60x SPI Trigger List

Table 25-3: ADSP-BF60x SPI Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
SPI0 TX DMA Channel	26	PULSE/EDGE
SPI0 RX DMA Channel	27	PULSE/EDGE
SPI1 TX DMA Channel	28	PULSE/EDGE
SPI1 RX DMA Channel	29	PULSE/EDGE

Table 25-4: ADSP-BF60x SPI Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
SPI0 TX DMA Channel	26	
SPI0 RX DMA Channel	27	
SPI1 TX DMA Channel	28	
SPI1 RX DMA Channel	29	

SPI Block Diagram

The figure below illustrates the block diagram 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.

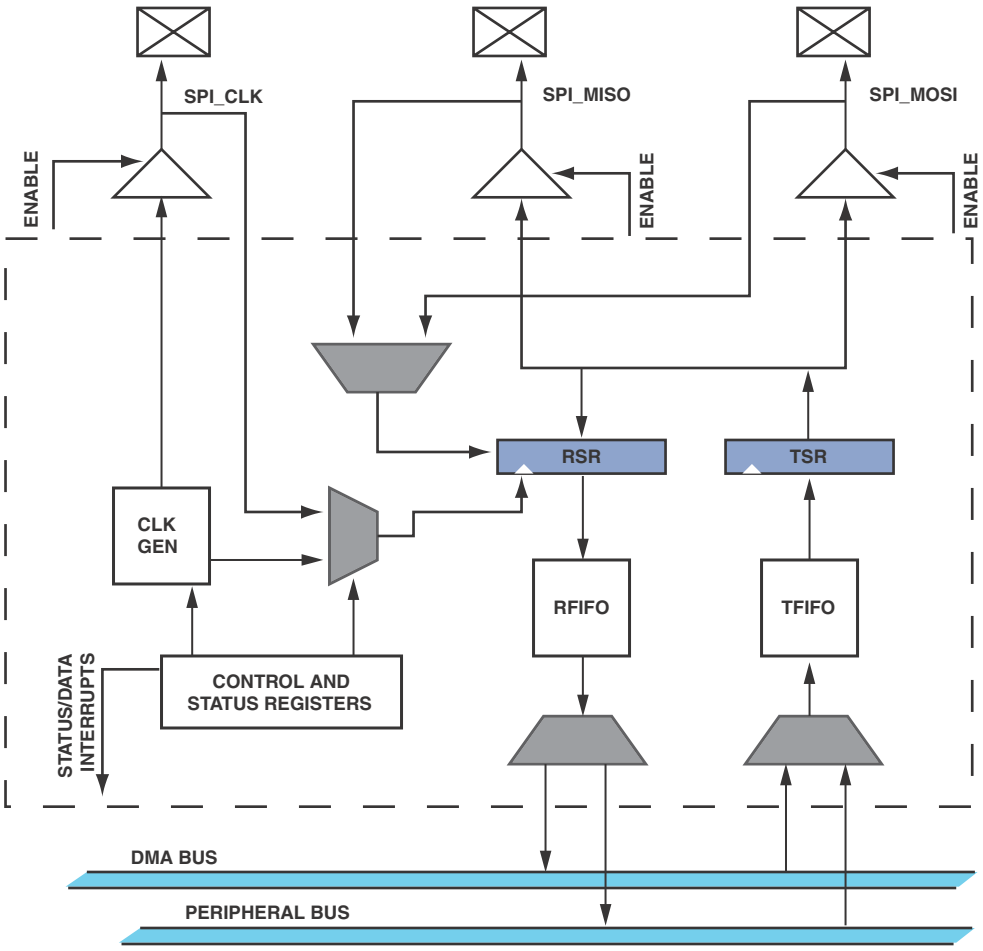


Figure 25-1: SPI Controller Block Diagram

Transfer Protocol

The SPI module implements two channels that are independent of each other. These channels are controlled by the SPI_RXCTL and SPI_TXCTL dedicated control registers. Except in dual and quad modes, both channels may be enabled and used 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 figures below 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 may 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 SPI_CLK signal is generated by the master, and the $\overline{\text{SPI\_SS}}$ 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 allowed. For example, a 16-bit transfer with the LSB first is another possible configuration.

The clock polarity and the clock phase should be identical for the master device and the slave device involved in the communication link. The transfer format from the master may be changed between transfers to adjust to various requirements of a slave device.

The SPI_CTL.ASSEL bit determines if the $\overline{\text{SPI\_SS}}$ line is controlled by the SPI hardware or by software. 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 $\overline{\text{SPI\_SS}}$ also depends on the parameters programmed into the SPI_DLY register. This is controlled automatically by the SPI hardware logic. When SPI_CTL.ASSEL=0, $\overline{\text{SPI\_SS}}$ may either remain active between successive transfers or be inactive. This must be controlled by the software via manipulation of the SPI_SLVSEL register.

The figures below illustrate the case when ASSEL = 1 and the $\overline{\text{SPI\_SS}}$ line is inactive between frames. If ASSEL = 0, the $\overline{\text{SPI\_SS}}$ line may remain active between frames; however, the first bit will only be driven when an active transition of SPI_CLK occurs.

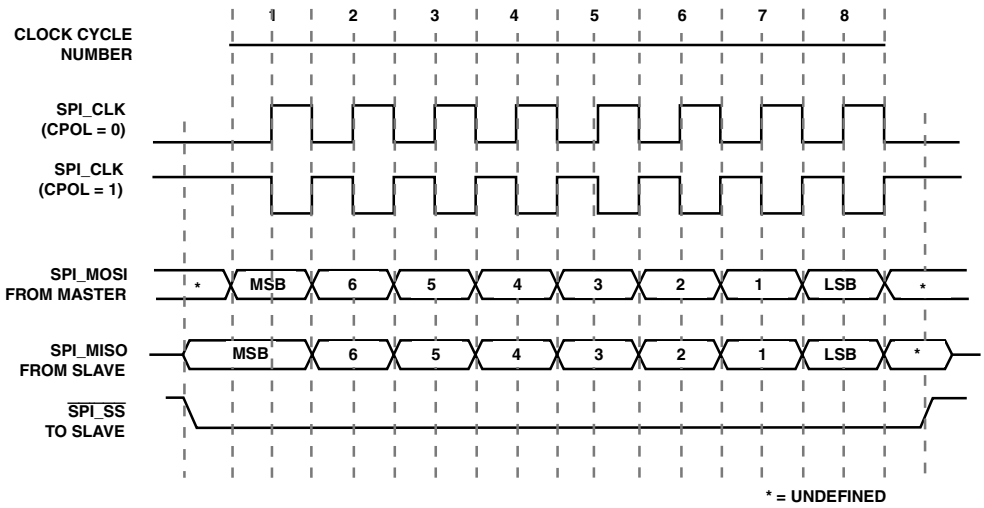


Figure 25-2: SPI Transfer Protocol for CPHA=0

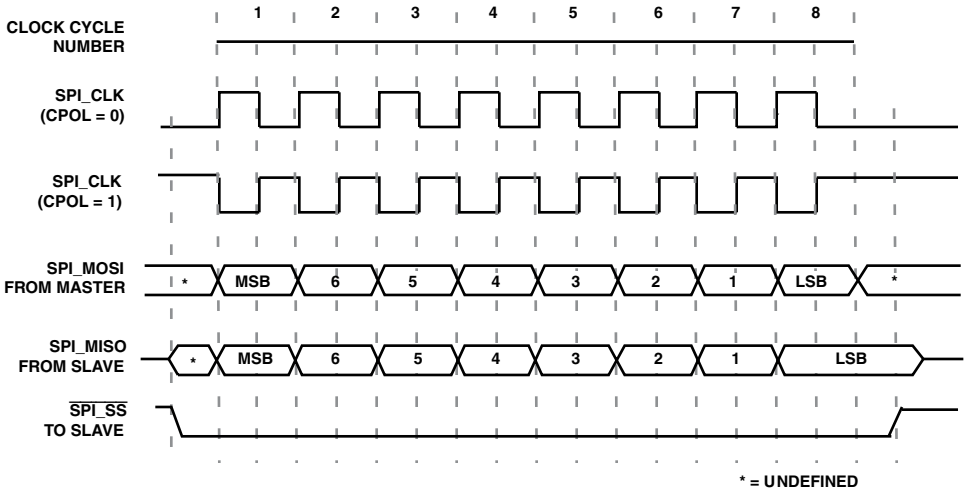


Figure 25-3: SPI Transfer Protocol for CPHA=1

SPI Clock Considerations

The SPI_CLK signal is a gated clock that is only active during data transfers, for the duration 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 as high as the SCLK rate, and both even and odd dividers from SCLK are supported. For master devices, the clock rate is determined by the SPI_CLKregister value, 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 if the slave select input is driven inactive. The SPI_CLK signal is used 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 figure below illustrates SPI timing with SPI_DLY.LEADX and SPI_DLY.LAGX programming. The LAGX controls the timing between the Slave Select (SPI_SEL) assertion and the first SPI_CLK edge, while LEADX controls the timing between the last SPI_CLK edge and deassertion of SPI_SEL. The lead and lag timing can be extended by 1 SPI_CLK duration to ease timing restrictions on the slave device.

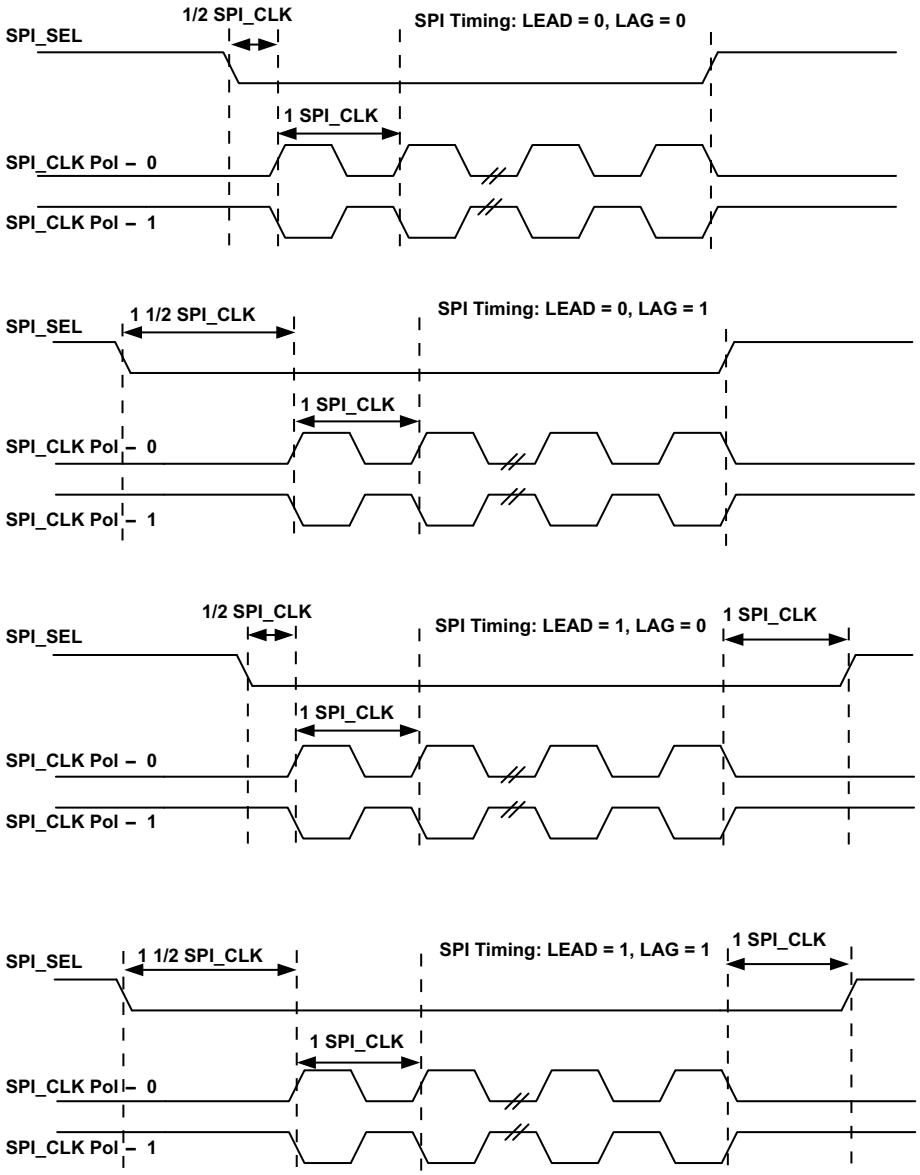


Figure 25-4: SPI Timing with Lead and Lag Programming (independent of SPI_CTL.CPHA setting)

The figure below illustrates SPI timing with STOP programming which is used to insert multiples of SPI_CLK period delays between transfers. The SPI_SEL line is deasserted for the duration specified in the SPI_DLY.STOP field, assuming the SPI_CTL.SELST bit is configured for deassertion between transfers.

If SPI_DLY.STOP is programmed to zero, the master operates in a *continuous mode*, resulting in 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.

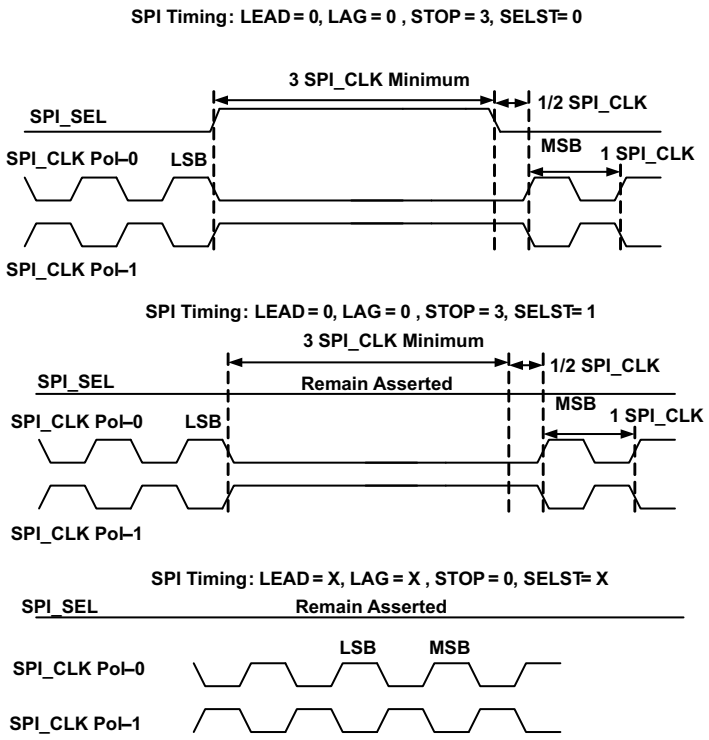


Figure 25-5: SPI Timing with SPI_DLY.STOP Programming (Independent of SPI_CTL.CPHA Setting)

When SPI_DLY.STOP is zero and initial conditions for a transfer are not met, the interface pauses before the next transfer. During this pause, the state of the slave select pin is determined by the SPI_CTL.SELST bit, and the SPI_DLY.LEAD and SPI_DLY.LAG bits determine the timing between SPI_CLK edges and the slave select line.

SPI Flow Control

In Master mode, the SPI_RDY pin acts as an input signal and should be driven by the slave device. SPI_RDY can be deasserted by the slave to stop the master from initiating any new transfer. If SPI_RDY is deasserted in the middle of a transfer, the current transfer will continue, and the next transfer will not start unless the slave asserts the SPI_RDYsignal. 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. This is controlled by the SPI_CTL.FCCH bit. If flow control is configured on the TX channel, as the SPI_TFIFO status nears the empty condition, SPI_RDY is deasserted. If flow control is configured on the RX channel, as the SPI_RFIFO status nears the full condition, SPI_RDY is deasserted. The FIFO status at which SPI_RDY deassertion should take place can be controlled by the SPI_CTL.FCWM bits. Note that flow control in Slave mode is purely based on the FIFO status and doesn't depend on the word counters.

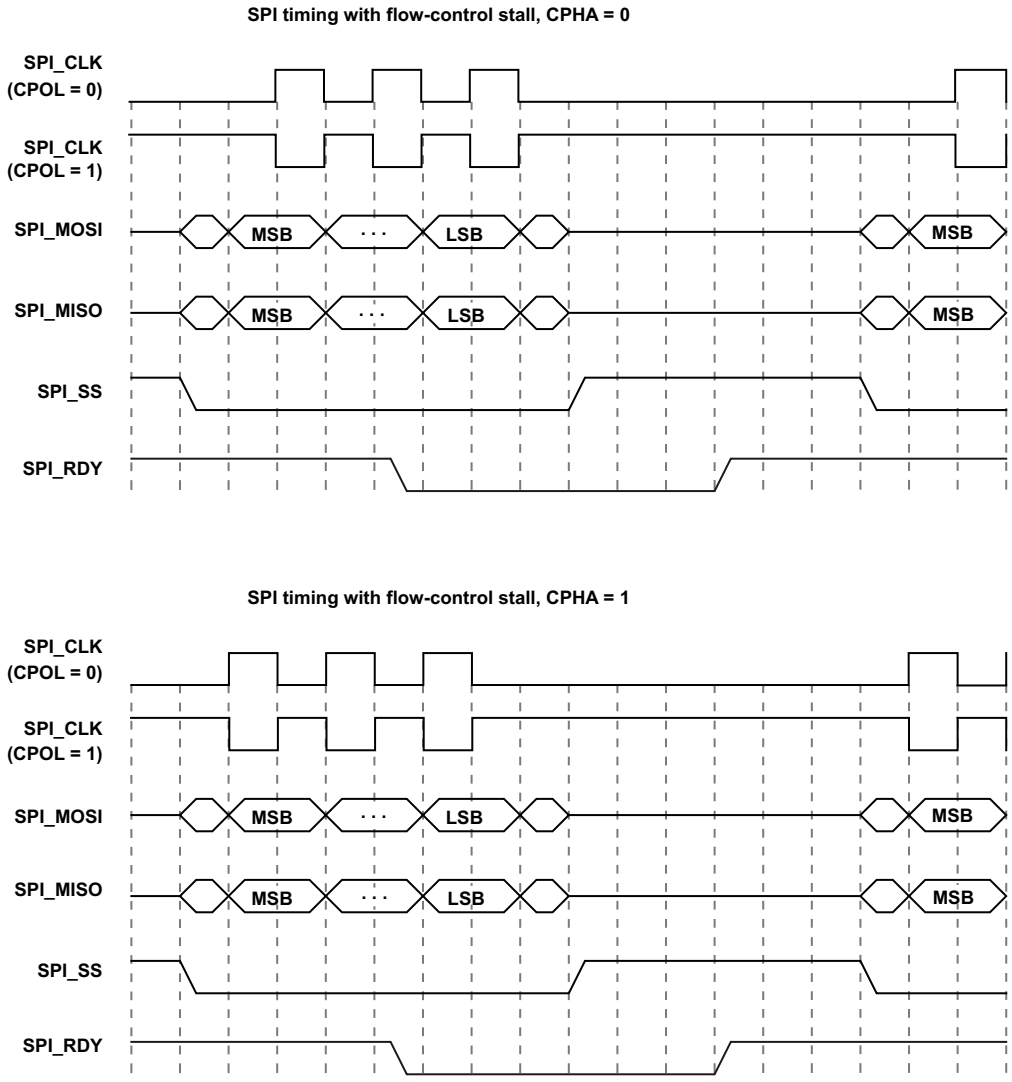


Figure 25-6: SPI Flow Control Timing in Master Mode.

Slave Select Operation

If the SPI is in slave mode, $\overline{\texttt{SPI\_SS}}$ acts as the slave select input. When SPI is enabled as a master, $\overline{\texttt{SPI\_SS}}$ 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 $\overline{\texttt{SPI\_SS}}$ input is the master mode error input. Otherwise, $\overline{\texttt{SPI\_SS}}$ is ignored.

The $\overline{\texttt{SPI\_SS}}$ signal is an active-low signal and should be asserted during the transfer by the master. It can be deasserted or remain asserted between transfers. When $\overline{\texttt{SPI\_SS}}$ is deasserted, SPI_CLK and inputs are ignored, and outputs are three-stated.

The slave select bits (SPI_SLVSEL.SSE1 – 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 processor's SPI is the master. The SPI_SLVSEL.SSE1 – 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 micro controllers/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 (enabled by clearing the SPI_CTL.EMISO bit in the six other slave processors) at a time and transmit broadcast data to all seven at the same time. This EMISO feature may be available in some other micro controllers. Therefore, it is possible to use the EMISO feature with any other SPI device that includes this functionality.

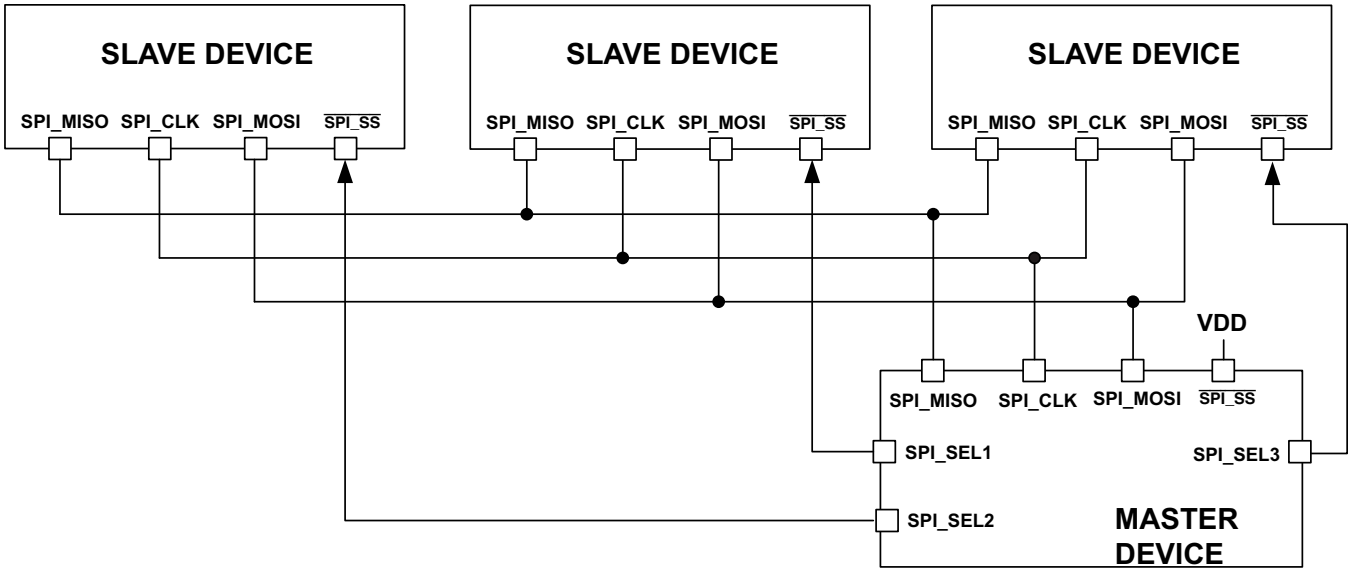


Figure 25-7: Single-Master, Multiple-Slave Configuration

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 $\overline{\text{SPI\_SS}}$ 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 considered finished 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=0, the hardware maintains responsibility for toggling $\overline{\text{SPI\_SS}}$ between frames. If SPI_CTL.ASSEL=1, software controls the $\overline{\text{SPI\_SS}}$ line and may 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, indicating that a new word has just 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 may have a slightly different behavior from that of other commercially available devices.

In master mode with the SPI_CTL.ASSEL bit cleared, software should manually assert the required slave select signal before starting the transaction. After all data has been transferred, software typically releases the slave select line.

When the receive or transmit word counters are enabled in the SPI_TXCTL or SPI_RXCTL registers, a finish interrupt is generated at the end of the transfer to signal the end of all transfers relating to that transaction.

Transmit Operation in Non-DMA Mode

Transmit operation on non-DMA mode is enabled through the SPI_TXCTL.TENbit. Transmit operation can be enabled independently from receive operation, and the transmit channel can become the initiating channel based on the SPI_TXCTL.TTIsetting.

Transmit underrun is not possible in this mode, as no new transfer would be initiated 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 can occur if SPI_TXCTL.TTI=1 and the receive channel is enabled in a non-initiating capacity.

A SPI transmit interrupt is signalled once the transmit channel has been enabled and the transmit FIFO is not full. The frequency of the interrupt is controlled by the SPI_TXCTL.TDR setting.

Receive Operation in Non-DMA Mode

Receive operation on non-DMA mode is enabled through the SPI_RXCTL.RENbit. Receive operation can be enabled independently from transmit operation, and the receive channel can become the initiating channel based on the SPI_RXCTL.RTIbit setting.

Receive overflow is not possible in this mode, as no new transfer would be initiated when the receive FIFO is full (in the case of SPI_RXCTL.RTI=1). A transmit underrun can occur (SPI_TXCTL.TDU bit) if no valid data is in the SPI_TFIFO register when a transfer is initiated. This can occur if SPI_RXCTL.RTI=1 and the transmit channel is enabled in a non-initiating capacity.

A SPI receive interrupt is signaled once the receive channel has been enabled and there is data waiting to be read. The frequency of the interrupt is controlled by the SPI_RXCTL.RDRbit setting.

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 order of bits on the pins are determined by the SPI_CTL.SOSI bit which, when set, 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 generation is unaffected by dual I/O mode. However, the gap between successive interrupts is reduced, since the individual transfer latency is halved.

Changing to Quad SPI mode should be done when the SPI is in a quiescent state.

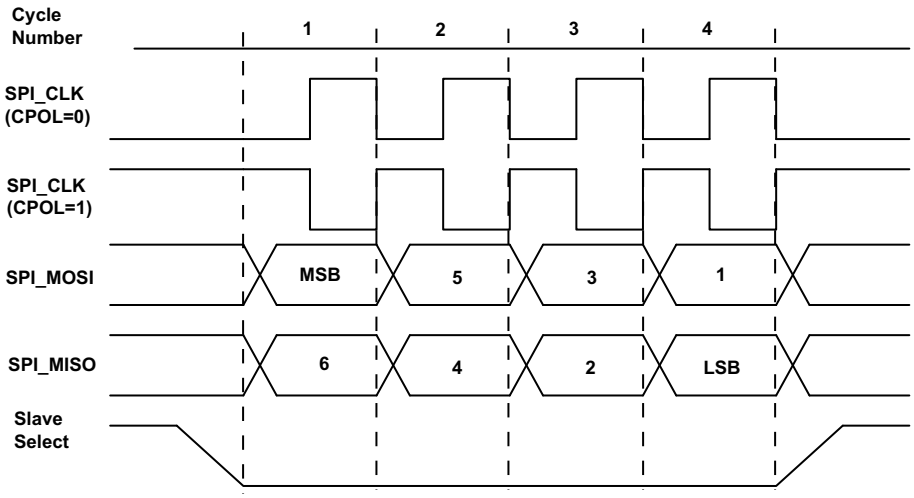


Figure 25-8: Dual I/O Mode Transfer Protocol for CPHA=0, SOSI=1, 8-Bit Transfer, LSBF=0.

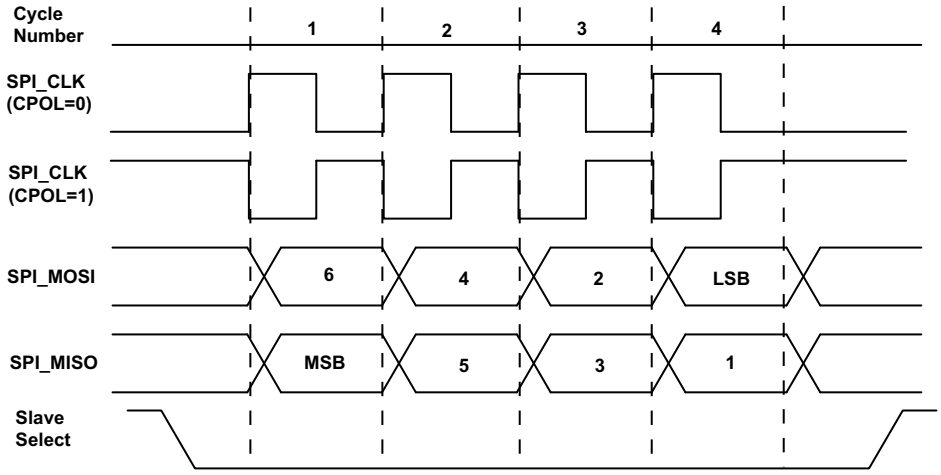


Figure 25-9: Dual I/O Mode Transfer Protocol for CPHA=1, SOSI=0, 8-Bit Transfer, LSBF=0.

Quad I/O Mode

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 order of bits on the pins are determined by the SPI_CTL.SOSI bit which, when set, 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 and 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, 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 should be done when the SPI is in a quiescent state.

While using Dual or Quad I/O mode for communicating with SPI Flash devices, it is advised to program the SPI_CTL.CPHAand the SPI_CTL.CPOLbits =1. This programming is to avoid bus contention during read operations, because the SPI flash device starts driving out the bits immediately after dummy cycles in read header.

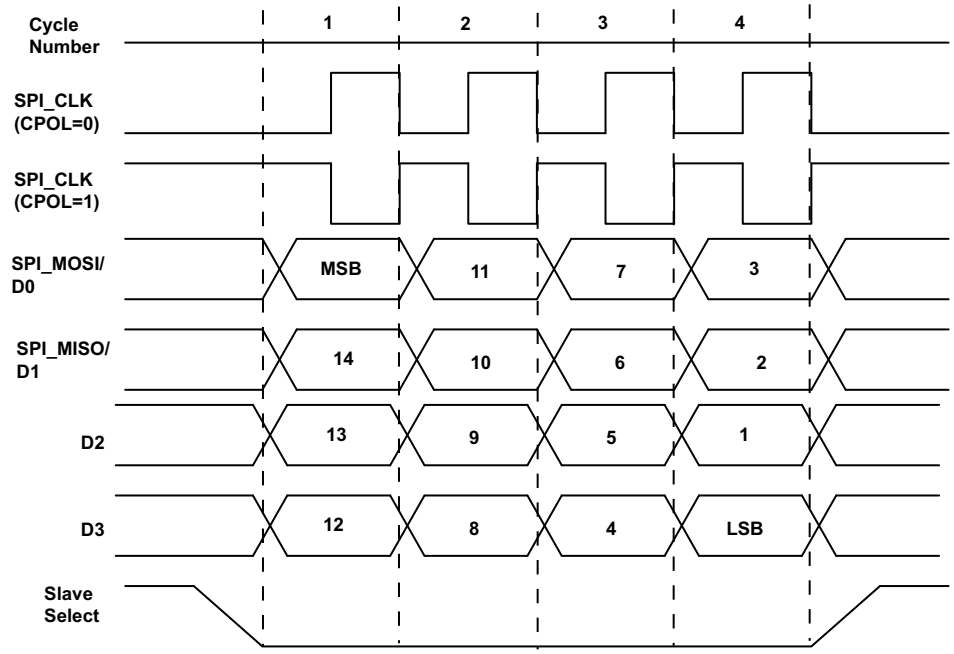


Figure 25-10: Quad Mode Timing for CPHA=0, SOSI=1, 16-Bit Transfer, LSBF=0.

NOTE: Quad SPI 8-bit transfer is not supported in Slave Continuous mode of operation with a *SCLK*:SPI_CLK ratio less than 1:2. A minimum of 2 *SCLK* cycles is required between transfers in 8-bit quad SPI slave mode with *SCLK*:SPI_CLK ratio less than 1:2.

Fast Mode

Fast Mode is similar to 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 is operating in fast mode, the slave should drive the data for one full cycle.

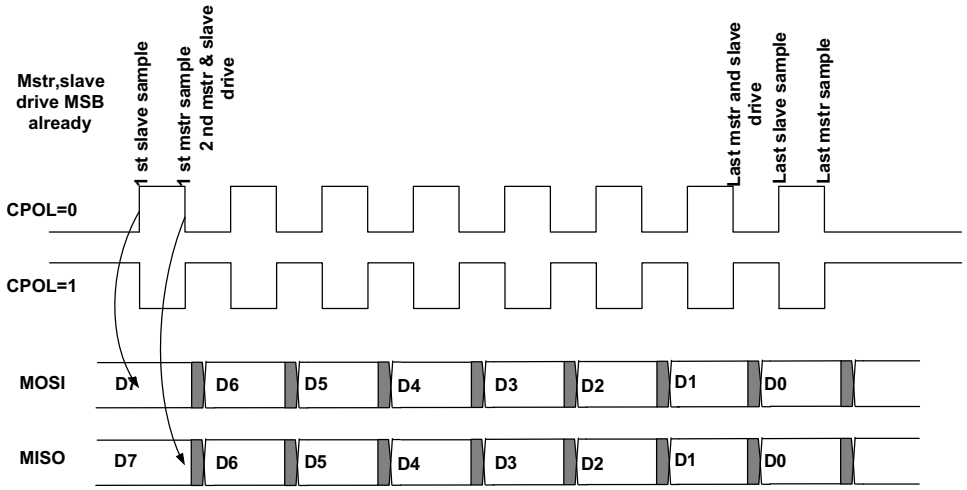


Figure 25-11: SPI Transfer Protocol in Fast Mode for SPI_CTL.CPHA = 0

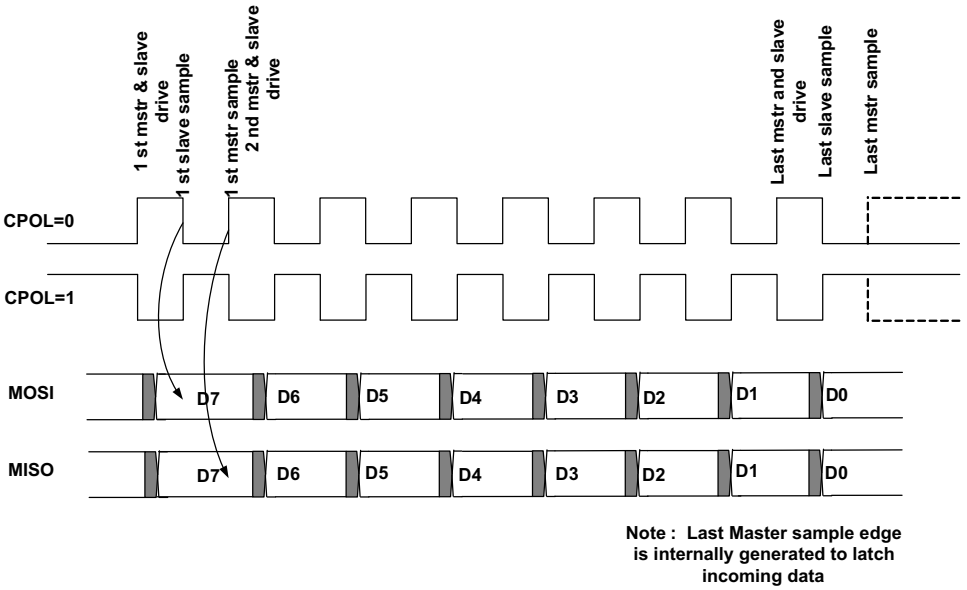


Figure 25-12: SPI Transfer Protocol in Fast Mode for SPI_CTL.CPHA = 1

SPI Interrupt Signals

The SPI controller supports three types of interrupt signals, corresponding 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 to independent read and write

interfaces with DMA, the read and write data interrupts are independent. When the DMA channel(s) are not being used, the interrupts are routed directly to the system event controller, occupying the same interrupt vector locations as the corresponding DMA channels do.

Each of the data interrupts can be individually controlled by programming the SPI_RXCTL.RDR and SPI_TXCTL.TDR bit fields for receive and transmit, respectively. When receive is enabled, the RX interrupt is issued whenever there is data available in the receive data path to be read (according to the SPI_RXCTL.RDR bit setting). When transmit is enabled, the TX interrupt is issued whenever the transmit data path can be written to (according to the SPI_TXCTL.TDR setting). DMA data interrupts are made compatible with second generation DMA to incorporate urgent data request and transfer finish interrupt apart from usual data request interrupt. Note that transmit interrupts operate independently from the word counter value in the SPI_TWC register.

Status Interrupts

The SPI controller supports several status interrupts to indicate different conditions of the receiver and transmitter. All status interrupts can be masked. Status interrupts are signaled directly through a single SPI status IRQ line, which may or may not be combined with the SPI error IRQ line for a given processor. The following table describes the status interrupts that are available for the SPI controller.

Table 25-5: SPI Status Interrupts

SPI_STAT Bit	Description
SPI_STAT.RUWM	Receive FIFO Urgent Watermark Interrupt. Issued when the level of the RFIFO breaches the watermark set in the SPI_RXCTL.RUWM field. It is cleared when the level of the RFIFO reaches the watermark set in the SPI_RXCTL.RRWM field. If the RX channel is configured in DMA mode, RUWM is multiplexed with the data request.
SPI_STAT.TUWM	Receive FIFO Urgent Watermark Interrupt. Issued when the level of the TFIFO breaches the watermark set using the SPI_TXCTL.TUWM bit. It is cleared when the level of the TFIFO reaches the watermark set in the SPI_TXCTL.TRWM field. If the TX channel is configured in DMA mode, TUWM is multiplexed with the data request.
SPI_STAT.TS	Transmit Start Interrupt. Issued when the start of a transmit burst is detected by loading of the SPI_TWC register with the contents of the SPI_TWCR register.
SPI_STAT.RS	Receive Start Interrupt. Issued when the start of a receive burst is detected by loading of SPI_RWC with the contents of SPI_RWCR.
SPI_STAT.TF	Transmit Finish Interrupt. Issued when a transmit burst completes (SPI_TWC decrements to zero).
SPI_STAT.RF	Receive Finish Interrupt. Issued when a receive burst completes (SPI_RWC decrements to zero).

Error Conditions

The SPI controller supports interrupts upon several different error conditions. All interrupts are maskable. The individual interrupt indications combine into a single SPI error IRQ signal, which may be multiplexed on some processors with the aggregated SPI status IRQ signal. The following table details the possible error indications.

Error conditions and interrupts arise depending on which of the channels (transmit and/or receive) are enabled. If a channel is disabled, all errors relating to it are ignored. When both channels are enabled, errors and interrupts from both channels are enabled.

Table 25-6: SPI Error Interrupts

Bit	Description
SPI_STAT.MF	Mode Fault. Signalled when another device is also trying to be a master in a multi-master system and drives the $\overline{\text{SPI\_SS}}$ input low. This error is signalled in master mode operation.
SPI_STAT.TUR	Transmission Error. Signalled when an underflow condition occurs on the transmit channel. This occurs when a new transfer starts but SPI_TFIFO is empty. This error does not occur in master transmit initiating mode since SPI_TFIFO Not Empty is one of the conditions for transfer initiation.
SPI_STAT.ROR	Reception Error. Signalled when an overflow condition occurs on the receive channel. This occurs when a new data word is received, but the SPI_RFIFO is full. This error condition will not occur in master receive initiating mode since SPI_RFIFO not full is one of the conditions for transfer initiation.
SPI_STAT.TC	Transmit Collision Error. Signalled when loading data to the transmit shift register happens near the first transmitting edge of SPI_CLK. In Slave mode of operation, the SPI controller is unaware of when the next transfer starts, so loading of data to the transmit shift register may happen just after the transmitting edge. This will result in setup time not being met for the first bit being transmitted, and thus the transmitted data will be corrupted. In SPI_CTL.CPHA - 1 mode, the first SPI_CLK edge is taken as first transmitting edge, whereas if SPI_CTL.CPHA=0 the last SPI_CLK edge of the last transmission (SPI_CTL.SELST=1) or slave select deassertion (SPI_CTL.SELST=0) is taken as the first transmitting edge. This error is signalled only in Slave mode of operation. In Master mode of operation, it is always ensured that loading of data happens before the first transmitting edge of SPI_CLK.

SPI Programming Concepts

The following sections provide general programming guidelines and procedures.

Programming Guidelines

It is acceptable to program SPI_RX_CTL and SPI_TX_CTLregisters after programming SPI_CTL, but the initiating mode register and its counter register, if enabled, should be programmed after the non-initiating mode register. For example, if Transmit is the initiating mode and Receive is the non-initiating mode, then SPI_RX_CTL and SPI_RWC should be programmed before SPI_TX_CTL and SPI_TWC. If both transmit and receive are to be enabled in initiating mode, SPI_CTL should be enabled after programming both SPI_RX_CTL and SPI_TX_CTL.

These programming guidelines prevent SPI from starting a transfer when SPI registers are still being programmed. Other ways of programming are also allowed as long as commencement of communication is prevented by initiating conditions until all the utilized SPI registers are programmed.

Precautions must be taken to avoid data corruption when changing the SPI module configuration. The configuration must not be changed during a data transfer. Additionally, the clock polarity should only be changed when no slaves are selected. An exception to this is when a SPI communication link consists of a single master and a single slave, SPI_CTL.ASSEL = 0, and the slave select input of the slave is permanently tied low. In this case, the slave is always selected, and data corruption can be avoided by enabling the slave only after both the master and slave devices are configured.

The module supports 8, 16 and 32-bit word sizes. To ensure correct operation, both the master and slave must be configured 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 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, enabling the device as a master and configuring the SPI system by specifying 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, the user activates the desired slaves by clearing one or more of the SPI_SLVSEL flag bits. Otherwise, the SPI hardware takes care of 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 the user to utilize the unused 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, the start of a transfer is triggered by a transition of the $\overline{\text{SPI\_SS}}$ select signal to the active state (low), or by the first active edge of SPI_CLK, 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 to define 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 $\overline{\text{SPI\_SS}}$ falling edge is detected, the slave starts sending data on active SPI_CLK edges and sampling data on inactive SPI_CLK edges.
4. Reception/transmission continues until $\overline{\text{SPI\_SS}}$ is released or until the slave has received the proper number of clock cycles.
5. The slave device continues to receive/transmit with each new falling edge transition on $\overline{\text{SPI\_SS}}$ and/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. These may be used individually or in a lock-step manner in duplex mode (SPI_TXCTL.TTI= SPI_RXCTL.RTI=1)

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.
2. Write to the SPI_SLVSEL register, setting one or more of the SPI flag select bits.
3. Write to the SPI_CLK and SPI_CTLregisters, enabling the device as a master and configuring the SPI system by specifying the appropriate word length, transfer format, baud rate, etc.
4. Write to SPI_RXCTL to configure SPI master receive mode, and/or write to SPI_TXCTL to configure SPI Master Transmit mode.

5. Finally, write to the SPI_RXCTL.REN bit to enable the receive channel, and/or write to SPI_TXCTL.TEN to enable the transmit channel.

6. 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_RWCregister 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 so that 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 DMA write 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, provided that SPI_RWC does not become zero while SPI_RXCTL.RWCEN=1.

7. 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_TWCregister is non-zero if SPI_TXCTL.TWCEN=1, the SPI continues to initiate transfers until disabled.

8. If both the SPI_TXCTL.TTI and SPI_RXCTL.RTI bits are enabled, the SPI controller requests a DMA read from memory as long as there is space for more data in the transmit pipe and the number of words written into the SPI is less than SPI_TWCif 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.

   ADDITIONAL INFORMATION: Data received from the transfer is moved from the SPI receive shift register to the receive FIFO.

   ADDITIONAL INFORMATION: The SPI controller requests a DMA write to memory.

   ADDITIONAL INFORMATION: 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_RWCexpires, 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

When enabled as a slave with the DMA engine configured to transmit or receive data, the start of a transfer is triggered by a transition of the $\overline{\text{SPI\_SS}}$ signal to the active-low state or by the first active edge of SPI_CLK, 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 DMA write 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 until the SPI_RWC counts to zero. The SPI slave continues receiving words on active SPI_CLK edges as long as the $\overline{\text{SPI\_SS}}$ input is active.
   f. If the data collected in the receive pipe breaches the level set according to the SPI_CTL.FCWM field, and the DMA engine is unable to keep up with the receive rate, the slave may de-assert the SPI_RDY signal, throttling the master. The signal is deasserted as the DMA drains the receive FIFO. Alternatively, the SPI_RXCTL.RDO bit can decide if 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

until the SPI_TWC register value counts down to 0. The SPI slave continues transmitting words on active SPI_CLK edges as long as the $\overline{\text{SPI\_SS}}$ input is active.

f. If the number of outstanding data entries waiting for transmission in the transmit pipe breaches the level set according to the SPI_CTL.FCWM field and the DMA is unable to keep up with the transmit rate, the slave may deassert the SPI_RDY signal, throttling the master. 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 stated above for each channel. Transfers will continue as long as both receives and transmits can accommodate new data.

   a. If the receive pipe fills up due to unavailability of DMA grant, the SPI interface will stall the master by asserting the SPI_RDY pin. This signal is deasserted as the DMA drains the receive FIFO. Alternately, the SPI_RXCTL.RDO bit decides if the incoming data is discarded or overwritten into the receive FIFO (when SPI_CTL.FCEN is deasserted).

   b. If the transmit pipe fills up, the SPI will stop requesting for DMA writes until the pipe clears.

   c. If there is an underflow problem in the transmit pipe, the slave will stall the master by deasserting SPI_RDY while DMA fills the transmit FIFO. Alternately, the SPI_TXCTL.TDU bit decides the state of the transmit data (when SPI_CTL.FCEN is deasserted).

ADSP-BF60x SPI Register Descriptions

Serial Peripheral Interface (SPI) contains the following registers.

Table 25-7: ADSP-BF60x SPI Register List

Name	Description
SPI_CTL	Control Register
SPI_RXCTL	Receive Control Register
SPI_TXCTL	Transmit Control Register
SPI_CLK	Clock Rate Register
SPI_DLY	Delay Register
SPI_SLVSEL	Slave Select Register
SPI_RWC	Received Word Count Register
SPI_RWCR	Received Word Count Reload Register

Table 25-7: ADSP-BF60x SPI Register List (Continued)

Name	Description
SPI_TWC	Transmitted Word Count Register
SPI_TWCR	Transmitted Word Count Reload Register
SPI_IMSK	Interrupt Mask Register
SPI_IMSK_CLR	Interrupt Mask Clear Register
SPI_IMSK_SET	Interrupt Mask Set Register
SPI_STAT	Status Register
SPI_ILAT	Masked Interrupt Condition Register
SPI_ILAT_CLR	Masked Interrupt Clear Register
SPI_RFIFO	Receive FIFO Data Register
SPI_TFIFO	Transmit FIFO Data Register

Control Register

The SPI_CTL register enables the SPI and configures settings for operating modes, communication protocols, and buffer operations.

SPI_CTL: Control Register - R/W

Reset = 0x0000 0050

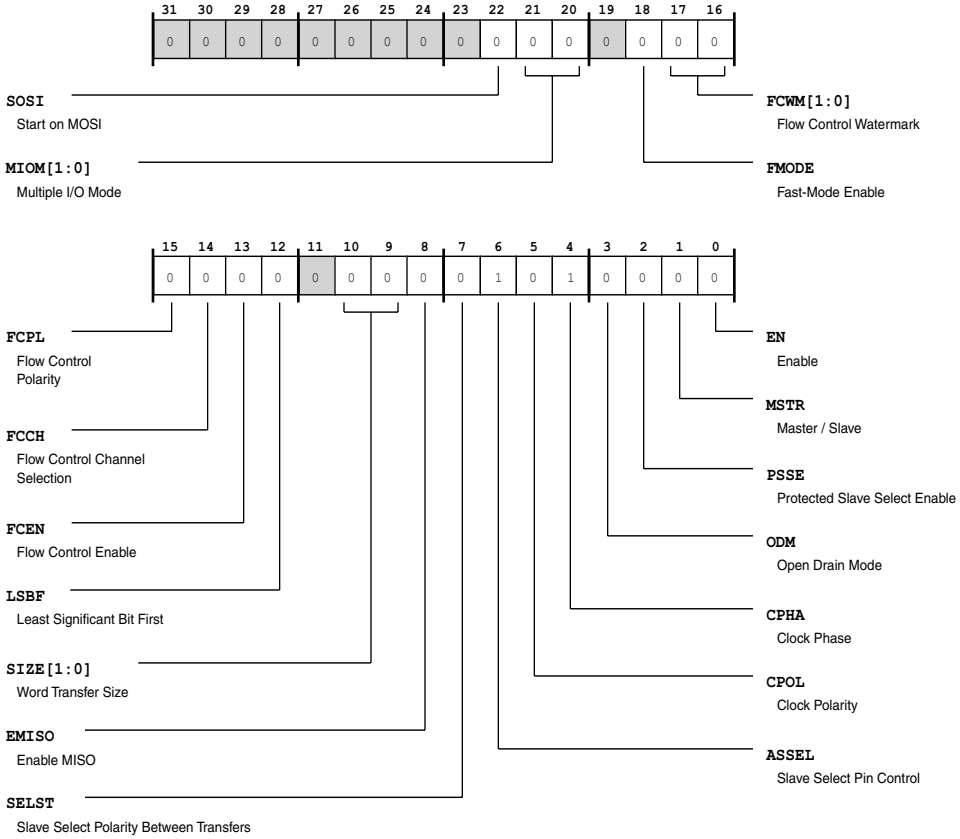


Figure 25-13: SPI_CTL Register Diagram

Table 25-8: SPI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/W)	SOSI	Start on MOSI. The SPI_CTL.SOSI bit is valid only when SPI_CTL.MIOM 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 (SPI_CTL.SOSI =0) SPI sends first bit on the SPI_MISO pin and second bit on the SPI_MOSI pin. In QIOM, by default, the SPI sends first bit on the SPI_D3 pin, second bit on the SPI_D2 pin, third bit on the SPI_MISO pin and fourth bit on the SPI_MOSI pin. This order can be reversed by setting the SPI_CTL.SOSI bit. When this bit is set, the SPI sends first bit on the SPI_MOSI pin. The first bit referred to here depends on the SPI_CTL.LSBF bit setting (MSB bit or LSB bit).	
		0	Start on MISO (DIOM) or start on SPIQ3 (QSPI)
		1	Start on MOSI
21:20 (R/W)	MIOM	Multiple I/O Mode. The SPI_CTL.MIOM bits enable SPI operation in dual I/O mode (DIOM) or quad I/O mode (QIOM). These bits may only be changed when the SPI is disabled (SPI_CTL.EN =0).	
		0	No MIOM (disabled)
		1	DIOM operation
		2	QIOM operation
		3	Reserved
18 (R/W)	FMODE	Fast-Mode Enable. The SPI_CTL.FMODE 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

Table 25-8: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17:16 (R/W)	FCWM	Flow Control Watermark. The SPI_CTL.FCWM bits select the watermark level of the transmit channel (SPI_TFIFO buffer) or receive channel (SPI_RFIFO buffer) that triggers flow control operation. These bits are applicable only when the SPI is a slave (SPI_CTL.MSTR = 0) and flow control is enabled (SPI_CTL.FCEN =1). When the watermark condition is met, the SPI slave de-asserts the SPI_RDY 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 SPI_CTL.FCPL bit selects flow control polarity for the SPI_RDY pin when flow control is enabled. When the SPI_RDY pin is active, the SPI is indicating it's ready for data transfer.	
		0	Active-low RDY
		1	Active-high RDY
14 (R/W)	FCCH	Flow Control Channel Selection. The SPI_CTL.FCCH bit selects whether the SPI applies flow control to the transmit channel (SPI_TFIFO buffer) or receive channel (SPI_RFIFO 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 SPI_CTL.FCEN bit enables SPI flow control operation, which permits slow slave devices to interface with fast master devices. This bit controls operation of the SPI_RDY pin. Note that options for flow control operation are available using the SPI_CTL.FCCH, SPI_CTL.FCPL, and SPI_CTL.FCWM bits.	
		0	Disable
		1	Enable

Table 25-8: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	LSBF	Least Significant Bit First. The SPI_CTL.LSBF bit selects whether the SPI transmits/receives data as LSB first (little endian) or MSB first (big endian). This bit may 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 SPI_CTL.SIZE 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 may only be changed when the SPI is disabled (SPI_CTL.EN =0).	
		0	8-bit word
		1	16-bit word
		2	32-bit word
		3	Reserved
8 (R/W)	EMISO	Enable MISO. The SPI_CTL.EMISO bit enables master-in-slave-out (MISO) mode. This SPI mode is applicable only when the SPI is a slave.	
		0	Disable
		1	Enable
7 (R/W)	SELST	Slave Select Polarity Between Transfers. The SPI_CTL.SELST bit selects the state (polarity) for the $\overline{\text{SPI\_SELn}}$ pin in-between SPI transfers when the SPI is a master and hardware slave select assertion is enabled (SPI_CTL.ASSEL =1). In slave mode, this bit affects the detection of both transmit collision (SPI_STAT.TC and under-run (SPI_STAT.TUR) errors.	
		0	De-assert slave select (high)
		1	Assert slave select (low)

Table 25-8: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	ASSEL	Slave Select Pin Control. The SPI_CTL.ASSEL bit selects whether the SPI hardware sets the $\overline{\text{SPI\_SELn}}$ pin output value (ignoring the slave select SPI_SLVSEL.SSEL1 - SPI_SLVSEL.SSEL7 bits) or whether software control of the slave select bits set the $\overline{\text{SPI\_SELn}}$ pin output value. This feature is applicable only when the SPI is a master. When hardware control is enabled, the $\overline{\text{SPI\_SELn}}$ pin output is asserted during the transfers, and the pin polarity between transfers is selected by the SPI_CTL.SELST bit. When software control is enabled, the $\overline{\text{SPI\_SELn}}$ 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 SPI_CTL.CPOL bit selects whether the SPI uses an active-low or active-high signal for the SPI clock (SPI_CLK). This bit works with the SPI_CTL.CPHA bit to select combinations of clock phase and polarity for the SPI_CLK pin. This bit may 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 SPI_CTL.CPHA bit selects whether the SPI starts toggling the signal for the SPI clock (SPI_CLK) from the start of the first data bit or from the middle of the first data bit. The SPI_CTL.CPHA bit works with the SPI_CTL.CPOL bit to select combinations of clock phase and polarity for the SPI_CLK pin. This bit may only be changed when the SPI is disabled.	
		0	SPI CLK toggles from middle
		1	SPI CLK toggles from start

Table 25-8: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	ODM	Open Drain Mode. The SPI_CTL.ODM bit configures the data output pins (SPI_MOSI and SPI_MISO) to behave as open drain outputs, which prevents contention and possible damage to pin drivers in multi-master or multi-slave SPI systems. When SPI_CTL.ODM is enabled and the SPI is a master, the SPI three-states the SPI_MOSI pin when the data driven out on MOSI is a logic-high. The SPI does not three-state the SPI_MOSI pin when the driven data is a logic-low. When SPI_CTL.ODM is enabled and the SPI is a slave, the SPI three-states the SPI_MISO pin when the data driven out on SPI_MISO is a logic-high. Note that an external pull-up resistor is required on both the SPI_MOSI and SPI_MISO pins when SPI_CTL.ODM is enabled.	
		0	Disable
		1	Enable
2 (R/W)	PSSE	Protected Slave Select Enable. The SPI_CTL.PSSE bit enables the $\overline{\text{SPI\_SS}}$ 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 $\overline{\text{SPI\_SS}}$ pin while SPI is enabled as master (and SPI_CTL.PSSE is enabled), this condition causes a mode fault error.	
		0	Disable
		1	Enable
1 (R/W)	MSTR	Master / Slave. The SPI_CTL.MSTR bit toggles the SPI between master mode and slave mode. This bit may only be changed when the SPI is disabled.	
		0	Slave
		1	Master
0 (R/W)	EN	Enable. The SPI_CTL.EN bit enables SPI operation.	
		0	Disable SPI module
		1	Enable

Receive Control Register

The SPI_RXCTL register enables the SPI receive channel, initiates receive transfers, and configures SPI_RFIFO buffer watermark settings.

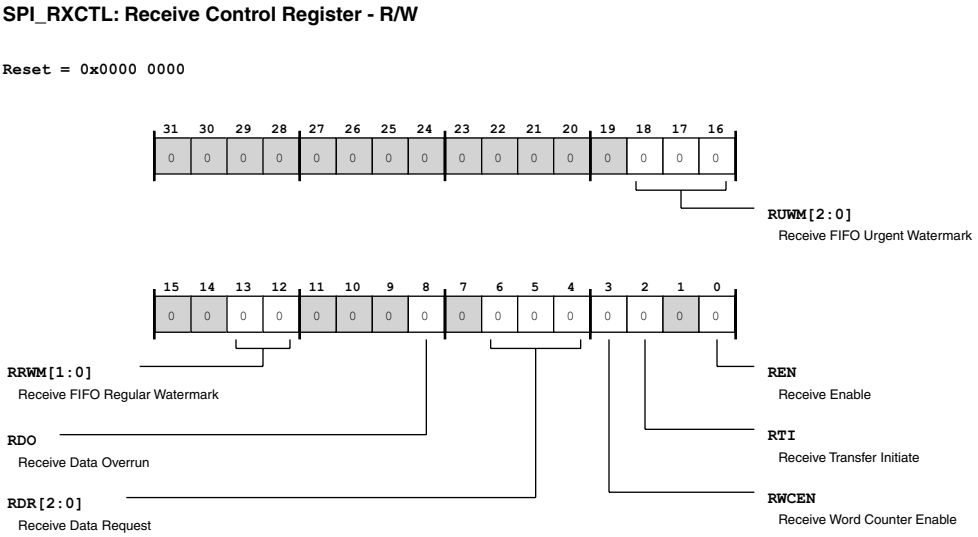


Figure 25-14: SPI_RXCTL Register Diagram

Table 25-9: SPI_RXCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	RUWM	Receive FIFO Urgent Watermark. The SPI_RXCTL.RUWM bits select the receive FIFO (SPI_RFIFO) watermark level for urgent data bus requests. The SPI also uses this watermark level for generation of the SPI_ILAT.RUWM interrupt. When an urgent SPI_RFIFO watermark is enabled with SPI_RXCTL.RUWM, the SPI_RXCTL.RRWM selection is used as the de-assertion condition for any SPI_ILAT.RUWM 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 25-9: SPI_RXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13:12 (R/W)	RRWM	Receive FIFO Regular Watermark. The SPI_RXCTL.RRWM bits select the receive FIFO (SPI_RFIFO) watermark level for regular data bus requests. When an urgent SPI_RFIFO watermark is enabled with SPI_RXCTL.RUWM, the SPI_RXCTL.RRWM selection is used as the de-assertion condition for any SPI_ILAT.RUWM 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 SPI_RXCTL.RDO bit selects handling for receive data requests when the receive buffer (SPI_RFIFO) is full. If enabled and SPI_RFIFO is full, the SPI overwrites old data in the buffer with incoming data. If disabled and SPI_RFIFO is full, the SPI keeps old data in the buffer and discards incoming data.	
		0	Discard incoming data if SPI_RFIFO is full
		1	Overwrite old data if SPI_RFIFO is full
6:4 (R/W)	RDR	Receive Data Request. The SPI_RXCTL.RDR bits select receive FIFO (SPI_RFIFO) 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

Table 25-9: SPI_RXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	RWCEN	Receive Word Counter Enable. The SPI_RXCTL.RWCEN bit enables the decrement of the SPI_RWC register when the count is not zero and SPI_RXCTL.RTI is enabled. Enabling SPI_RXCTL.RWCEN prevents receive overrun errors from occurring. The SPI_RXCTL.RWCEN bit is valid only when the SPI is a master.	
		0	Disable
		1	Enable
2 (R/W)	RTI	Receive Transfer Initiate. The SPI_RXCTL.RTI bit enables initiation of receive transfers if the receive FIFO (SPI_RFIFO) is not full. The bit also enables this initiation if SPI_RWC is not zero when SPI_RXCTL.RWCEN is enabled. Enabling SPI_RXCTL.RTI prevents receive overrun errors from occurring. The SPI_RXCTL.RTI bit is valid only when the SPI is a master.	
		0	Disable
		1	Enable
0 (R/W)	REN	Receive Enable. The SPI_RXCTL.REN bit enables SPI receive channel operation.	
		0	Disable
		1	Enable

Transmit Control Register

The SPI_TXCTL register enables the SPI transmit channel, initiates transmit transfers, and configures SPI_TFIFO buffer watermark settings.

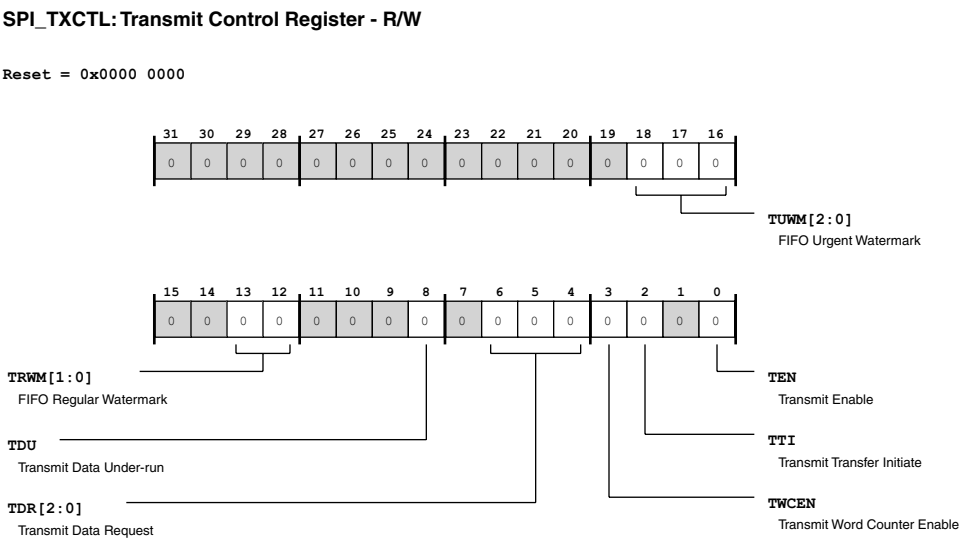


Figure 25-15: SPI_TXCTL Register Diagram

Table 25-10: SPI_TXCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	TUWM	FIFO Urgent Watermark. The SPI_TXCTL.TUWM bits select the transmit FIFO (SPI_TFIFO) watermark level for urgent data bus requests. The SPI also uses this watermark level for generation of the SPI_ILAT.TUWM interrupt. When an urgent SPI_TFIFO watermark is enabled with SPI_TXCTL.TUWM, the SPI_TXCTL.TRWM selection is used as the de-assertion condition for any SPI_ILAT.TUWM interrupts that are latched.	
		0	Disabled
		1	25% empty TFIFO
		2	50% empty TFIFO
		3	75% empty TFIFO
		4	Empty TFIFO

Table 25-10: 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 de-assertion condition for any SPI_ILAT.TUWM interrupts 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 Under-run. 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.	
		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 under-run errors from occurring. The SPI_TXCTL.TWCEN bit is valid only when the SPI is a master.	
		0	Disable
		1	Enable

Table 25-10: SPI_TXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	TTI	Transmit Transfer Initiate. The SPI_TXCTL.TTI bit enables initiation of transmit transfers if the transmit FIFO (SPI_TFIFO) is not empty. The bit also enables this initiation if SPI_TWC is not zero when SPI_TXCTL.TWCEN is enabled. Enabling SPI_TXCTL.TTI prevents transmit underrun errors from occurring. The SPI_TXCTL.TTI bit is valid only when the SPI is a master.	
		0	Disable
		1	Enable
0 (R/W)	TEN	Transmit Enable. The SPI_TXCTL.TEN bit enables SPI transmit channel operation.	
		0	Disable
		1	Enable

Clock Rate Register

The SPI_CLK register selects the baud rate for SPI data transfers, relating this rate to the SPI serial clock (SCK) and the system clock (SCLK).

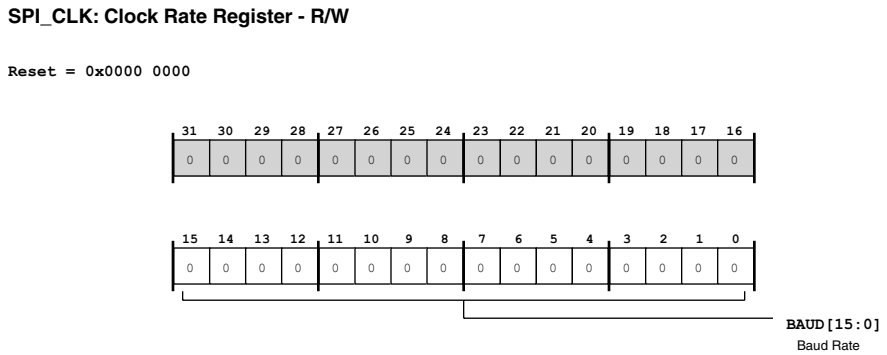


Figure 25-16: SPI_CLK Register Diagram

Table 25-11: SPI_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	BAUD	Baud Rate. The SPI_CLK.BAUD bits set the SPI baud rate according to the formula: BAUD = (SCLK / SPI Clock) - 1

Delay Register

The SPI_DLY register selects a transfer delay and the lead/lag timing between slave select signals and SPI clock edge assertion/de-assertion.

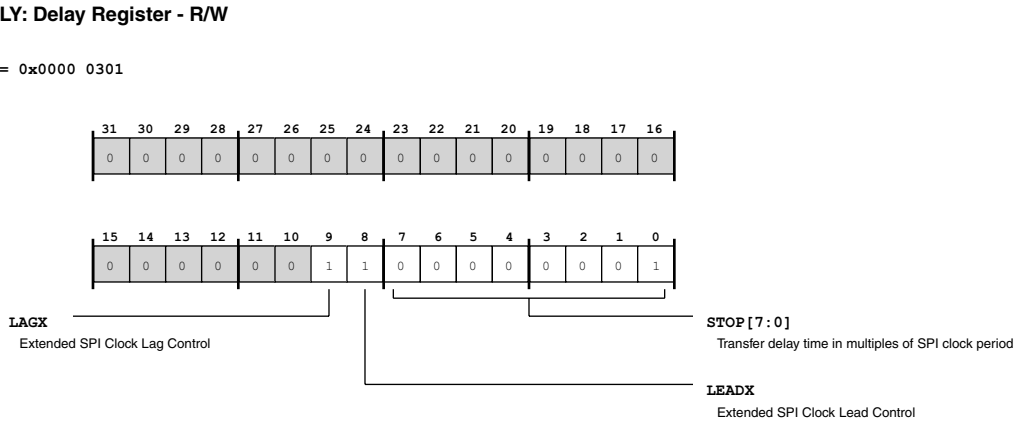


Figure 25-17: SPI_DLY Register Diagram

Table 25-12: SPI_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	LAGX	Extended SPI Clock Lag Control. The SPI_DLY.LAGX bit enables insertion of a 1-SPI_CLK cycle lag (extend lag) in the timing between the slave select ($\overline{\text{SPI\_SELn}}$) assertion and first SPI Clock edge.	
		0	Disable
		1	Enable
8 (R/W)	LEADX	Extended SPI Clock Lead Control. The SPI_DLY.LEADX bit enables insertion of a 1-SPI_CLK cycle lead (extend lead) in the timing between the slave select ($\overline{\text{SPI\_SELn}}$) de-assertion and last SPI Clock edge.	
		0	Disable
		1	Enable
7:0 (R/W)	STOP	Transfer delay time in multiples of SPI clock period. The SPI_DLY.STOP 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 SPI_DLY.STOP bits may be programmed with smaller delay values, resulting in continuous operation (for example, stop bits =0).	

Slave Select Register

The SPI_SLVSEL register enables the $\overline{\text{SPI\_SELn}}$ pins for input and indicates the state (high or low) of these pins when enabled.

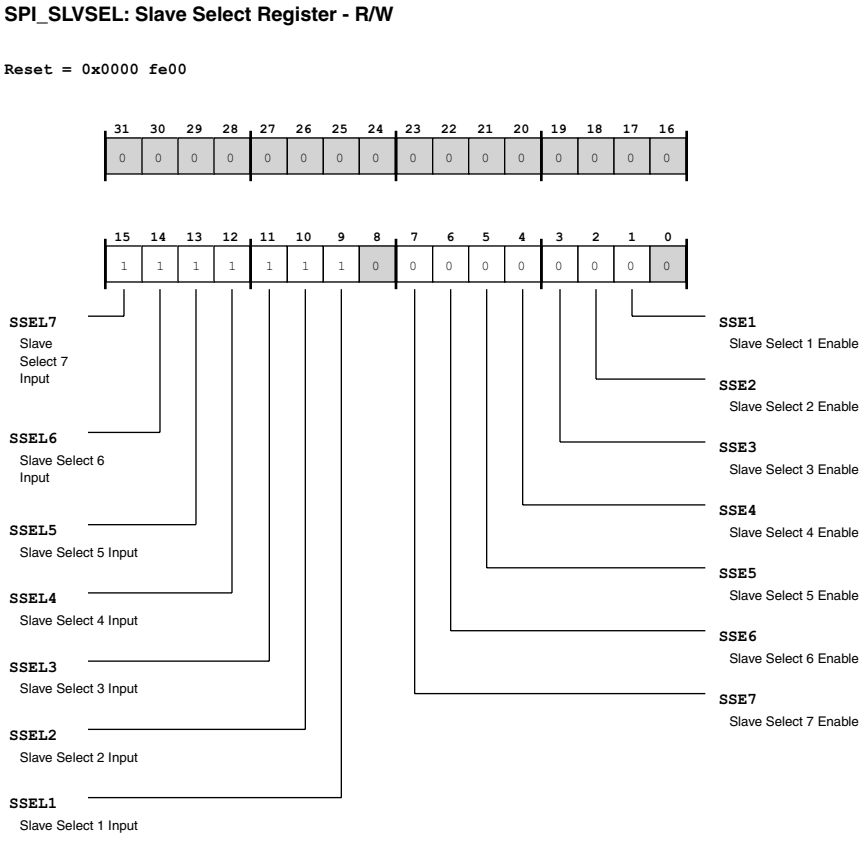


Figure 25-18: SPI_SLVSEL Register Diagram

Table 25-13: SPI_SLVSEL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	SSEL7	Slave Select 7 Input. The SPI_SLVSEL.SSEL7 bit state indicates the value on the related $\overline{\text{SPI\_SELn}}$ pin.	
		0	Low
		1	High

Table 25-13: SPI_SLVSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	SSEL6	Slave Select 6 Input. The SPI_SLVSEL.SSEL6 bit state indicates the value on the related $\overline{\text{SPI\_SELn}}$ pin.	
		0	Low
		1	High
13 (R/W)	SSEL5	Slave Select 5 Input. The SPI_SLVSEL.SSEL5 bit state indicates the value on the related $\overline{\text{SPI\_SELn}}$ pin.	
		0	Low
		1	High
12 (R/W)	SSEL4	Slave Select 4 Input. The SPI_SLVSEL.SSEL4 bit state indicates the value on the related $\overline{\text{SPI\_SELn}}$ pin.	
		0	Low
		1	High
11 (R/W)	SSEL3	Slave Select 3 Input. The SPI_SLVSEL.SSEL3 bit state indicates the value on the related $\overline{\text{SPI\_SELn}}$ pin.	
		0	Low
		1	High
10 (R/W)	SSEL2	Slave Select 2 Input. The SPI_SLVSEL.SSEL2 bit state indicates the value on the related $\overline{\text{SPI\_SELn}}$ pin.	
		0	Low
		1	High
9 (R/W)	SSEL1	Slave Select 1 Input. The SPI_SLVSEL.SSEL1 bit state indicates the value on the related $\overline{\text{SPI\_SELn}}$ pin.	
		0	Low
		1	High

Table 25-13: SPI_SLVSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	SSE7	Slave Select 7 Enable. The SPI_SLVSEL.SSE7 bit enables the related $\overline{\texttt{SPI\_SELn}}$ pin for input. If disabled, the SPI three-states the related $\overline{\texttt{SPI\_SELn}}$ pin. When the SPI is a slave, the master (not the SPI) asserts the input during the transfer. The input may be de-asserted or remain asserted between transfers. While the input is de-asserted, the SPI ignores SPI Clock, ignores inputs, and three-states outputs.	
		0	Disable
		1	Enable
6 (R/W)	SSE6	Slave Select 6 Enable. The SPI_SLVSEL.SSE6 bit enables the related $\overline{\texttt{SPI\_SELn}}$ pin for input. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable
5 (R/W)	SSE5	Slave Select 5 Enable. The SPI_SLVSEL.SSE5 bit enables the related $\overline{\texttt{SPI\_SELn}}$ pin for input. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable
4 (R/W)	SSE4	Slave Select 4 Enable. The SPI_SLVSEL.SSE4 bit enables the related $\overline{\texttt{SPI\_SELn}}$ pin for input. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable
3 (R/W)	SSE3	Slave Select 3 Enable. The SPI_SLVSEL.SSE3 bit enables the related $\overline{\texttt{SPI\_SELn}}$ pin for input. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable

Table 25-13: SPI_SLVSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	SSE2	Slave Select 2 Enable. The SPI_SLVSEL.SSE2 bit enables the related $\overline{\text{SPI\_SELn}}$ pin for input. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable
1 (R/W)	SSE1	Slave Select 1 Enable. The SPI_SLVSEL.SSE1 bit enables the related $\overline{\text{SPI\_SELn}}$ pin for input. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable

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 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 maintains the number of frames to be received in a transfer. The SPI_RWC should only be changed when the counter is disabled.

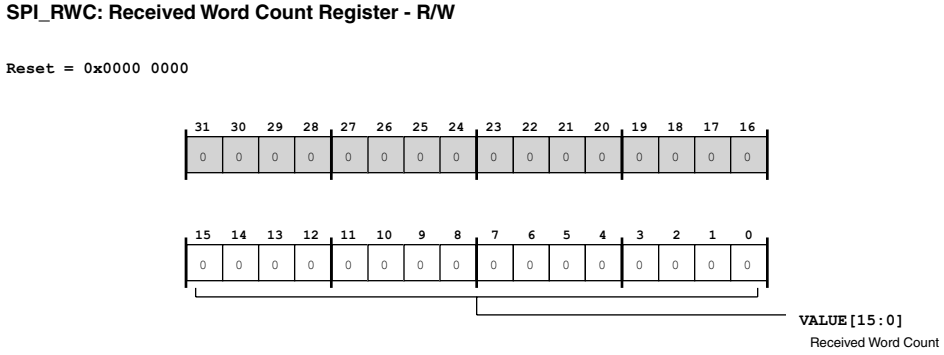


Figure 25-19: SPI_RWC Register Diagram

Table 25-14: SPI_RWC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Received Word Count. The SPI_RWC.VALUE 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 should only be changed when the counter is disabled.

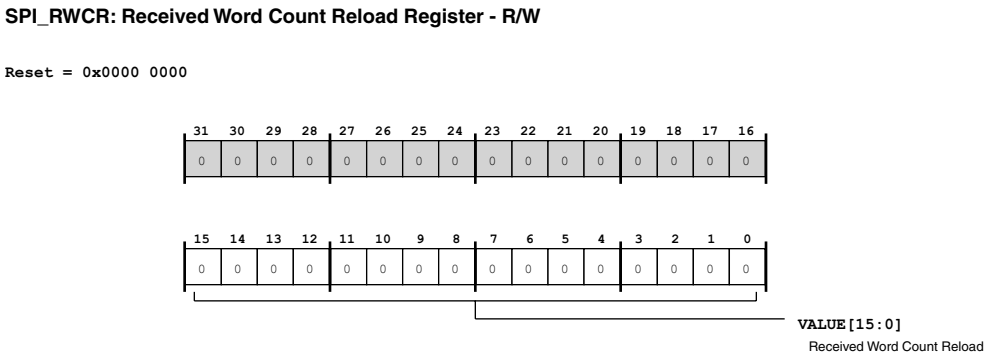


Figure 25-20: SPI_RWCR Register Diagram

Table 25-15: SPI_RWCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Received Word Count Reload. The SPI_RWCR.VALUE bits hold the receive transfer word count reload value.

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. 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.

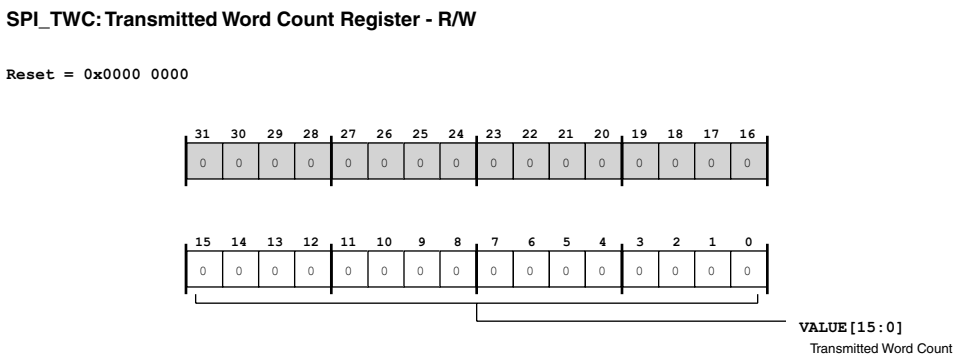


Figure 25-21: SPI_TWC Register Diagram

Table 25-16: SPI_TWC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Transmitted Word Count. The SPI_TWC.VALUE 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.

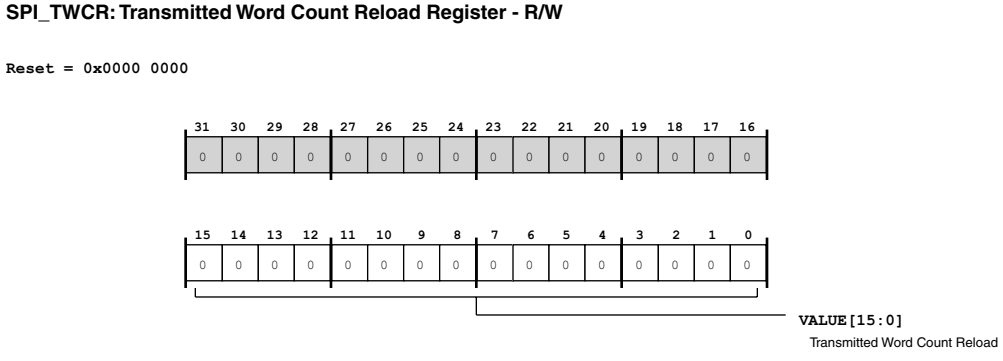


Figure 25-22: SPI_TWCR Register Diagram

Table 25-17: SPI_TWCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Transmitted Word Count Reload. The SPI_TWCR.VALUE bits hold the transmit transfer word count reload value.

Interrupt Mask Register

The SPI_IMSK register unmasks (enables) or masks (disables) SPI interrupts. When a condition is indicated by a bit in the SPI_STAT register and the corresponding interrupt is unmasked in SPI_IMSK, the SPI latches the interrupt's bit in the SPI_ILAT register, queuing the interrupt for service.

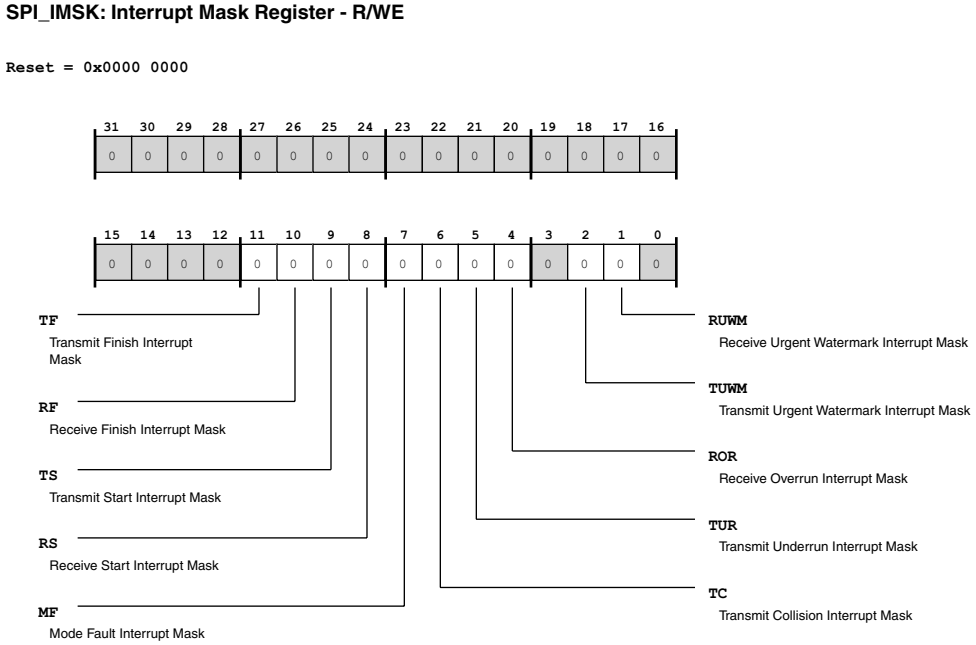


Figure 25-23: SPI_IMSK Register Diagram

Table 25-18: SPI_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/NW)	TF	Transmit Finish Interrupt Mask.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
10 (R/NW)	RF	Receive Finish Interrupt Mask.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
9 (R/NW)	TS	Transmit Start Interrupt Mask.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt

Table 25-18: SPI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	RS	Receive Start Interrupt Mask.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
7 (R/NW)	MF	Mode Fault Interrupt Mask.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
6 (R/NW)	TC	Transmit Collision Interrupt Mask.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
5 (R/NW)	TUR	Transmit Underrun Interrupt Mask.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
4 (R/NW)	ROR	Receive Overrun Interrupt Mask.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
2 (R/NW)	TUWM	Transmit Urgent Watermark Interrupt Mask.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
1 (R/NW)	RUWM	Receive Urgent Watermark Interrupt Mask.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt

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 SPI_IMSK_CLR to clear the corresponding bit in the SPI_IMSK register.

SPI_IMSK_CLR: Interrupt Mask Clear Register - R/W

Reset = 0x0000 0000

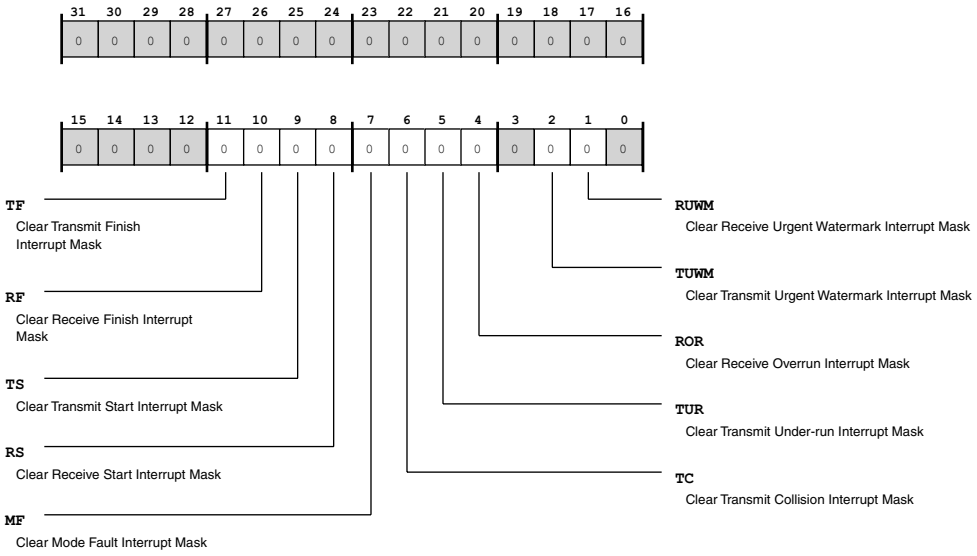


Figure 25-24: SPI_IMSK_CLR Register Diagram

Table 25-19: SPI_IMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/W1C)	TF	Clear Transmit Finish Interrupt Mask.
10 (R/W1C)	RF	Clear Receive Finish Interrupt Mask.
9 (R/W1C)	TS	Clear Transmit Start Interrupt Mask.
8 (R/W1C)	RS	Clear Receive Start Interrupt Mask.
7 (R/W1C)	MF	Clear Mode Fault Interrupt Mask.
6 (R/W1C)	TC	Clear Transmit Collision Interrupt Mask.
5 (R/W1C)	TUR	Clear Transmit Under-run Interrupt Mask.
4 (R/W1C)	ROR	Clear Receive Overrun Interrupt Mask.

Table 25-19: SPI_IMSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	TUWM	Clear Transmit Urgent Watermark Interrupt Mask.
1 (R/W1C)	RUWM	Clear Receive Urgent Watermark Interrupt Mask.

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 SPI_IMSK_SET to set the corresponding bit in the SPI_IMSK register.

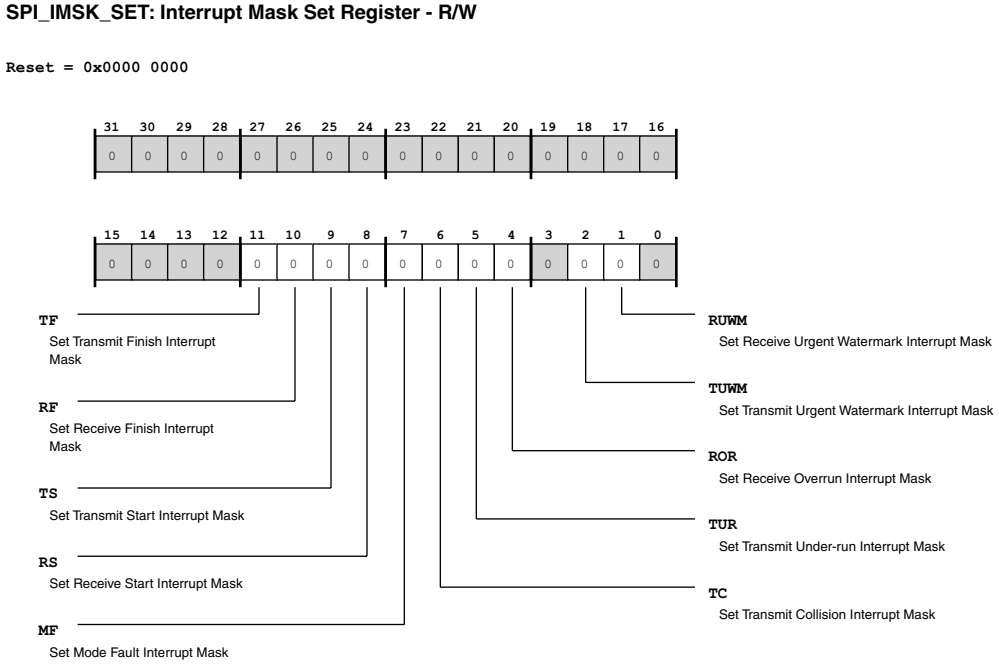


Figure 25-25: SPI_IMSK_SET Register Diagram

Table 25-20: SPI_IMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/W1S)	TF	Set Transmit Finish Interrupt Mask.
10 (R/W1S)	RF	Set Receive Finish Interrupt Mask.

Table 25-20: SPI_IMSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/W1S)	TS	Set Transmit Start Interrupt Mask.
8 (R/W1S)	RS	Set Receive Start Interrupt Mask.
7 (R/W1S)	MF	Set Mode Fault Interrupt Mask.
6 (R/W1S)	TC	Set Transmit Collision Interrupt Mask.
5 (R/W1S)	TUR	Set Transmit Under-run Interrupt Mask.
4 (R/W1S)	ROR	Set Receive Overrun Interrupt Mask.
2 (R/W1S)	TUWM	Set Transmit Urgent Watermark Interrupt Mask.
1 (R/W1S)	RUWM	Set Receive Urgent Watermark Interrupt Mask.

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 is latched into the corresponding bit in the SPI_ILAT register.

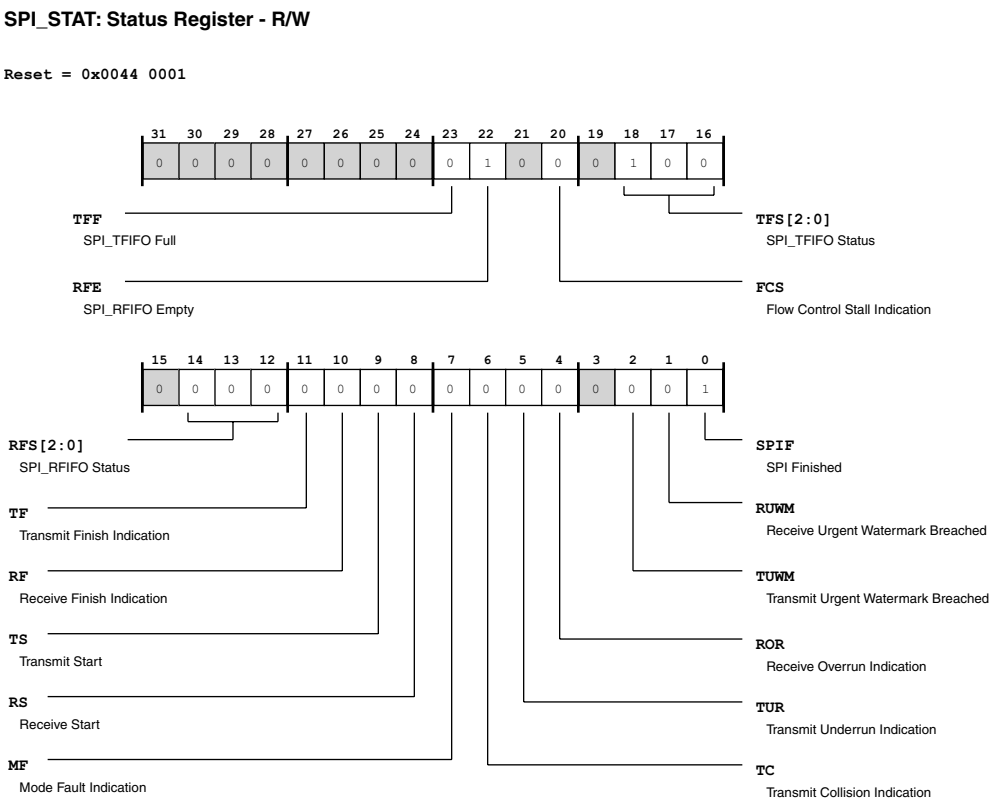


Figure 25-26: SPI_STAT Register Diagram

Table 25-21: SPI_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23 (R/NW)	TFF	SPI_TFIFO Full. The SPI_STAT.TFF bit indicates whether the SPI_TFIFO is full or not full.	
		0	Not full Tx FIFO
		1	Full Tx FIFO
22 (R/NW)	RFE	SPI_RFIFO Empty. The SPI_STAT.RFE bit indicates whether the SPI_RFIFO is empty or not empty.	
		0	RX FIFO not empty
		1	RX FIFO empty

Table 25-21: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/NW)	FCS	Flow Control Stall Indication. The SPI_STAT.FCS bit indicates whether a slave has de-asserted the SPI_RDY 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 (SPI_CTL.MSTR =1 and flow control is enabled (SPI_CTL.FCEN =1).	
		0	No Stall (RDY pin asserted)
		1	Stall (RDY pin de-asserted)
18:16 (R/NW)	TFS	SPI_TFIFO Status. The SPI_STAT.TFS bits indicate the status of the SPI_TFIFO. 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
14:12 (R/NW)	RFS	SPI_RFIFO Status. The SPI_STAT.RFS bits indicate the status of the SPI_RFIFO. 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 SPI_STAT.TF bit indicates that the SPI has detected the finish of a transmit burst transfer (the SPI_TWC count decrements to zero). This condition can only occur when SPI_TXCTL.TTI and SPI_TXCTL.TWCEN are enabled.	
		0	No status
		1	Transmit finish detected

Table 25-21: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W1C)	RF	Receive Finish Indication. The SPI_STAT.RF bit indicates that the SPI has detected the finish of a receive burst transfer (the SPI_RWC count decrements to zero). This condition can only occur when SPI_RXCTL.RTI and SPI_RXCTL.RWCEN are enabled.	
		0	No status
		1	Receive finish detected
9 (R/W1C)	TS	Transmit Start. The SPI_STAT.TS bit indicates that the SPI has detected the start of a transmit burst transfer. A transmit bursts starts with the load of SPI_TWC from the SPI_TWCR. This condition can only occur when SPI_TXCTL.TTI and SPI_TXCTL.TWCEN are enabled.	
		0	No status
		1	Transmit start detected
8 (R/W1C)	RS	Receive Start. The SPI_STAT.RS bit indicates that the SPI has detected the start of a receive burst transfer. A receive bursts starts with the load of SPI_RWC from the SPI_RWCR. This condition can only occur when SPI_RXCTL.RTI and SPI_RXCTL.RWCEN are enabled.	
		0	No status
		1	Receive start detected
7 (R/W1C)	MF	Mode Fault Indication. The SPI_STAT.MF bit, when SPI is a master and SPI_CTL.PSSE 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 SPI_STAT.TC 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

Table 25-21: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1C)	TUR	Transmit Underrun Indication. The SPI_STAT.TUR bit, when the transmit FIFO (SPI_TFIFO 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 SPI_STAT.ROR bit, when the receive FIFO (SPI_RFIFO 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 SPI_STAT.TUWM bit indicates that the transmit urgent watermark (SPI_TXCTL.TUWM) has been reached. This condition is cleared when the transmit FIFO fills enough to reach the transmit regular watermark (SPI_TXCTL.TRWM).	
		0	TX Regular Watermark reached
		1	TX Urgent Watermark breached
1 (R/NW)	RUWM	Receive Urgent Watermark Breached. The SPI_STAT.RUWM bit indicates that the receive urgent watermark (SPI_RXCTL.RUWM) has been reached. This condition is cleared when the receive FIFO empties enough to reach the receive regular watermark (SPI_RXCTL.RRWM).	
		0	RX Regular Watermark reached
		1	RX Urgent Watermark breached
0 (R/NW)	SPIF	SPI Finished. The SPI_STAT.SPIF bit indicates that a single word transfer is complete.	
		0	No status
		1	Completed single-word transfer

Masked Interrupt Condition Register

The SPI_ILAT register latches interrupts, queuing the interrupts for service. When a condition is indicated by a bit in the SPI_STAT register and the corresponding interrupt is unmasked in SPI_IMSK, the SPI latches the interrupt's bit in SPI_ILAT.

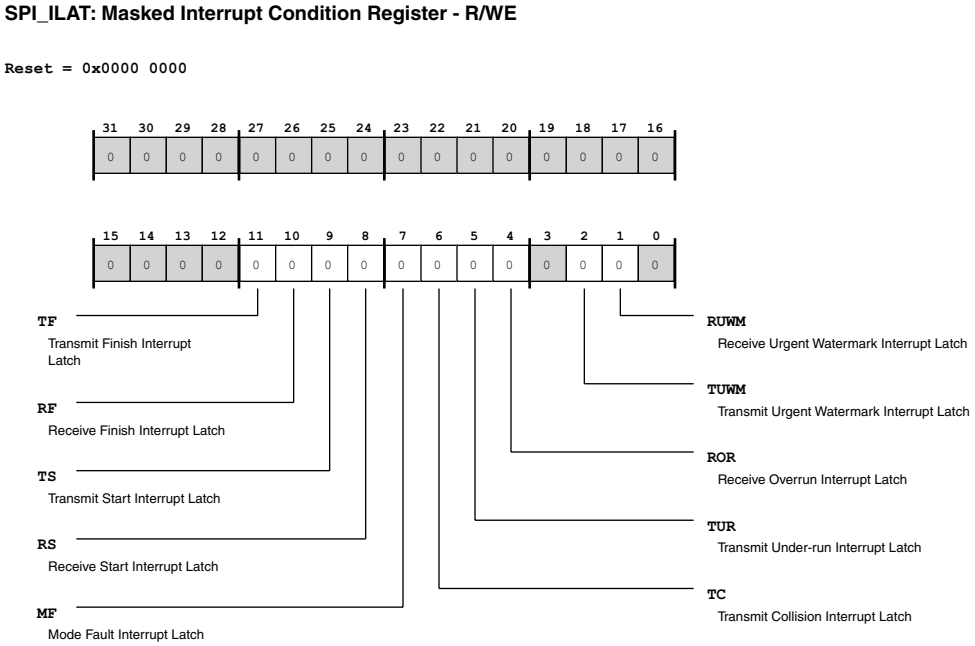


Figure 25-27: SPI_ILAT Register Diagram

Table 25-22: SPI_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/NW)	TF	Transmit Finish Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
10 (R/NW)	RF	Receive Finish Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
9 (R/NW)	TS	Transmit Start Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt

Table 25-22: SPI_ILAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	RS	Receive Start Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
7 (R/NW)	MF	Mode Fault Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
6 (R/NW)	TC	Transmit Collision Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
5 (R/NW)	TUR	Transmit Under-run Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
4 (R/NW)	ROR	Receive Overrun Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
2 (R/NW)	TUWM	Transmit Urgent Watermark Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
1 (R/NW)	RUWM	Receive Urgent Watermark Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt

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 SPI_ILAT_CLR to clear the corresponding bit in the SPI_ILAT register.

SPI_ILAT_CLR: Masked Interrupt Clear Register - R/W

Reset = 0x0000 0000

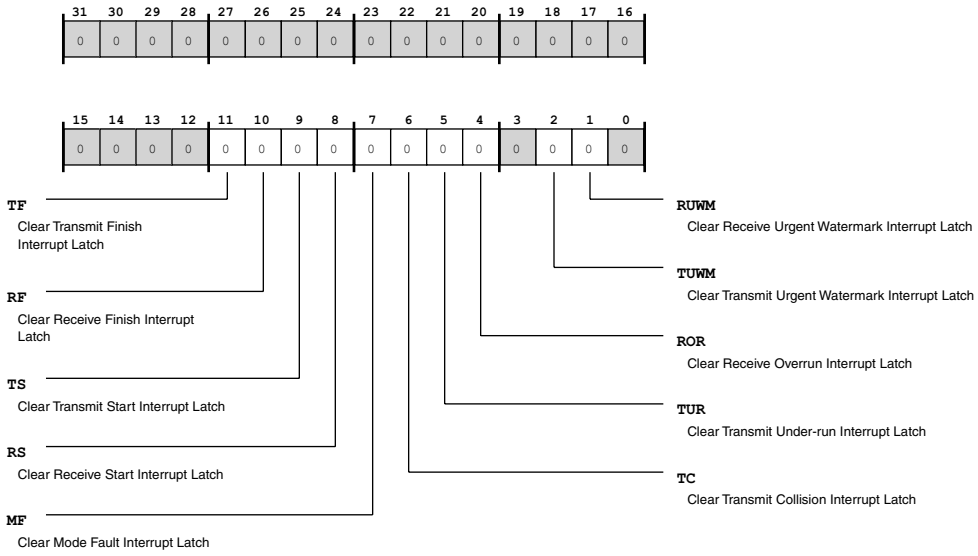


Figure 25-28: SPI_ILAT_CLR Register Diagram

Table 25-23: SPI_ILAT_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/W1C)	TF	Clear Transmit Finish Interrupt Latch.
10 (R/W1C)	RF	Clear Receive Finish Interrupt Latch.
9 (R/W1C)	TS	Clear Transmit Start Interrupt Latch.
8 (R/W1C)	RS	Clear Receive Start Interrupt Latch.
7 (R/W1C)	MF	Clear Mode Fault Interrupt Latch.
6 (R/W1C)	TC	Clear Transmit Collision Interrupt Latch.
5 (R/W1C)	TUR	Clear Transmit Under-run Interrupt Latch.
4 (R/W1C)	ROR	Clear Receive Overrun Interrupt Latch.

Table 25-23: SPI_ILAT_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/NW)	TUWM	Clear Transmit Urgent Watermark Interrupt Latch.
1 (R/NW)	RUWM	Clear Receive Urgent Watermark Interrupt Latch.

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 word size is 8-bit, or the size is 4 if word size is 16-bit, or the size is 2 if 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, and the SPI indicates this condition with receive overrun (SPI_STAT.ROR). 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.

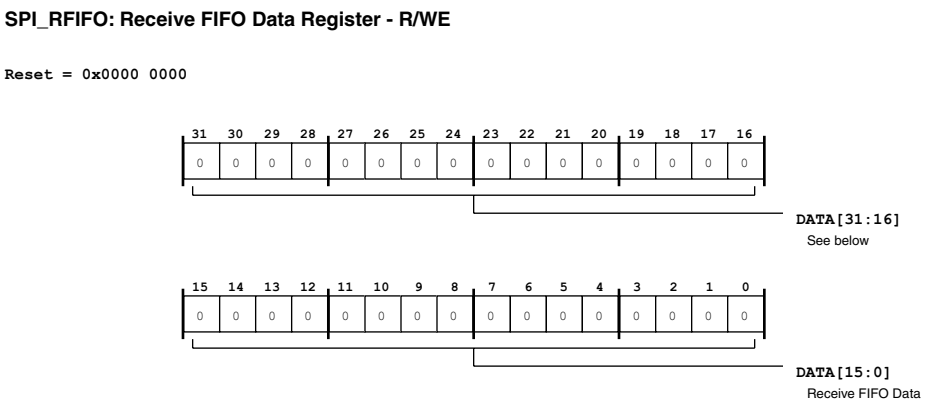


Figure 25-29: SPI_RFIFO Register Diagram

Table 25-24: SPI_RFIFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	DATA	Receive FIFO Data.

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 word size is 8-bit, or the size is 4 if word size is 16-bit, or the size is 2 if 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.

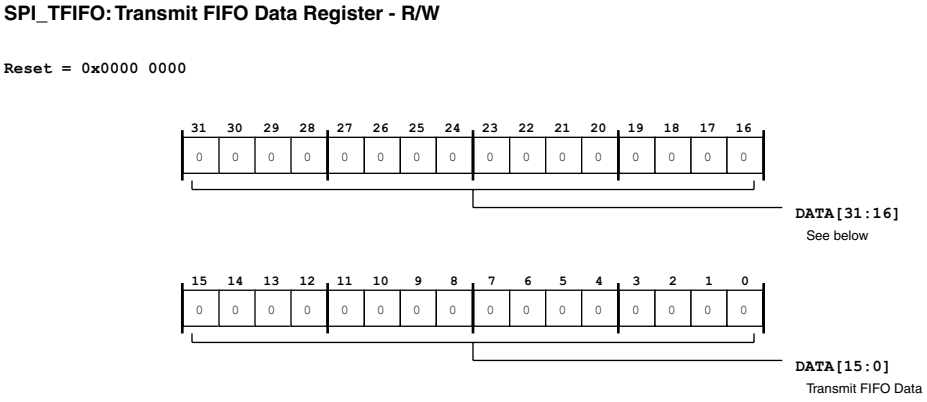


Figure 25-30: SPI_TFIFO Register Diagram

Table 25-25: SPI_TFIFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Transmit FIFO Data.

26 Serial Port (SPORT)

Unlike the SPI interface, which has been designed for SPI-compatible communication only, the serial ports (SPORTs) support a variety of serial data communication protocols. In addition, the SPORTs provide a glueless hardware interface to many industry-standard data converters and codecs. With support for high data rates and dual half-duplex data paths, the SPORT interface is a perfect choice for direct serial interconnection between two or more processors in a multiprocessor system. Many processors provide compatible serial interfaces, including DSPs from Analog Devices and other manufacturers.

The SPORT top module comprises of two half SPORTs with identical functionality. Each SPORT half can be independently configured as either a transmitter or receiver and can be coupled with the other HSPORT within the same SPORT. Further, each SPORT half provides two synchronous half-duplex data lines to double the total supported data streams.

Each SPORT half has the same capabilities and is programmed in the same way. The interface specifications of each SPORT half are shown in the following table.

Table 26-1: SPORT Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	Yes
Internal connections between SPORT halves	Yes. Only Clock and/or Frame Sync can be loopbacked internally between paired SPORT halves.
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	Yes
Transmission Simplex	Yes
Transmission Half Duplex	Yes
Transmission Full Duplex	No. The paired SPORT halves can however, be effectively used for full-duplex communication.
Access Type	
Data Buffer	Yes. Each SPORT half has its own set of control registers and data buffers.
Core Data Access	Yes

Table 26-1: SPORT Specifications (Continued)

Feature	Availability
DMA Data Access	Yes
DMA Channels	One per SPORT half
DMA Chaining	Yes
Boot Capable	No
Local Memory	No
Clock Operation	See data sheet

Features

An individual SPORT module consists of two independently configurable SPORT halves with identical functionality. These SPORT halves offer the following features.

- Two bidirectional data lines—Primary (0) and Secondary (1) per SPORT half, configurable as either transmitters or receivers. Therefore, each SPORT half can be configured for two transmitter or two receiver channels, permitting two unidirectional streams into or out of the same SPORT half. This bidirectional functionality provides greater flexibility for serial communications. Further, two SPORT halves can be combined to enable full-duplex, dual-stream communications.
- Six operation modes
    a. Standard DSP serial mode
    b. I ^{2}S mode
    c. Left-Justified mode
    d. Right-Justified Mode
    e. Multichannel Mode
    f. Packed mode
- Improved granularity for internal clock generation, allowing both even and odd SCLK to SPORT_CLK ratios. If both data lines of a SPORT half are active, it can have a maximum throughput of 2 x SPORT_CLK. The SPORTs can accept an input clock from an external source.
- Configurable rising or falling edge of the SPORT_CLK for driving or sampling data and frame sync.
- Gated clock mode support for both internal clock and external clock mode in DSP serial mode and stereo modes (Left-justified and I ^{2}S mode).
- Operates with or without a frame synchronization signal for each data word, with internally generated or externally generated frame signals, with active high or active low frame signals, and with either of two configurable pulse widths and frame signal timing.

- Status flagging and optional interrupt generation for prematurely received external frame syncs.
- External frame sync signal can be configured as level-sensitive or edge-sensitive signal.
- Serial data words between 4 and 32 bits in length, either in most significant bit (MSB) first or in least significant bit (LSB) first format. Optional sign-extension on received data.
- Optional 16-bit to 32-bit word packing when SPORT is configured as receiver and 32-bit to 16-bit word unpacking when configured as Transmitter.
- When configured as transmitter, both primary and secondary data paths can have optional compress engines enabled. Similarly, in receiver mode, both paths can have optional expand engines enabled. A-law and μ-law compression/decompression hardware companding according to G.711 specification on transmitted and received words in all operating modes.
- Status flagging and optional interrupt generation for Transmit under-run or Receive over-flow.
- Supports multichannel mode for TDM interfaces. Each SPORT half can transmit or receive data selectively from a time-division-multiplexed serial bit stream on 128 contiguous channels from a stream of up to 1024 total channels. This mode can be useful for H.100/H.110 and other telephony interfaces as a network communication scheme for multiple processors.
- Performs interrupt-driven, single word transfers to and from on-chip or off-chip memory under processor control.
- Dedicated DMA channel for each SPORT half. This DMA is common for both data lines and can be configured for multiple work units such as auto-buffer based (for a repeated, identical range of transfers) or descriptor-based (individual or repeated ranges of transfers with differing DMA parameters).
- SPORT DMA's can be programmed to accept the incoming trigger when configured as Trigger Slave and are capable of generating outgoing trigger as well.
- When using DMA in transmit mode, a Transfer Finish Interrupt (TFI) can be used to make sure that the last word of the transfer has been shifted out of the transmit shift register.
- SPMUX, a local multiplexing block integrated between the SPORT and the PinMux logic, provides the ability to route and share the clocks and/or frame sync between the SPORT halves of the SPORT module. The internal routing helps to reduce the total number of processor pins required for the interface. This is especially efficient when a SPORT is used for full-duplex data transfers.
- Interface with the ADC Control Module (ACM) block. This allows the frame sync and clock signals generated from ACM block to be routed internally to one of the SPORT halves of SPORT1.

Signal Descriptions

Each SPORT half module has five dedicated pins, as described in the following table. The actual pin name varies with different SPORT halves. The individual SPORT half does not share any of its pins across the pair. However, if required, clock and frame sync signals can be interconnected between the SPORT half pair, as explained in SPORT pin MUX section.

All the SPORT signals are available on the GPIO pins and are multiplexed with other peripheral signals. By default, these pins are in GPIO mode. To enable the pins for SPORT functionality, the appropriate bits must be set in the PORTx_FER and PORTx_MUX registers. It is advised to configure PORTx_MUX register before PORTx_FER.

Table 26-2: SPORT Pin Descriptions

Internal Node	Direction	Description
SPORT_CLK	I/O	Transmit/Receive Serial Clock. Data and Frame Sync are driven/sampled with respect to this clock. This signal can be either internally or externally generated.
SPORT_FS	I/O	Transmit/Receive Frame Sync. The frame sync pulse initiates shifting of serial data. This signal is either generated internally or externally.
SPORT_D0	I/O	Transmit/receive Primary Data channel. Bidirectional data pin. This signal can be configured as an output to transmit serial data, or as an input to receive serial data.
SPORT_D1	I/O	Transmit/receive secondary Data channel. Bidirectional data pin. This signal can be configured as an output to transmit serial data, or as an input to receive serial data.
SPORT_TDV	O	Multichannel Transmit Data Valid. This signal is only active if SPORT is configured in multichannel transmit mode. The signal is asserted during enabled slots based on the channel selection registers (SPORT_CS0_A through SPORT_CS3_B).

The signals are known as Transmit signal when the serial port is configured in transmit mode (SPORT_CTL_A.SPTRAN = 1); while are known as Receive signals when configured in receive mode (SPORT_CTL_A.SPTRAN = 0). These SPORT signals are described in the sections below.

Serial Clock

The serial port clock (SPT_ACLK) signal is considered a Receive serial clock if the transfer direction is configured as receiver; while it is considered a Transmit serial clock when configured as transmitter.

The serial clock (SPT_ACLK) is one of the control signal of serial port depending on which the data bits are shifted-in or shifted-out serially based on the direction selected. The frame sync signal is also driven (in internal frame sync mode) or sampled (in external frame sync mode) with respect to serial clock signal. The serial clock can be internally generated from processor's system clock (SCLK1) or externally provided, based on SPORT_CTL_A.ICLK bit setting. If a SPORT is configured in internal clock mode (SPORT_CTL_A.ICLK = 1), then the SPORT_DIV_A.CLKDIV field specifies the divider to generate serial port clock signal from its fundamental clock, SCLK. This divisor is a 16-bit value, allowing a wide range of serial clock rates. Use the following equation to calculate the serial clock frequency:

```
SPT_ACLK = [SCLK ÷ ( SPORT_DIV_A.CLKDIV + 1)]
```

Use the following equation to determine the value of SPORT_DIV_A.CLKDIV, given the SCLK frequency and desired serial port clock frequency:

```
SPORT_DIV_A.CLKDIV = [(SCLK ÷ SPT_ACLK) - 1]
```

This equation results in improved granularity for internal clock generation, allowing both odd and even SCLK: SPT_ACLK ratios.

It also supports 1:1 SPT_ACLK to SCLK ratio, when CLKDIV field is programmed to zero, resulting in serial port clock frequency equal to system clock. But caution must be exercised not to exceed the maximum SPT_ACLK frequency specified in the data sheet. Therefore if SCLK is greater than the data sheet limit, SPT_ACLK:SCLK ratio must be limited to 1:2. For other SCLK frequencies, this ratio can be programmed up to 1:1.

In certain operating modes, the serial port can be configured to generate gated clock which is active only for the duration of valid data. In some applications, it can be used to generate a general-purpose clock in the system. In this case the SPORT must be enabled with 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), then serial clock is a input signal making the SPORT to operate in slave mode. The SPORT_DIV_A.CLKDIV is ignored. The optional loopback capability provided by SPMUX block, allows slave SPORT to use the serial clock from the neighboring serial port.

Note that externally supplied serial clock need not be in synchronous with processor system clock. Further, the external clock can be a gated clock but it must comply the requirements described in Gated Clock Mode section. Please refer appropriate product data sheet for exact a.c. timing specifications.

Frame Sync

The serial port frame sync (SPT_AFS) signal is considered a Receive Frame Sync if the transfer direction is configured as receiver; while it is considered a Transmit Frame Sync when configured as transmitter.

Frame sync is also a control signal, generally used to determine the start of new word or frame. Upon detecting this signal, serial port starts shifting in or out the new data bits serially based on the direction selected. The frame sync signal can be internally generated from its serial clock (SPT_ACLK) or externally provided, based on the SPORT_CTL_A.IFS bit setting.

If SPORT is configured for internal frame sync mode (SPORT_CTL_A.IFS = 1), then the SPORT_DIV_A.FSDIV field specifies the divider to generate SPT_AFS signal from the serial clock. This divisor is a 16-bit value, allowing a wide range of frame sync rates to initiate periodic transfers. The serial clock may be internally generated or externally supplied and it is counted equal to divisor specified before a frame sync pulse is generated. The formula for the number of cycles between frame sync pulses is:

Number of serial clocks between frame syncs = (SPORT_DIV_A.FSDIV + 1)

Use the following equation to determine the value of SPORT_DIV_A.FSDIV, given the serial clock frequency and desired frame sync frequency:

SPORT_DIV_A.FSDIV = [(SPT_ACLK ÷ SPT_AFS) – 1]

The frame sync is continuously active when SPORT_DIV_A.FSDIV = 0. The value of SPORT_DIV_A.FSDIV should not be less than the serial word length minus one (the value of the SPORT_CTL_A.SLEN bit field), as this may 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) serial clocks.

If a SPORT is configured in external frame sync mode (SPORT_CTL_A.IFS = 0), then SPT_AFS is a input signal and the SPORT_DIV_A.FSDIV field of the SPORT_DIV_A register is ignored. By default, this external signal is level-sensitive, but can be configured as an edge-sensitive signal by setting 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 product specific data sheet.

The serial port can be used as a counter for dividing an external clock to generate periodic pulses or periodic interrupts. The SPORT must be enabled with appropriate SPORT_DIV_A.FSDIV divisor field in external clock, internal data-independent frame sync mode.

In some of the operating modes, the serial port can be programmed to treat the frame sync signal as an optional signal by clearing the SPORT_CTL_A.FSR bit (it can be used to start the continuous transfers and subsequently ignored). Characteristics of the frame sync depend on the settings in the SPORT control registers and the SPORT's operating mode. For more information, refer to the SPORT control register bits and respective operating mode details.

Data Signals

Each SPORT half has two bi-directional data lines known as the primary transmit or receive data channel (SPT_AD0) and the secondary transmit or receive data channel (SPT_AD1). Both the data lines can be configured as either transmitters or receivers using the SPORT_CTL_A.SPTRAN bit, permitting dual unidirectional data streams to increase the data throughput of the serial port.

Both data lines can be individually enabled or disabled using the SPORT_CTL_A.SPENPRI and the SPORT_CTL_A.SPENSEC bits. However, if using both, it is advised to enable or disable them concurrently. They do not behave as totally separate SPORTs; rather, they operate in a synchronous manner (sharing a clock and frame sync) but on separate data paths. All of the SPORT control settings are common for both channels but the single DMA channel per serial half serves both primary and secondary data channels. Also, both primary and secondary channels have separate data buffers, shift registers and optional companding logic in their path.

When a serial port is configured in multichannel transmit mode, the data pins three-states during inactive channel slots. This allows multiple serial port transmitters to operate on the same bus with different active channels.

See the Architecture section for more details about data transfer operation.

Transmit Data Valid Signal

The Transmit Data Valid (SPT_ATDV) signal is available only in multichannel modes (including packed mode) of a SPORT configured as a transmitter. This signal is active during transmission of enabled multichannel slots and remains in an inactive state for the disabled channels. In other words, the SPT_ATDV signal is active whenever a serial port is driving the data pins and stays inactive when the data pins three-states. Therefore the SPT_ATDV signal can serve as an output-enable signal for the data transmit pin.

Functional Description

The following section provides general information about functionality of the serial ports of processors.

- *Architectural Concepts*
- *Data Types and Companding*
- *Transmit Path*
- *Receive Path*

ADSP-BF60x 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 glue-less 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 govern SPORT operations. For more information on SPORT functionality, see the SPORT register descriptions.

Table 26-3: ADSP-BF60x SPORT Register List

Name	Description
SPORT_CTL_A	Half SPORT 'A' Control Register
SPORT_DIV_A	Half SPORT 'A' Divisor Register
SPORT_MCTL_A	Half SPORT 'A' Multi-channel Control Register
SPORT_CS0_A	Half SPORT 'A' Multi-channel 0-31 Select Register
SPORT_CS1_A	Half SPORT 'A' Multi-channel 32-63 Select Register
SPORT_CS2_A	Half SPORT 'A' Multi-channel 64-95 Select Register
SPORT_CS3_A	Half SPORT 'A' Multi-channel 96-127 Select Register

Table 26-3: ADSP-BF60x SPORT Register List (Continued)

Name	Description
SPORT_ERR_A	Half SPORT 'A' Error Register
SPORT_MSTAT_A	Half SPORT 'A' Multi-channel Status Register
SPORT_CTL2_A	Half SPORT 'A' Control 2 Register
SPORT_TXPRI_A	Half SPORT 'A' Tx Buffer (Primary) Register
SPORT_RXPRI_A	Half SPORT 'A' Rx Buffer (Primary) Register
SPORT_TXSEC_A	Half SPORT 'A' Tx Buffer (Secondary) Register
SPORT_RXSEC_A	Half SPORT 'A' Rx Buffer (Secondary) Register
SPORT_CTL_B	Half SPORT 'B' Control Register
SPORT_DIV_B	Half SPORT 'B' Divisor Register
SPORT_MCTL_B	Half SPORT 'B' Multi-channel Control Register
SPORT_CS0_B	Half SPORT 'B' Multi-channel 0-31 Select Register
SPORT_CS1_B	Half SPORT 'B' Multi-channel 32-63 Select Register
SPORT_CS2_B	Half SPORT 'B' Multichannel 64-95 Select Register
SPORT_CS3_B	Half SPORT 'B' Multichannel 96-127 Select Register
SPORT_ERR_B	Half SPORT 'B' Error Register
SPORT_MSTAT_B	Half SPORT 'B' Multi-channel Status Register
SPORT_CTL2_B	Half SPORT 'B' Control 2 Register
SPORT_TXPRI_B	Half SPORT 'B' Tx Buffer (Primary) Register
SPORT_RXPRI_B	Half SPORT 'B' Rx Buffer (Primary) Register
SPORT_TXSEC_B	Half SPORT 'B' Tx Buffer (Secondary) Register
SPORT_RXSEC_B	Half SPORT 'B' Rx Buffer (Secondary) Register

ADSP-BF60x SPORT Interrupt List

Table 26-4: ADSP-BF60x SPORT Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
SPORT0 Channel A DMA	43	0	LEVEL
SPORT0 Channel A Status	44		LEVEL
SPORT0 Channel B DMA	45	1	LEVEL
SPORT0 Channel B Status	46		LEVEL
SPORT1 Channel A DMA	47	2	LEVEL
SPORT1 Channel A Status	48		LEVEL
SPORT1 Channel B DMA	49	3	LEVEL
SPORT1 Channel B Status	50		LEVEL
SPORT2 Channel A DMA	51	4	LEVEL
SPORT2 Channel A Status	52		LEVEL
SPORT2 Channel B DMA	53	5	LEVEL
SPORT2 Channel B Status	54		LEVEL

ADSP-BF60x SPORT Trigger List

Table 26-5: ADSP-BF60x SPORT Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
SPORT0 Channel A DMA	20	PULSE/EDGE
SPORT0 Channel B DMA	21	PULSE/EDGE
SPORT1 Channel A DMA	22	PULSE/EDGE
SPORT1 Channel B DMA	23	PULSE/EDGE
SPORT2 Channel A DMA	24	PULSE/EDGE
SPORT2 Channel B DMA	25	PULSE/EDGE

Table 26-6: ADSP-BF60x SPORT Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
SPORT0 Channel A DMA	20	
SPORT0 Channel B DMA	21	

Table 26-6: ADSP-BF60x SPORT Trigger List Trigger Slaves (Continued)

Description	Trigger ID	Sensitivity
SPORT1 Channel A DMA	22	
SPORT1 Channel B DMA	23	
SPORT2 Channel A DMA	24	
SPORT2 Channel B DMA	25	

ADSP-BF60x SPORT DMA List

Table 26-7: ADSP-BF60x SPORT DMA List DMA Channel List

Description	DMA Channel
SPORT0 Channel A DMA	DMA0
SPORT0 Channel B DMA	DMA1
SPORT1 Channel A DMA	DMA2
SPORT1 Channel B DMA	DMA3
SPORT2 Channel A DMA	DMA4
SPORT2 Channel B DMA	DMA5

Block Diagram

The serial port is configured in transmit mode, if `SPORT_CTL_A.SPTRAN` control bit is set. If this bit is cleared, serial port configures in receive mode. If `SPORT_CTL_A.SPENPRI` control bit is set, then serial port activates primary transmit/receive path. If `SPORT_CTL_A.SPENSEC` control bit is set, then it activates secondary transmit/receive path. Both data channels can be enabled to allow synchronous dual-stream communication. Each path optionally supports Hardware companding or expanding as well. Once a path is activated, data is shifted in response to a frame sync at the rate of serial clock. Inactive data buffers are not used and should not be accessed. An application program must use the appropriate data buffers.

These serial ports are not UARTs and cannot communicate with an RS-232 device or any other asynchronous communications protocol. One way to implement RS-232 compatible communication with the processor is to use two of the FLAG pins as asynchronous data receive and transmit signals.

The following figure shows a detailed block diagram of a SPORT half side.

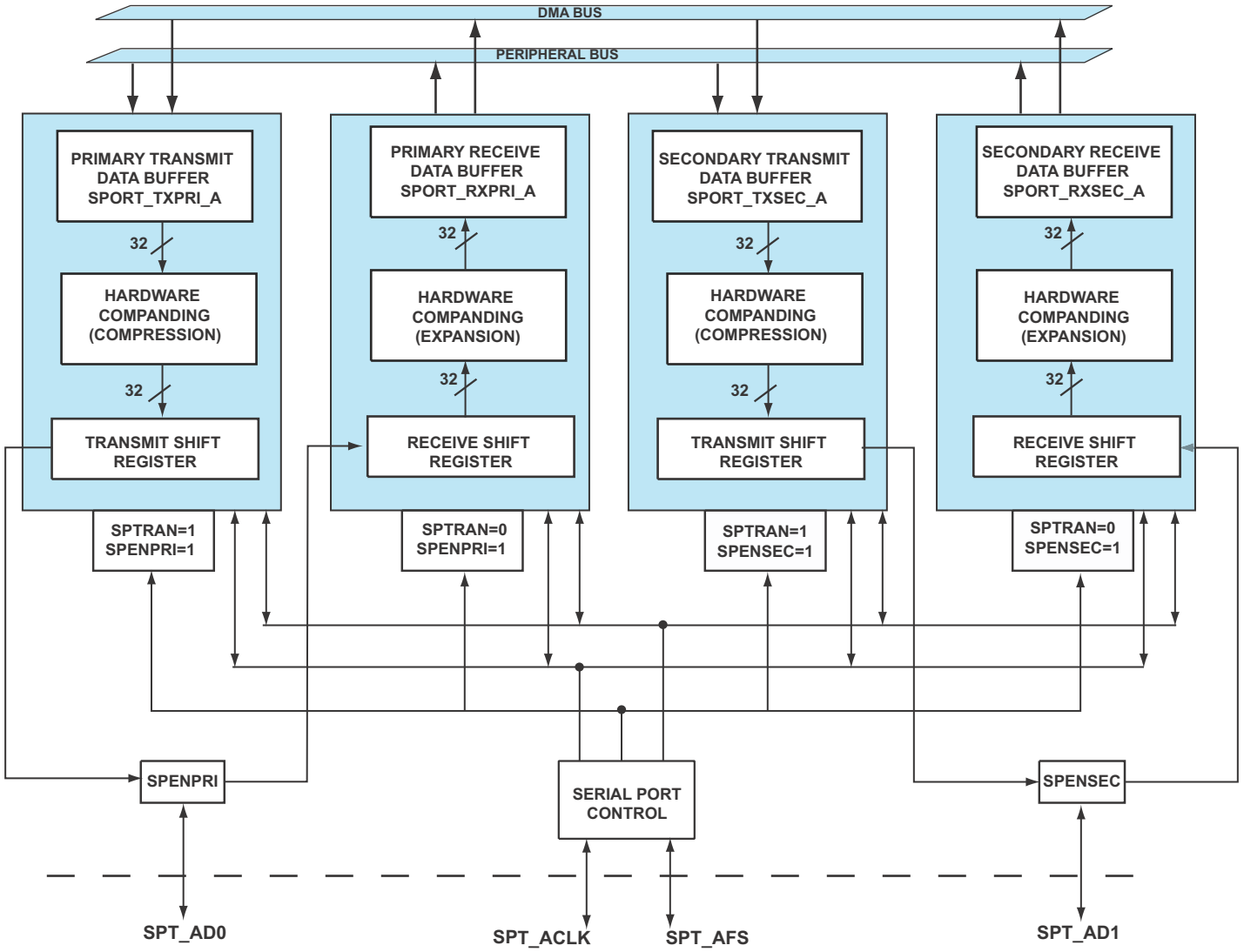


Figure 26-1: Serial Port Block Diagram

Architectural Concepts

Each SPORT module consists of two separate blocks, known as half-SPORT (HSPORT) A and B, with identical functionality. These blocks can be independently configurable as either transmitter or receiver; and optionally coupled together internally in a limited way. Each HSPORT also supports two synchronous bidirectional data paths, referred as primary (`D0`) and secondary (`D1`) data lines, as shown in the following figure.

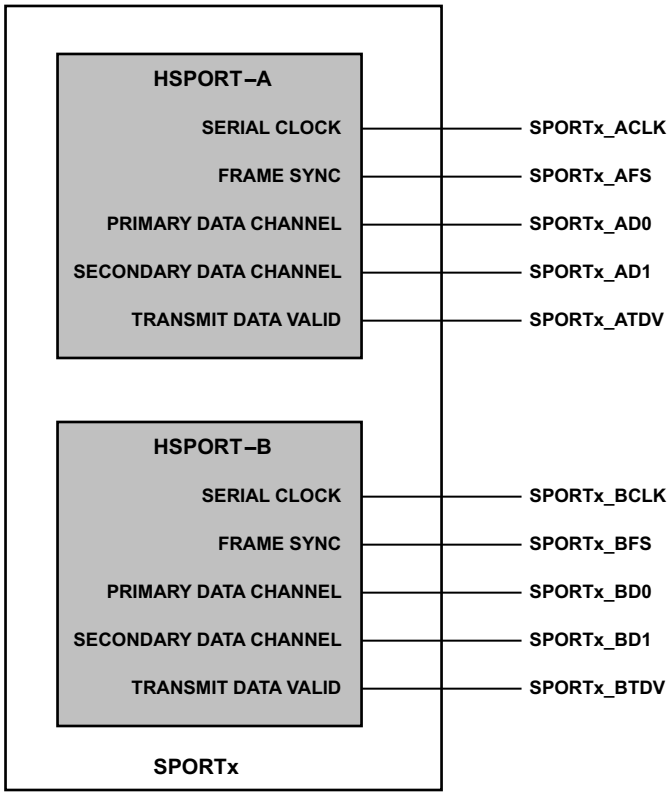


Figure 26-2: Top Level SPORT Diagram

Each HSPORT can be configured as either transmitter or receiver, according to which the pair of data signals transmit or receive data bits synchronously. The SPORT_CTL_A.SPTRAN bit controls the direction for both data paths of the HSPORT. Each HSPORT has its own set of control registers and data buffers grouped per SPORT module. The dual data signals of each HSPORT cannot transmit and receive the data simultaneously for full-duplex operation. Two HSPORTs must be combined to achieve full-duplex operation.

Serial communications are synchronized to the serial clock signal. Every data bit must be accompanied by a clock pulse. Each serial port can internally generate its own serial clock signal from the processor's system clock using the divisor field of the SPORT_DIV_A.CLKDIV bit field. If programmed, serial ports can also operate in external clock mode. Both primary and secondary data channels shift data based on SPORT_CLK rate and the SPORT_CTL_A.CKRE bit

In addition to the serial clock signal, data may be signaled by a frame synchronization signal. The framing signal can occur at the beginning of an individual word or at the beginning of a block of words. The configuration of frame sync signal depends upon the type of serial device connected to the processor. Each serial port can generate its own frame sync signal (SPORT_FS) depending on the bit settings of SPORT control register. An internally generated frame sync is derived from the SPORT clock using the divisor field of the SPORT_DIV_A.FSDIV bit field. Serial ports can also accept external SPORT_FS signal. Both primary and secondary data paths starts shifting data after detecting a valid frame sync signal according to control bit settings and operating mode of serial port. A variety of serial data communication protocols can be

emulated according to the frame sync format. All frame sync options are available whether the signal is generated internally or externally generated.

Multiplexer Logic

There is a local muxing block, known as SPMUX, that is integrated between the SPORT and the PinMux logic of processor. It allows flexibility to route and share the clock and frame sync signals between the SPORT half pair of a SPORT. This is where the two independent SPORT halves of a SPORT can be coupled together. This feature can be used to reduce the total number of pins for the interface and is considered to be efficient when the SPORT half pair is used for full-duplex operation.

The SPORT_CTL2_A register is used to configure this loopback feature. The control bits of this register are as described in the "Register Descriptions" section of this chapter.

The multiplexing depends on the SPORT_CTL_A.IFS and SPORT_CTL_A.ICLK bit settings and is controlled further by the SPORT_CTL2_A.CKMUXSEL and SPORT_CTL2_A.FSMUXSEL bit settings of the SPORT half pair. The following two tables show the valid combinations for the SPORT_CTL_A.IFS, SPORT_CTL_A.ICLK, SPORT_CTL2_A.CKMUXSEL and SPORT_CTL2_A.FSMUXSEL bit settings. All other settings are considered to be illegal. The illegal settings, however, are not checked or prevented by hardware. Programs should ensure that only legal combinations are used.

The table's Routing column uses the following abbreviations.

- HSx_FI = Frame sync input. It can be provided by external device or by the neighboring SPORT half.
- HSx_FO = Frame sync output. When SPORT is configured in internal frame sync mode.
- SPx_FS = signal appearing on frame sync pin of the SPORT half.

NOTE: In the tables, the Half-SPORT pair of a SPORT, A and B, are referred as HS0 and HS1 and are applicable for all SPORTs.

Table 26-8: Frame Sync Combinations

FS Combination ID	HS0_IFS	HS1_IFS	FS0MUX	FS1MUX	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
5	0	0	1	0	HS0_FI ≤ SP1_FS; HS1_FI ≤ SP1_FS
6	0	1	1	0	HS0_FI ≤ HS1_FO ≥ SP1_FS
7	0	0	0	1	HS1_FI ≤ SP0_FS; HS0_FI ≤ SP0_FS

Table 26-8: Frame Sync Combinations (Continued)

FS Combination ID	HS0_IFS	HS1_IFS	FS0MUX	FS1MUX	Routing
8	1	0	0	1	HS1_FI ≤ HS0_FO ≥ SP0_FS

The table's Routing column uses the following abbreviations.

- HSx_CI = serial clock input. It can be provided by external device or by neighboring SPORT half.
- HSx_CO = serial clock output. When SPORT is configured in internal clock mode.
- SPx_CLK = signal appearing on serial clock pin of the SPORT half.

Table 26-9: Clock Combinations

CLK Combination ID	HS0_ICLK	HS1_ICLK	CK0MUX	CK1MUX	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
13	0	0	1	0	HS0_CI ≤ SP1_CLK; HS1_CI ≤ SP1_CLK
14	0	1	1	0	HS1_CI ≤ HS1_CO ≥ SP1_CLK
15	0	0	0	1	HS1_CI ≤ SP0_CLK; HS0_CI ≤ SP0_CLK
16	1	0	0	1	HS1_CI ≤ HS0_FO ≥ SP0_CLK

The following additional points on these combination settings should be noted.

- FS IDs 1–4 are supported with all CLK IDs 9–16.
- FS ID 5 is only supported with CLK ID 13 and vice-versa.
- FS ID 6 is only supported with CLK ID 14 and vice-versa.
- FS ID 7 is only supported with CLK ID 15 and vice-versa.
- FS ID 8 is only supported with CLK ID 16 and vice-versa.
- CLK IDs 9–12 are supported with all FS IDs 1–8.

NOTE: From these tables, one can note that a SPORT half can import serial clock signal from paired HSPORT, only when it is configured in external clock mode similarly, it can import frame sync signal, only when it is configured in external frame sync mode. The neighboring SPORT may be master (generates it's own serial clock or frame sync signal) or slave (accepting external clock or

external frame sync). It can be also noticed that, SPORT_CTL2 register programming is required only at the acceptor SPORT half side; and not required at the donor SPORT half side to enable this sharing.

Note that Polarity bits such as SPORT_CTL_A.CKRE and SPORT_CTL_A.LFS should have identical settings when using muxing between two SPORT halves.

NOTE: There are some limitations on use of SPMUX logic when a SPORT is interfaced with an ACM module of the processor. Refer to the ACM chapter for more details.

Data Types and Companding

The Data Type select field SPORT_CTL_A.DTYPE bit specifies one of the four data formats supported by serial ports. These formats can be used in any of the operating mode of serial port.

Table 26-10: Data Type Bit Field Settings

DTYPE field	SPORT Receiver	SPORT Transmitter
00	Right-Justify, zero-fill unused MSB's	Normal Operation
01	Right-Justify, sign-extend unused MSB's	Reserved
10	Expand using u-law	Compress using u-law
11	Expand using A-law	Compress using A-law

These formats are applied 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; as when configured as transmitter, only the significant bits are transmitted (as 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/receive path. Companding (compressing or expanding) is the process of logarithmically encoding and decoding data to minimize the number of bits to be sent. The processor's SPORTs support the two most widely used companding algorithms, A-law and μ-law, which is 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 causes it's content compressed to eight bits (zero filled to the width of the transmit word) according to algorithm selected. Similarly, if configured in receive mode, the received 8-bits in the receive data buffers are expanded in right-justified, zero fill format as per the algorithm selected. If companding is enabled in multichannel mode, it is applied 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 may not work correctly. Also, if the data value is greater than 13-bit A-law or 14-bit μ-law maximum, it is automatically compressed to the maximum value.

NOTE: The processor companding logic supports in-place companding feature. So, companding can be used as debug feature without enabling SPORT. See 'Companding as a Function' section for more details.

Companding as a Function

Since the values in the transmit and receive buffers are actually companded in place, the companding hardware can be used without transmitting (or receiving) any data, for example 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 serial port as 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 for two system clock cycles. Any instruction not accessing the transmit buffer can be used to cause this delay. This allows the serial port companding hardware to reload the transmit buffer with the companded value.
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. When expanding data in this way, set the appropriate serial word length (SPORT_CTL_A.SLEN) bit.

Transmit Path

The SPORT_CTL_A.SPTRAN control bit, when set, configures the SPORT in transmit mode. It then enables primary and/or secondary transmit paths, based on the SPORT_CTL_A.SPENPRI and SPORT_CTL_A.SPENSEC bit settings. Both data paths, primary and secondary, are separate but identical and include Transmit Data Buffer, optional companding logic and a Transmit Shift Register.

The data buffer on primary transmit path is known as Primary Transmit Data Buffer, or SPORT_TXPRI_A; while the one on secondary transmit path is known as Secondary Transmit Data Buffer, or SPORT_TXSEC_A. The transmit data buffer and output shift register forms a FIFO type of structure. When packing is disabled (SPORT_CTL_A.PACK = 0), serial port can hold as many as 3 data; while if packing is enabled (SPORT_CTL_A.PACK = 1), it can hold 2 packed data at any given time.

The data to be transmitted on primary and/or 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 peripheral bus or in DMA mode through DMA bus. The inactive data buffer must not be accessed. This data is optionally compressed in hardware according to selected algorithm and then automatically transferred to transmit shift register. The shift register, clocked by SPT_ACLK signal, then serially shifts out

this data on SPT_AD0 or SPT_AD1 pins, synchronously. If framing signal is used, the SPT_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, it is required to interleave the data of these channels properly.

When SPORT is configured in non-multichannel mode as transmitter, the enabled SPORT data pins SPT_AD0 and/or SPT_AD1 are always driven. When a SPORT channel is enabled, data from Transmit Data Buffer is loaded into Transmit Shift register. The shift register then immediately latches the first bit of data (either LSB or MSB based on the SPORT_CTL_A.LSBF bit setting) and not with respect to frame sync. Similarly, if frame sync duration is greater than serial word length, then during inactive serial clock cycles (clock cycles after data transmission in the current frame), the data pins drives first bit of next word to be transmitted which is loaded into shift register. This does not cause any problem at the receiver end, as it starts sampling the data pin only after detecting a valid frame sync. In multichannel mode, data pin always three-states during inactive channel slots.

The serial port provides status of transmit data buffers and also error detection logic for transmit errors such as under-run. Please see the "Error Detection" section for more details.

When a serial port is configured in transmit mode, the receive paths (and hence the Receive Data Buffers and Receive Shift registers on those paths) are deactivated and do not respond to serial clock or frame sync signals. So, reading from an empty Receive Data Buffer may cause core to hang indefinitely.

Receive Path

The SPORT_CTL_A.SPTRAN bit, when cleared, configures the SPORT in receive mode. It then enables primary and/or secondary receive paths, based on the SPORT_CTL_A.SPENPRI and SPORT_CTL_A.SPENSEC bit settings. Both data paths, primary and secondary, are separate but identical and include a Receive Shift Register, optional companding logic and a Receive Data Buffer.

The data buffer on primary receive path is known as Primary Receive Data Buffer, or SPORT_RXPRI_A; while the one on secondary receive path is known as Secondary receive Data Buffer, or SPORT_RXSEC_A. The receive paths act like a 3-word deep (32-bit) FIFO because they have two data registers plus an input shift register.

Upon enabling the serial port data channels, the input Shift register shifts in data bits on the SPT_AD0 and/or SPT_AD1 pins, synchronous to the receive clock signal. If framing signal is used, the SPT_AFS signal indicates the beginning of the serial word being received. When an entire word is shifted in on the primary and secondary channels, the data is optionally expanded in hardware according to selected algorithm and then automatically transferred to SPORT_RXPRI_A and SPORT_RXSEC_A.

The Receive Data Buffers can be read in core mode through peripheral bus or in DMA mode through DMA bus. When DMA mode is used a single DMA reads the data buffers of enabled channels (primary and or secondary). When using both channels, it is required to de-interleave the data of these channels properly. The serial port provides the status of Receive Data buffers and also error detection logic for receive errors such as overflow. See the “Error Detection” section for more details.

When a serial port is configured in receive mode, the transmit paths (and the Transmit Data Buffers and Transmit Shift registers on those paths) are deactivated and do not respond to serial clock or frame sync signals. Therefore programs must not try to access them.

Sampling Edge

The serial port uses two control signals to sample or drive the serial data.

1. Serial clock (SPT_ACLK) applies the bit clock for each serial data
2. Frame sync (SPT_AFS) divides the incoming data stream into frames.

These control signals can be internally generated or externally provided, determined by theSPORT_CTL_A.ICLK and SPORT_CTL_A.IFS bit settings.

Data and frame syncs can be sampled on the rising or falling edges of the serial port clock signals. TheSPORT_CTL_A.CKRE bit controls the sampling edge. By default, when SPORT_CTL_A.CKRE = 0, the processor selects the falling edge of SPT_ACLK signal for sampling receive data and external frame sync. The receive data and frame sync are sampled on the rising edge of SPT_ACLK whenSPORT_CTL_A.CKRE = 1.

Note that transmit data and internal frame sync signals are driven (change their state) on the serial clock edge that is not selected. By default, (SPORT_CTL_A.CKRE = 0) the SPORTs drive data and frame sync signals on the rising edge of the SPT_ACLK signal and drives on falling edge when SPORT_CTL_A.CKRE = 1.

Therefore transmit and receive functions of any two serial ports connected together should always select the same value for SPORT_CTL_A.CKRE so internally-generated signals are driven on one edge and received signals are sampled on the opposite edge.

The serial port which drives serial clock and frame sync is usually called as master while the receiver of clock and frame sync is referred as slave. The following figure shows the typical SPORT signals at two sides of serial communication for SPORT_CTL_A.CKRE = 0. The SPORT configured as Transmitter also drives the serial clock and Frame sync signals as a master device.

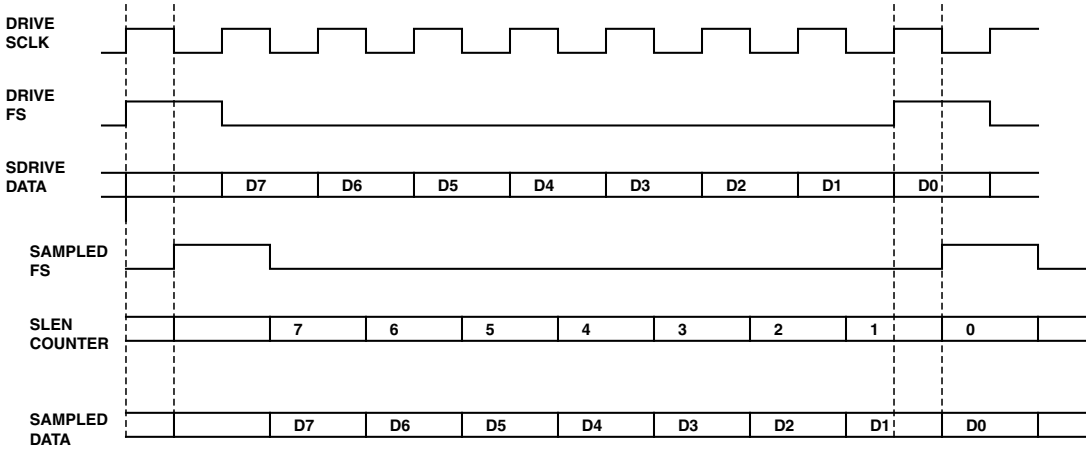


Figure 26-3: Frame Sync and Data Driven on Rising Edge

When slave samples the Frame Sync signal, the SPORT_CTL_A.SLEN word counter is reloaded to the maximum setting. Each SPT_ACLK decrements the SPORT_CTL_A.SLEN counter until the full frame is received.

Therefore, if the transmitter drives the internal frame sync and data on the rising edge of serial clock, the falling edge should be used by receiver to sample the external frame sync and data, and vice versa.

Premature Frame Sync Error Detection

A SPORT framing signal is used to synchronize transmit or receive data. In external FS mode, any frame sync received when an active frame is in progress is called premature and is invalid.

As an enhancement to processor's serial port, if a premature frame sync is received, the SPORT_ERR_A.FSERRSTAT bit is flagged to indicate this framing error. An optional error interrupt can be generated for this event by setting SPORT_ERR_A.FSERRMSK bit.

This feature is applicable in all the operating modes of serial port.

NOTE: The SPORT_ERR_A.FSERRSTAT bit is not set in the presence of uncleared underflow/overflow errors.

In stereo or I^2S mode, a premature frame sync may result in the SPORT receiving two consecutive left channels or two consecutive right channels and cause channel swapping. In the processor's serial port, swapping of channels due to a premature FS is avoided. If due to premature FS, one data gets corrupted, data will always be dropped in pairs to avoid channel swapping. The premature FS flagging in the error register will be done similarly.

As shown in the following figure, the frame sync error (which sets the error bit) is triggered when an early frame sync occurs during data transfer (transmission or reception) or for late frame sync if the period of the frame sync is smaller than the serial word length (SPORT_CTL_A.SLEN).

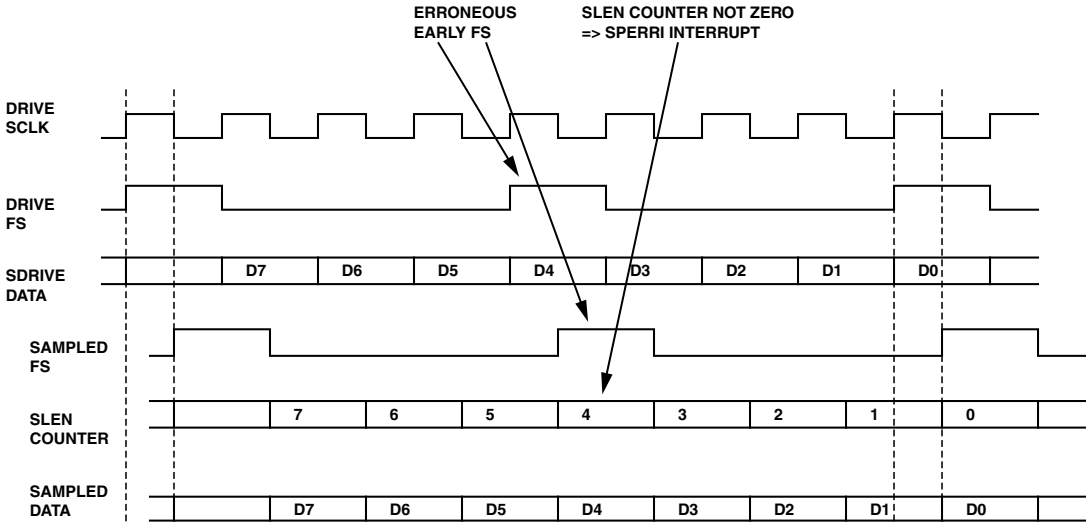


Figure 26-4: Frame Sync Error Detection

When a serial port is receiving or transmitting, its bit count is set to a word length (for example 32 bits). After each clock edge the bit count is decremented. After the word is received/ transmitted, the bit count reaches zero, and on next frame sync it is set to 32. When active transmission or reception is occurring, the bit count value is non-zero. When a frame sync with a bit count of non-zero is detected, a frame sync error occurs.

Support for Edge-Detected and Level-Sensitive Frame Syncs

The frame sync signal of the SPORT module of earlier Blackfin processors is level-sensitive. Though the level sensitive nature of frame sync will work fine in a noise free environment; but if noise corrupts the signals coming into the SPORT, then there is a chance that the start of frame sync may be missed by the internal logic either because clock getting corrupted or frame sync itself. The frame sync will be sampled from the next clock edge onwards if the frame sync happens to last for more than a bit clock period.

The following figure describes a scenario when an external frame sync signal gets corrupted due to noise and is sampled incorrectly by the slave SPORT module. Consider a frame sync, driven on the rising edge of serial clock at t_A and expected to be sampled by slave SPORT at the falling edge of serial clock at t_B. But due to the noise, the first edge of the FS is not seen by the SPORT and samples the FS only at t_C as shown in the figure. Subsequently the word length counter runs for a period equal to the SPORT_CTL_A.SLEN field of the control register and expires at t_E, instead of correctly at t_D, receiving incorrect data. Further if a new frame sync edge has come at time t_D, in level sensitive mode, the SPORT samples this framing signal again only at time t_E. So, the frame sync sampling continues to be unaligned with the external data.

The enhanced SPORT module provides an option to configure the frame sync signal as edge-sensitive signal. Edge sensitive frame sync detection looks for an edge in an external frame sync for considering it as a valid framing signal. In active-high frame syncs, the rising edge of frame sync is considered valid; while in active-low frame syncs, the falling edge is considered as valid. This optional feature can be activated by programming SPORT_CTL_A.FSED (external Frame Sync Edge select) bit.

NOTE: SPORT_CTL_A.FSED is valid only in External frame sync mode. In internal frame sync mode, this bit is a don't care.

In the example discussed above, consider frame sync configured as edge-sensitive frame sync. In this case, frame sync will not be detected at t_E because the edge of framing signal has already occurred in the previous cycle (t_D) and there is no edge to detect at t_E. So the word length counter remains idle for this frame, ignoring the incorrect data, and resumes the operation correctly at t_F when a new edge detects.

This sets the SPORT_ERR_A.FSERRSTAT bit and optionally generates a premature FS error interrupt.

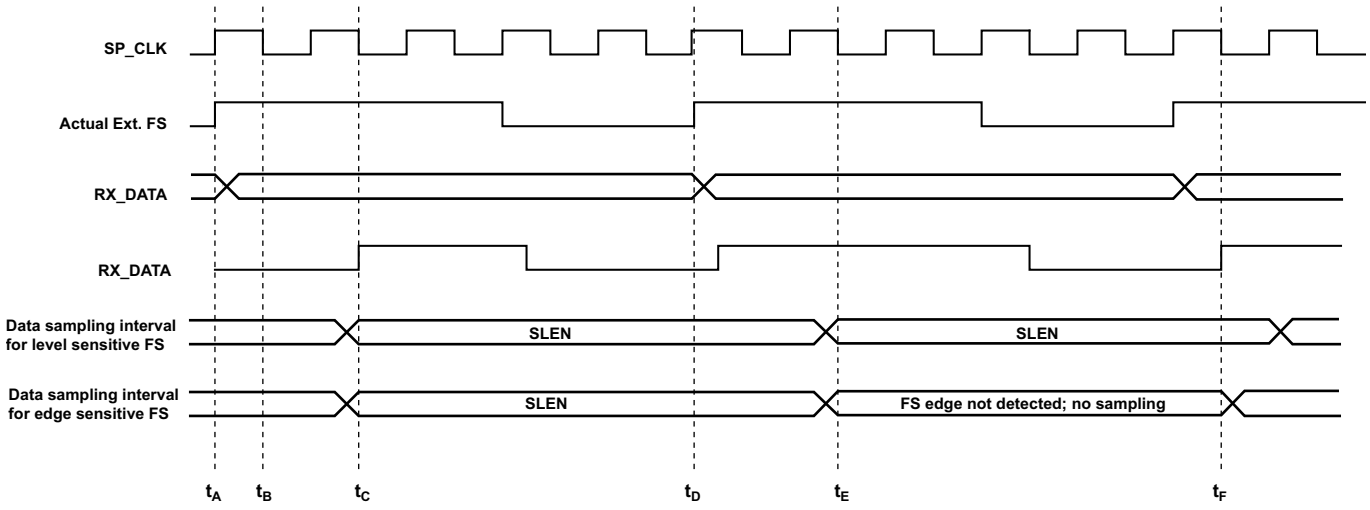


Figure 26-5: Level-Sensitive Frame Sync Vs. Edge Sensitive Frame Sync

Frame Sync edge detection is used by default for Stereo modes. MCM mode and DSP serial mode have an option to choose between edge detection and normal mode of FS detection.

NOTE: When the SPORT is enabled, an already active externally applied frame sync is not allowed to start operation. The SPORT waits for a valid state change from inactive to active for the external frame sync to consider it valid.

Serial Word Length

The `SPORT_CTL_A.SLEN` field of serial port control register determines the word length of serial data to transmit and receive. Each SPORT half can independently handle word lengths up to 32 bits. The minimum allowable word length depends on operating mode selected. Words smaller than 32 bits are right-justified in the transmit or receive buffers to least significant bit (LSB) position. However, data can be shifted-in or out in MSB first or LSB first format according to `SPORT_CTL_A.LSBF` bit setting. Also, the received word can be sign-extended while storing it in processor memory.

The value of the `SPORT_CTL_A.SLEN` field can be calculated as:

SLEN = Serial port word length - 1

The range of valid word lengths in the operating modes of SPORT are as shown in the following table.

Table 26-11: Data Length Versus SPORT Operating Modes

Mode	Serial Port word length (SLEN+1)
Standard DSP Serial	4–32
I^2S	5–32
Left-Justified	5–32
Right-Justified	5–32

Table 26-11: Data Length Versus SPORT Operating Modes (Continued)

Mode	Serial Port word length (SLEN+1)
Multichannel	5–32
Packed I^2S	5–32

NOTE: If the companding feature is enabled on the data path, 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/receive, the multichannel mode can be used (by enabling more than one channel).

Operating Modes

The SPORT has a number of operating modes:

- Standard serial mode
- I^2S mode
- Left-justified mode
- Right-Justified mode
- Multichannel mode
- Packed I^2S mode

The SPORT halves within a SPORT can be independently configured in any of these operating modes, unless they are not coupled together using SPMUX logic. Each SPORT half has its own set of control and data registers and are programmed similarly.

The main control register of serial port, `SPORT_CTL_A`, controls the operating modes of the SPORT. The following table lists all the bits of the control register. The `SPORT_CTL_A` register is unique in that the bit function may change depending on the operating mode selected. It should be noted that many bits in the control registers, that control the function of the mode, are the same bit but have a different name depending on the operating mode. The bits common across operating mode columns (for example `SPORT_CTL_A.SLEN`) signifies that they function similarly across those operating modes. However, the bits divided as per operating modes (for example `SPORT_CTL_A.LFS`) indicate different meaning depending on operating mode. Further, some bits are reserved depending on mode of operation (for example the `SPORT_CTL_A.FSR` bit is reserved in I^2S, left-justified sample pair, packed I^2S and in multichannel mode).

NOTE: When changing operating modes, clear the serial port control register before the new mode is written to the register.

Table 26-12: Control Bits comparison for different operating modes

Bit (NAME)	Standard Serial	I2S and Left-Justified	Right-Justified	Multichannel	Packed I2S
Control Bits					
0 (SPENPRI)	Yes				
2–1 (DTYPE)	Yes	Reserved		Yes	
3 (LSBF)	Yes	Reserved		Yes	
8–4 (SLEN)	Yes				
9 (PACK)	Yes				
10 (ICLK)	Yes				
11 (OPMODE)	Yes				
12 (CKRE)	Yes	Reserved		Yes	
13 (FSR)	Yes	Reserved			
14 (IFS)	Yes	Reserved		Yes	
15 (DIFS)	Yes			Reserved	
16 (LFS)	Yes	Yes		Yes	Yes
17 (LAFS)	Yes			Reserved	
18 (RJUST)	Reserved		Yes	Reserved	
19 (FSED)	Yes	Reserved		Yes	Reserved
20 (TFIEN)	Yes				
21 (GCLKEN)	Yes		Reserved		
24 (SPENSEC)	Yes				
25 (SPTRAN)	Yes				
Status Bits					
26 (DERRSEC)	Yes				
28–27 (DXSSEC)	Yes				
29 (DERRPRI)	Yes				
31–30 (DXSPRI)	Yes				

Mode Selection

The serial port operating mode is configured in the SPORT_CTL_Aand the SPORT_MCTL_A registers. The following table provides values for each of the bits in the SPORT serial control registers that must be set in

order to configure a specific SPORT operation mode. The shaded columns indicate that the bits come from different control registers.

Table 26-13: SPORT Operating Modes

Operating Modes	OPMODE (11)	LAFS (17)	RJUST (18)	Multichannel Control Register (SPORT_MCTL_A)
Standard DSP Serial	0	Valid	0	0
I2S	1	0	0	0
Left-Justified	1	1	0	0
Right-Justified	1	1	1	0
Multichannel	0	x	0	1
Packed I^2S mode	1	x	0	1

The following sections provide detailed information on each operating mode available using the serial ports.

Standard 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. The standard serial mode lets programs configure serial ports for use by a variety of serial devices such as serial data converters and audio codecs. In order to connect to these devices, a variety of clocking, framing, and data formatting options are available.

Timing Control Bits

Several bits in the serial port control register enable and configure standard serial mode operation.

- SLEN: serial word length select (4-32 bits)
- LSBF: little endian Vs big endian serial bit format
- ICLK: Internal clock generation Vs external clock mode
- CKRE: sampling edge as rising edge Vs falling edge
- IFS: Internal frame sync generation Vs external FS mode
- FSR: framed mode Vs unframed mode
- DIFS: Data-dependent frame sync Vs data-independent frame sync

- LFS: active-high FS Vs active-low FS
- LAFS: early frame sync Vs late frame sync
- PACK: 16-bit to 32-bit packing enable Vs packing disable
- GCLKEN: normal free-running clock Vs Gated clock mode

Clocking Options

In standard serial mode, the serial ports can either accept an external serial clock or generate it internally. The SPORT_CTL_A.ICLK bit determines the selection of these options. 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.

In addition, the serial clock edge can be selected for the sampling or driving serial data and/or frame sync. This selection is performed using the SPORT_CTL_A.CKRE bit.

- If SPORT_CTL_A.CKRE = 0, incoming data and/or frame sync signals are sampled with respect to falling edge of serial clock; while data and/or frame sync output signals are driven at the rising edge of clock.
- If SPORT_CTL_A.CKRE = 1, incoming data and/or frame sync signals are sampled with respect to rising edge of serial clock; while data and/or frame sync output signals are driven at the falling edge of clock.
- The SPORT_CTL_A.GCLKEN bit enables clock gating option, in which serial clock is active only during the valid data bits.

Frame Sync Options

The following sections provide generic information about how frame sync signal is used by the serial port in an operating mode. Note that SPORT halves within a SPORT are independently configurable. Additional information about frame syncs and data sampling that applies to a specific operating mode can be found in *Operating Modes*.

Data-Dependent Versus Data-Independent Frame Sync

When a SPORT is configured as a transmitter (SPORT_CTL_A.SPTRAN = 1) and if data-independent frame sync select (SPORT_CTL_A.DIFS) bit = 0, then an internally-generated transmit frame sync is only output when a new data word has been loaded into the channel transmit buffer of the SPORT. In other words, frame sync signal generation and therefore data transmission is data-dependent. This mode of operation allows data to be transmitted only at specific times.

When SPORT is configured as receiver (SPORT_CTL_A.SPTRAN = 0) and if SPORT_CTL_A.DIFS = 0, then a receive frame sync signal is generated only when receive data buffer status is not full.

The data-independent frame sync mode allows the continuous generation of the framing signal, regardless of new data in the buffers. Setting SPORT_CTL_A.DIFS activates this mode. When SPORT_CTL_A.DIFS = 1,

a transmit frame sync signal is generated regardless of the transmit data buffer status (if SPORT_CTL_A.SPTRAN = 1) or receive data buffer status (if SPORT_CTL_A.SPTRAN = 0).

Note that the SPORT DMA controller typically keeps the transmit buffer full. The application is responsible for filling the transmit buffers with data.

Early Versus Late Frame Syncs

The frame sync signals can be early or late. Frame sync signals can occur during the first bit of each data word (late) or during the serial clock cycle immediately preceding the first bit (early). The SPORT_CTL_A.LAFS bit of the serial port control register configures this option.

By default, when SPORT_CTL_A.LAFS is cleared (=0), the frame sync signal is configured as early framing signal. This is the normal mode of operation. In this mode, the first bit of the transmit data word is available (and the first bit of the receive data word is latched) in the serial clock cycle after the frame sync is asserted. The frame sync is not checked again until the entire word has been transmitted (or received). In multichannel operation, this is the case when the frame delay is one.

If data transmission is continuous in early framing mode (in other words, the last bit of each word is immediately followed by the first bit of the next word), then frame sync signal occurs during the last bit of each word. Internally-generated frame syncs are asserted for one clock cycle in early framing mode. This is not an error condition, so the SPORT_ERR_A.FSERRSTAT bit is not flagged.

When SPORT_CTL_A.LAFS is set (=1), late frame syncs are configured; this is the alternate mode of operation. In this mode, the first bit of the transmit data word is available (and the first bit of the receive data word is latched) in the same serial clock cycle that the frame sync is asserted. In multichannel operation, this is the case when frame delay is zero. Receive data bits are latched by serial clock edges, 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. Externally-generated frame syncs are only checked during the first bit. They do not need to be asserted after that time period.

Therefore, for early framing, the frame sync precedes data by one cycle; for late framing, the frame sync is checked on the first bit only. The following figure illustrates the two modes of frame signal timing.

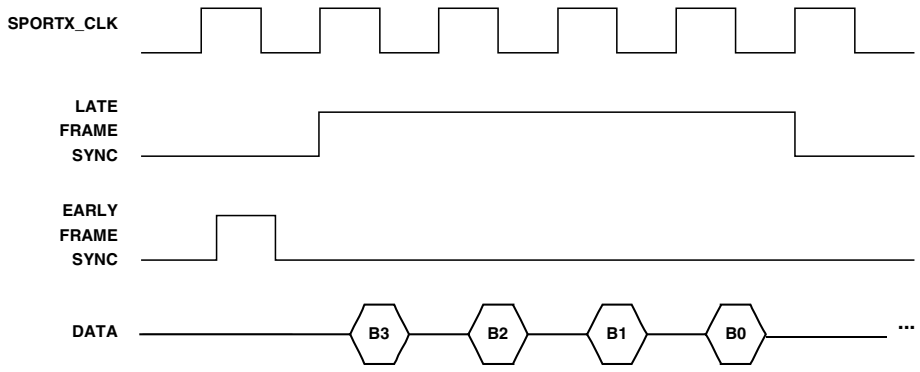


Figure 26-6: Normal Framing (Early Frame Sync) Versus Alternate Framing (Late Frame Sync)

Framed Versus Unframed Frame Syncs

The use of frame sync signal is optional in serial port communications. The SPORT_CTL_A.FSR (frame sync required) bit determines whether framing signal is required.

When SPORT_CTL_A.FSR bit is set (=1), a frame sync signal is required for every data word. To allow continuous transmission from the processor, each new data word must be loaded into the transmit buffer before the previous word is shifted out and transmitted.

When SPORT_CTL_A.FSR is cleared (=0), the corresponding frame sync signal is not required. A single frame sync is required to initiate communications but it is ignored after the first bit is transferred. Data words are then transferred continuously in what is referred to as an unframed mode. Unframed mode is appropriate for continuous reception. The following figure shows the framed vs unframed mode of serial port operation.

NOTE: When DMA is enabled in a mode where frame syncs are not required, DMA requests may be held off by chaining or may not be 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.

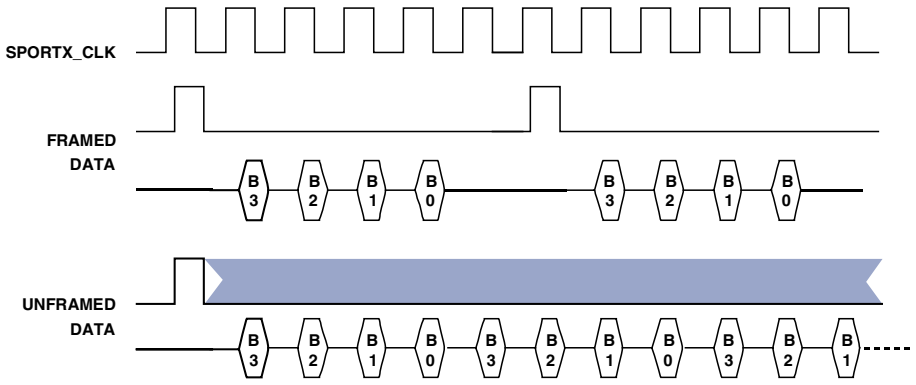


Figure 26-7: Framed Versus Unframed Data Stream

Logic Level

The framing signals may be active high or active low. The SPORT_CTL_A.LFS bit selects the logic level of the frame sync signals.

- 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 frame sync signal.

Stereo Modes

The processor serial port support three widely used stereo modes, which are I²S mode, Left-Justified mode and right-Justified mode. In these modes, the serial data stream consists of left and right channels. These modes are described in the following sections.

Channel Order First

The active low frame sync (SPORT_CTL_A.LFS) bit, which determines the polarity of frame sync level in DSP serial mode/multichannel mode, holds different meaning for stereo modes of SPORT operation. For left-justified, I²S and packed I²S modes, the following table demonstrates which word is transmitted or receive first depending on the SPORT_CTL_A.LFS bit setting.

Table 26-14: Channel Order First Bit Settings

OPMODE	LFS=0 (Left Channel First, Default)	LFS=1 (Right Channel First
Left-Justified	Data first after rising edge	Data first after falling edge
I²S	Data first after falling edge	Data first after rising edge
Packed I²S	Data first after rising edge	Data first after falling edge

I²S Mode

I²S mode is a very commonly used stereo mode. In this mode, for each frame sync cycle, two samples of data are transmitted/received. One sample is transmitted/received on the low segment of the frame sync, which is known as left channel. The other sample is transmitted/received on the high segment of the frame sync, which is known as right channel.

The SPORT can be configured in I²S mode by setting SPORT_CTL_A.OPMODE = 1, SPORT_CTL_A.LFS = 0 and SPORT_MCTL_A.MCE = 0.

Protocol Configuration Options

Several bits in the SPORT_CTL_A control register enable and configure I²S mode of operation:

- SLEN: serial word length select. For I²S mode, the range of allowable word length is 5-32 bits.
- LSBF: little endian or big endian serial bit format. For I²S mode, serial data should be in big endian format (MSB bit is transmitted/received first). But, it can be changed depending on the user's preference when emulating similar non-standard protocol
- ICLK: Internal bit clock generation or external bit clock mode
- IFS: Internal frame sync generation or external FS mode. In I²S mode, master serial port is the one which generates bit clock and L/R clock (frame sync) internally for the serial communication. The serial port which accepts these clocking signals is known as slave. So, in standard I²S mode, the SPORT_CTL_A.IFS bit should depend (equal to) on SPORT_CTL_A.ICLK bit setting. However, as an enhancement,

the SPORT_CTL_A.IFS bit setting may be changed depending on the application when emulating a non-standard protocol.

- LFS: left channel first or right channel first. This bit setting may be changed depending on user's preference to sample right channel first (LFS = 0) or left channel first (LFS = 1).
- CKRE: sampling edge as rising edge or falling edge.
- PACK: 16-bit to 32-bit packing enable or packing disable.

Serial Clock and Frame Sync Rates

The serial bit clock rate for internal clocks can be set using a SPORT_DIV_A.CLKDIV bit field; while the L/R clock rate for internal frame sync can be set using the SPORT_DIV_A.FSDIV bit field in the same register, depending on SPORT_CTL_A.ICLK and SPORT_CTL_A.IFS bit settings.

The following figure shows the timing in I^2S mode. Note that in I^2S mode, the data is delayed by one SCLK cycle and the operation transfer starts on the left channel first.

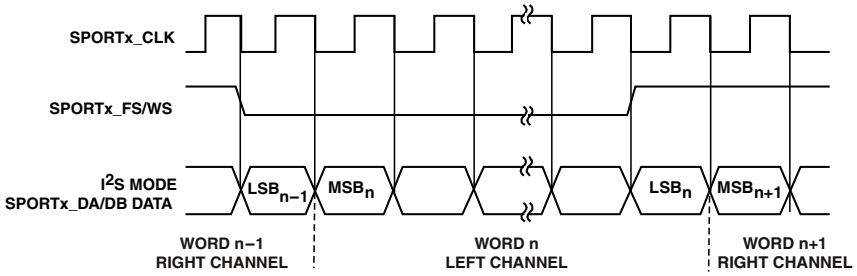


Figure 26-8: Word Select Timing in I2S Mode

Unlike serial port of SHARC processor, the serial port of the processor does not generate a frame sync (L/R clock) edge after the transmission of last word in the DMA (same as behavior in the earlier Blackfin processors). Standard I^2S receivers looks for the edge to latch and read data. Therefore I^2S slave receivers connected to Blackfin SPORT may not able to latch the last word of the TX DMA.

Left-Justified Mode

Left-justified mode is a stereo mode. In this mode, for each frame sync cycle, two samples of data are transmitted/received. One sample is transmitted/received on the high segment of the frame sync, which is known as left channel. The other sample is transmitted/received on the low segment of the frame sync, which is known as right channel.

This operating mode is simply a subset of the I2S mode. The SPORT can be configured in Left-Justified mode by setting the SPORT_CTL_A.OPMODE and SPORT_CTL_A.LAFS bits and clearing the SPORT_MCTL_A.MCE bit.

Protocol Configuration Options

Several bits in the SPORT_CTL_A control register enable and configure the left-justified mode of operation:

- SLEN: serial word length select. For left-justified mode, the range of allowable word length is 5-32 bits.
- LSBF: little endian or big endian serial bit format. For left-justified mode, serial data should be in big endian format (MSB bit is transmitted/received first). But, it can be changed depending on the user's preference when emulating similar non-standard protocol
- ICLK: Internal bit clock generation or external bit clock mode
- IFS: Internal frame sync generation or external FS mode. In left-justified mode, master serial port is the one which generates bit clock and L/R clock (frame sync) internally for the serial communication. The serial port which accepts these clocking signals is known as slave. So, in standard left-justified mode, the SPORT_CTL_A.IFS bit should depend (equal to) on SPORT_CTL_A.ICLK bit setting. However, as an enhancement, the SPORT_CTL_A.IFS bit setting may be changed depending on the application when emulating a non-standard protocol.
- LFS: left channel first or right channel first. In standard left-justified mode, left channel is sampled first (corresponds to SPORT_CTL_A.LFS = 0). However, SPORT_CTL_A.LFS bit setting may be changed depending on user's preference to sample right channel first (first data after rising edge of L/R clock).
- CKRE: sampling edge as rising edge or falling edge.
- PACK: 16-bit to 32-bit packing enable or packing disable.

Serial Clock and Frame Sync Rates

The serial bit clock rate for internal clocks can be set using a SPORT_DIV_A.CLKDIV field; while the L/R clock rate for internal frame sync can be set using SPORT_DIV_A.FSDIV field, depending on SPORT_CTL_A.ICLK and SPORT_CTL_A.IFS bit settings. The following figure shows the serial port timing in left-justified mode (it is shown in MSB bit first format, but LSB bit first format is also possible). As shown, the first bit of a word is transmitted/received in the same clock cycle as the word select (SPORT_FS) signal changes.

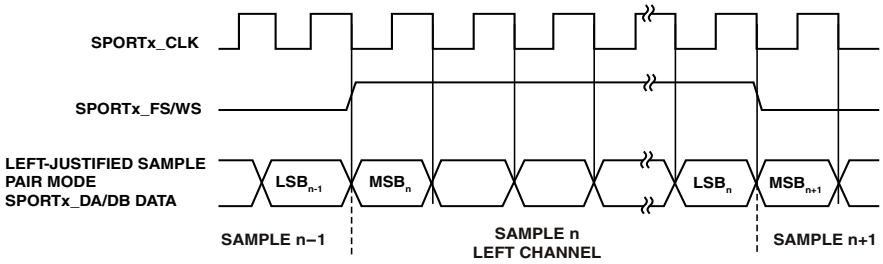


Figure 26-9: Word Select Timing in Left-Justified Mode

Right-Justified Mode

Right-justified mode is a standard commonly available in most of the SPORT compatible devices such as ADC and DACs. Right-justified mode requires that the design align the data to the end of the frame sync. The SPORT_CTL_A.RJUST bit aligns the serial data to the end of the frame sync.

The following figure shows the SPORT timing in right-justified mode. As shown, the transmitter aligns the data to be transmitted such that the last bit of the serial word is sent in the last clock cycle of the word select (frame sync) signal marking the channels. The timing seems similar to left-justified mode (where the transmitter sends the MSB bit of the serial word in the same clock cycle as the word select signal changes), but data is shifted such that it aligns to end of the channel.

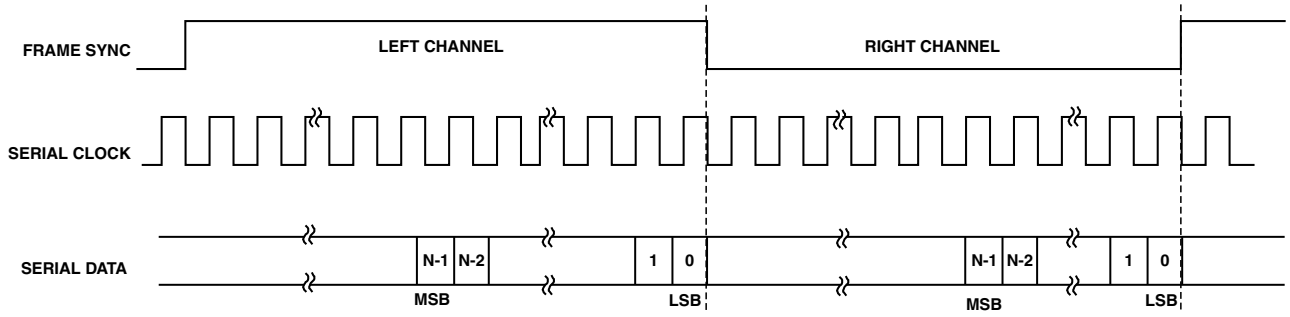


Figure 26-10: Word Select Timing in Right-Justified Mode

NOTE: For some SPORT compatible ADC or DACs (for example the AD1871) right-justified mode is limited to some commonly used ratios such as 64 FS and 128 FS as the bit clock frequency (where FS is the sampling frequency of ADC/DACs, known as L/R clock or the frame sync of the SPORT).

As an illustration, consider the SPORT timing for right-justified mode, as shown in the figure below. If the L/R clock runs at the FS rate and the SPORT clock at the 64 FS rate, the frame sync width (either channel) is limited to 32 SPORT clock periods or 32 bits per channel. If the data is confined to 24 bits, the SPORT introduces a 32–24=8 bit clock delay before it starts to transmit/capture data.

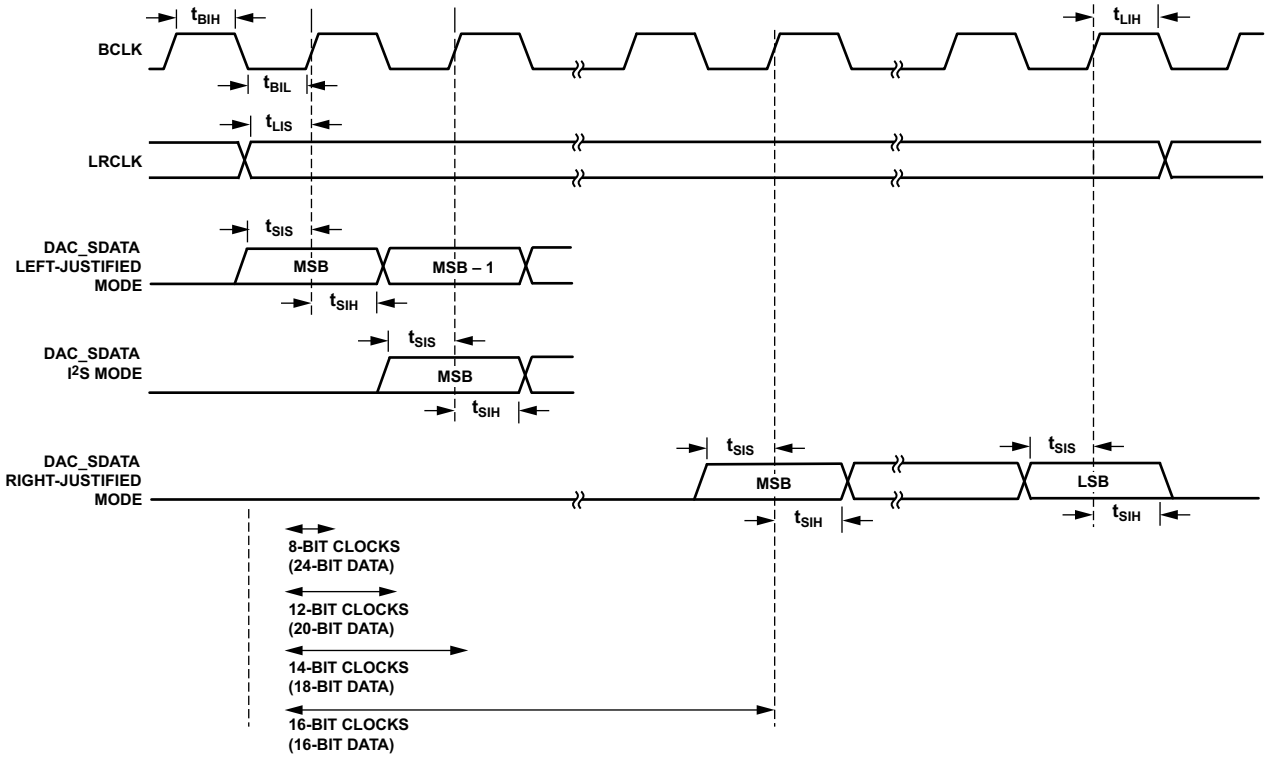


Figure 26-11: Timing Comparison Between Different Stereo Modes

Similarly, if 128 FS bit clock frequency is to be supported, then the frame sync width becomes 64 serial clock periods (bits) per channel. In this case, the delay can be a maximum of (64 – minimum serial data length in right-justified mode) = 59 bits (the minimum `SPORT_CTL_A.SLEN` setting is 4). This implies that a 6-bit counter is needed to set this delay.

Therefore, using this counter in right-justified mode, the starting point of the first bit is delayed so that the serial data is aligned properly with the end of the channel. A 6-bit counter is added for this purpose in the stereo mode. This counter is programmed by writing into the `SPORT_MCTL_A` 16-21 bit field. Note that these bits are used to configure the window offset size in multichannel mode. But since stereo serial mode and multichannel mode are mutually exclusive, the separation of role of this field in each mode is clearly defined and implemented. The software has to configure this register with the appropriate delay keeping in mind the word length (`SPORT_CTL_A.SLEN`) and the number of bit clocks in one channel (left/right).

Timing Control Bits

The following bits in the `SPORT_CTL_A` register enable and configure right-justified mode.

- SLEN: serial word length select. For right-justified mode, the range of allowable word length is 5-32 bits.
- LSBF: little endian or big endian serial bit format. For right-justified mode, serial data should be in big endian format (MSB bit transmitted/received first). But, it can be changed depending on the application when using non-standard protocol.

- ICLK: Internal bit clock generation or external bit clock mode
- IFS: Internal frame sync generation or external FS mode.
- LFS: left channel first or right channel first. In standard right-justified mode, the left channel is sampled first (corresponds to SPORT_CTL_A.LFS = 0). However, the SPORT_CTL_A.LFS bit setting may be changed depending on the application to sample the right channel first (first data after falling edge of L/R clock).
- CKRE: sampling edge as rising edge or falling edge.
- PACK: 16-bit to 32-bit packing enable or packing disable.
- MCTL16-21: 6-bit counter depends on SPORT_CTL_A.SLEN and number of bit clocks in a channel.

Serial Clock and Frame Sync Rates

The serial bit clock rate for internal clocks can be set using the SPORT_DIV_A.CLKDIV; while the L/R clock rate for internal frame sync can be set using the SPORT_DIV_A.FSDIV bit field in the same register, depending on SPORT_CTL_A.ICLK and SPORT_CTL_A.IFS bit settings.

Multichannel Mode

The processor's SPORTs offer a multichannel mode of operation, which allows the SPORT to communicate in a time division multiplexed (TDM) serial system. In multichannel communications, each data word of the serial bit stream occupies a separate channel. Each word belongs to the next consecutive channel. For example, a 24-word block of data contains one word for each of the 24 channels.

The multichannel mode of SPORT can be selected by setting SPORT_CTL_A.OPMODE = 0 and SPORT_MCTL_A.MCE = 1.

Up to 128 channels are available for transmitting or receiving. The SPORT can automatically select some words for particular channels while ignoring others. In other words, each SPORT can receive or transmit data selectively from any of the 128 channels. These 128 channels can be any 128 out of the 1024 total channels in the system. 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

Optionally, data companding and DMA transfers can be used in multichannel mode on both primary and secondary data lines.

The SPORT multichannel select registers (SPORT_CS0_A) must be programmed before enabling SPORT operation for multichannel mode. This is especially important in DMA data unpacked mode, since the SPORT data buffers begin operation immediately after the SPORT data lines are enabled. The SPORT_MCTL_A.MCE must also be enabled prior to enabling SPORT operation.

Multichannel mode operates completely independently and each SPORT uses its own serial clock and frame sync signal either internally generated or externally provided.

Protocol Configuration Options

The following bits in the SPORT_CTL_A and SPORT_MCTL_A registers enable and configure multichannel mode.

- SLEN: serial word length select
- LSBF: little endian or big endian serial bit format
- ICLK: Internal clock generation or external clock mode
- CKRE: sampling edge as rising edge or falling edge
- IFS: Internal frame sync generation or external FS mode
- LFS: active-high FS or active-low FS
- PACK: 16-bit to 32-bit packing enable or packing disable
- MFD: Multichannel frame delay
- WSIZE: Number of multichannel channels
- WOFFSET: window offset size
- MCPDE: Multichannel DMA packing enable

Clocking Options

In multichannel mode, the SPORTs can either accept an external serial clock or generate it internally. The SPORT_CTL_A.ICLK bit determines the selection of these options. For internally-generated serial clock (SPORT_CTL_A.ICLK = 1), the SPORT_DIV_A.CLKDIV bit field configures the serial clock from the system clock.

In addition, the serial clock edge can be selected for the sampling or driving serial data and/or frame sync. This selection is performed using the SPORT_CTL_A.CKRE bit.

- If SPORT_CTL_A.CKRE= 0, incoming data and/or frame sync signals are sampled with respect to falling edge of serial clock; while data and/or frame sync output signals are driven at the rising edge of clock.
- If SPORT_CTL_A.CKRE = 1, incoming data and/or frame sync signals are sampled with respect to rising edge of serial clock; while data and/or frame sync output signals are driven at the falling edge of clock.

Frame Sync Options

The frame sync signal synchronizes the channels and restarts each multichannel sequence. The SPORT_ FS signal initiates the start of the channel 0 data word. The frame sync period in multichannel is defined as:

FS period = [(SPORT_CTL_A.SLEN + 1) × number of channels] - 1

The frame sync can be configured in master or slave mode based on the setting of the SPORT_CTL_A.IFS bit and its logic level can be changed using the SPORT_CTL_A.LFS bit.

In multichannel mode, frame sync timing is similar to late frame mode (though the SPORT_CTL_A.LAFS bit is reserved in this mode)—the first bit of the transmit data word is available and the first bit of the receive data word is sampled in the same serial clock cycle that the frame sync is asserted, provided that multichannel frame delay (SPORT_MCTL_A.MFD) is set to 0.

The frame sync signal is used for the block or frame start reference, after which the word transfers are performed continuously with no further frame sync signals required during the ongoing frame for different channels. Therefore, internally generated frame syncs are always data independent (SPORT_CTL_A.DIFS bit is reserved).

Transmit Data Valid (TDV)

Each serial port has its own Transmit Data Valid signal (SPT_ATDV) which is active during the transmission of enabled words. Because the serial port signals are three-stated when the time slot is not active, the SPT_ATDVsignal specifies if the SPORT data is being driven by the processor. It serves as an output-enabled signal for the data transmit pin. After the transmit data buffer is loaded, transmission begins and the SPORT_TDV signal is asserted.

The polarity of this Transmit Data Valid signal is always active high in that SPT_ATDV is asserted high when a data is transmitted during the active channel slot of serial port.

The following figure shows an example of timing for a multichannel transfer having following characteristics.

- The half SPORT pair of SPORT0, A and B, is configured as a transmitter and receiver respectively; while half SPORT A of SPORT1 is configured as transmitter.
- The serial clock and frame sync signals are input to all of these HSPORTs.
- Only primary channels of these SPORTs are enabled (0).
- Multichannel is configured to 8 channels.
- SPORT0_A drives data (on its primary data line) during slot 1–0 which asserts SPORT0_ATDV for 2 slots.
- SPORT1_A drives data (on its primary data line) during slot 3–2 which asserts SPORT1_ATDV for 2 slots.
- SPORT0_B receives data (from its primary data line) during slot 3–0.

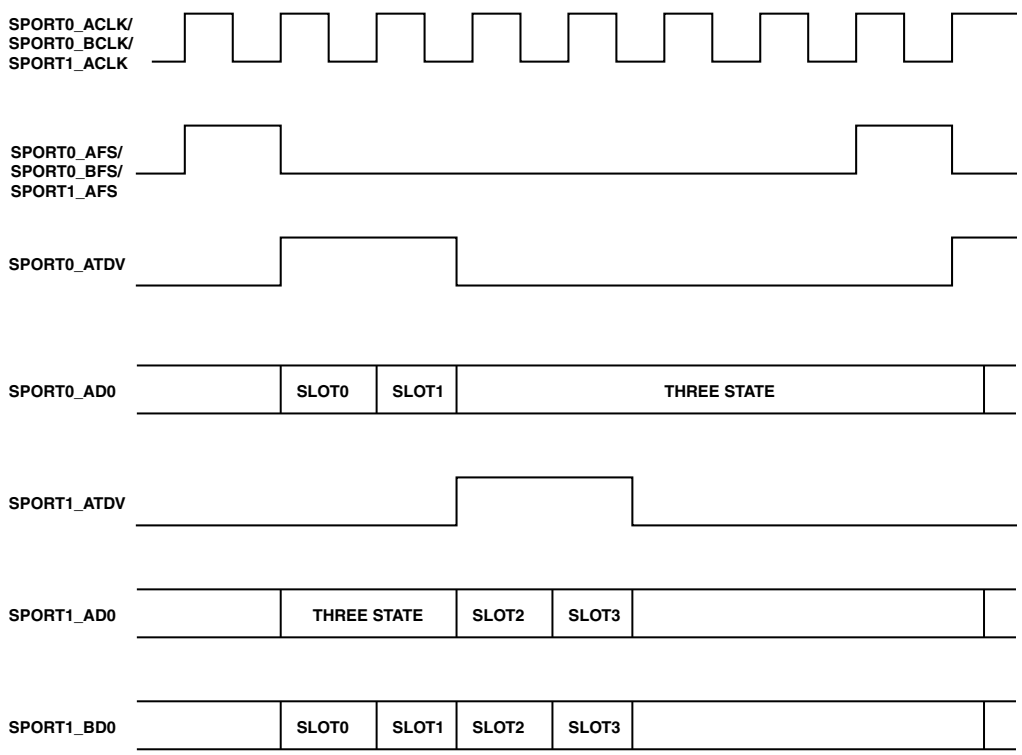


Figure 26-12: Multichannel Timing

Active Channel Selection Registers (SPORT_CS0_A)

In multichannel mode, SPORT supports up to 128 channels for transmitting or receiving. It can receive or transmit data selectively from any of the 128 channels. Specific channels can be individually enabled or disabled, using multichannel selection registers (CSx), to select the words that are transmitted or received during multichannel communications. Data words from the enabled channels are transmitted or received, while disabled channel words are three-stated or ignored.

Each of the four multichannel selection registers is 32 bits in length. Therefore these registers provide channel selection for 128 (0 to 127) channels. Setting a bit enables that channel so that the serial port selects its word from the multiple-word block of data (for either receive or transmit). The 128 channels are sequentially numbered from bit 0 in the CS0 register to bit 31 of SPORT_CS3_A register. As an example setting bit 13 of the SPORT_CS1_A register enables channel number 45 (31+13+1); similarly setting bit 5 of the SPORT_CS3_A register enables channel number 101 (31+32+32+5+1).

Multichannel Frame Delay (MFD)

The 4-bit multichannel frame delay (SPORT_MCTL_A.MFD) field in the multichannel control registers (MCTL_x) specifies a delay between the frame sync pulse and the first data bit in frame. The value of SPORT_MCTL_A.MFD is the number of serial clock cycles of the delay. This multichannel frame delay allows the processor to work with different types of telephony interface devices.

A value of zero for SPORT_MCTL_A.MFD causes the frame sync to be concurrent with the first data bit. The maximum value allowed for SPORT_MCTL_A.MFD is 15. If SPORT_MCTL_A.MFD>0, a new frame sync may occur during the last channels of a previous frame, which is considers as valid frame sync signal.

NOTE: If more than 15 bits frame delay is required, the Window Offset field may be used to delay the start of channel 0.

Number of Multichannel Slots (WSIZE)

Select the number of channels used in multichannel operation by using the 7-bit SPORT_MCTL_A.WSIZE field in the multichannel control register. Set SPORT_MCTL_A.WSIZE to the actual number of channels minus one (SPORT_MCTL_A.WSIZE = Number of channels - 1). So, the granularity of number of channels selected is 1.

A 10-bit field in the multichannel mode status register, SPORT_MSTAT_A, holds the channel number which is being serviced in the multichannel operation.

Window Offset (WOFFSET)

The window offset (SPORT_MCTL_A.WOFFSET) field register specifies where in the 1024-channel range to place the start of the active window. A value of 0 specifies no offset and 896 (1024 – 128) is the largest value that allows using all 128 channels.

As an example, a program could define an active window with 8 multichannel slots (SPORT_MCTL_A.WSIZE = 7) and an offset of 93 (SPORT_MCTL_A.WOFFSET = 93). This 8-channel window then resides in the range from 93 to 100.

Neither the window offset nor the number of multichannel slots (SPORT_MCTL_A.WSIZE) can be changed while the SPORT is enabled. If the combination of the window size and the window offset would place any portion of the window outside of the range of the channel counter, none of the out-of-range channels in the frame are enabled.

Companding Selection

Like all other operating modes, companding logic can be applied to serial data. In transmit mode, compression logic is applied to the data to be transmitted; while in receive mode, expansion logic can be applied to received data. The two widely used companding algorithms, A-law and μ-law can be applied by configuring SPORT_CTL_A.DTYPE field of the control register.

If companding is enabled, the companding algorithm is applied to both the data paths. 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 specified with the SPORT_MCTL_A.MCPDE bit setting.

If the bits are set, indicating that data is packed, the SPORT expects the data contained by the DMA buffer corresponds only to the enabled SPORT channels. For example, if an MCM frame contains 10 enabled channels, the SPORT expects the DMA buffer to contain 10 consecutive words for each frame.

If the bits are cleared (the default, indicating that data is not packed), the SPORT expects the DMA buffer to have a word for each of the channels in the active window, whether enabled or not, so the DMA buffer size must be equal to the size of the window. For example, if only channel number 1 and 10 are enabled, then DMA buffer size would have to be 10 words (unless the secondary side is enabled). The data to be transmitted or received would be placed at addresses 1 and 10 of the buffer, and the rest of the words in the DMA buffer would be ignored.

Multichannel Frame

A multichannel frame contains more than one channel, as specified by the `SPORT_MCTL_A.WSIZE` field and window offset field of multichannel control register. A complete multichannel frame consists of 1–1024 channels, starting with channel 0. The particular channels of the multichannel frame that are selected for the SPORT are a combination of the window offset, the window size, and the multichannel select registers.

The following figure illustrates the relationship between different parameters of multichannel timings. Frame length is set by frame sync divider or external frame sync period

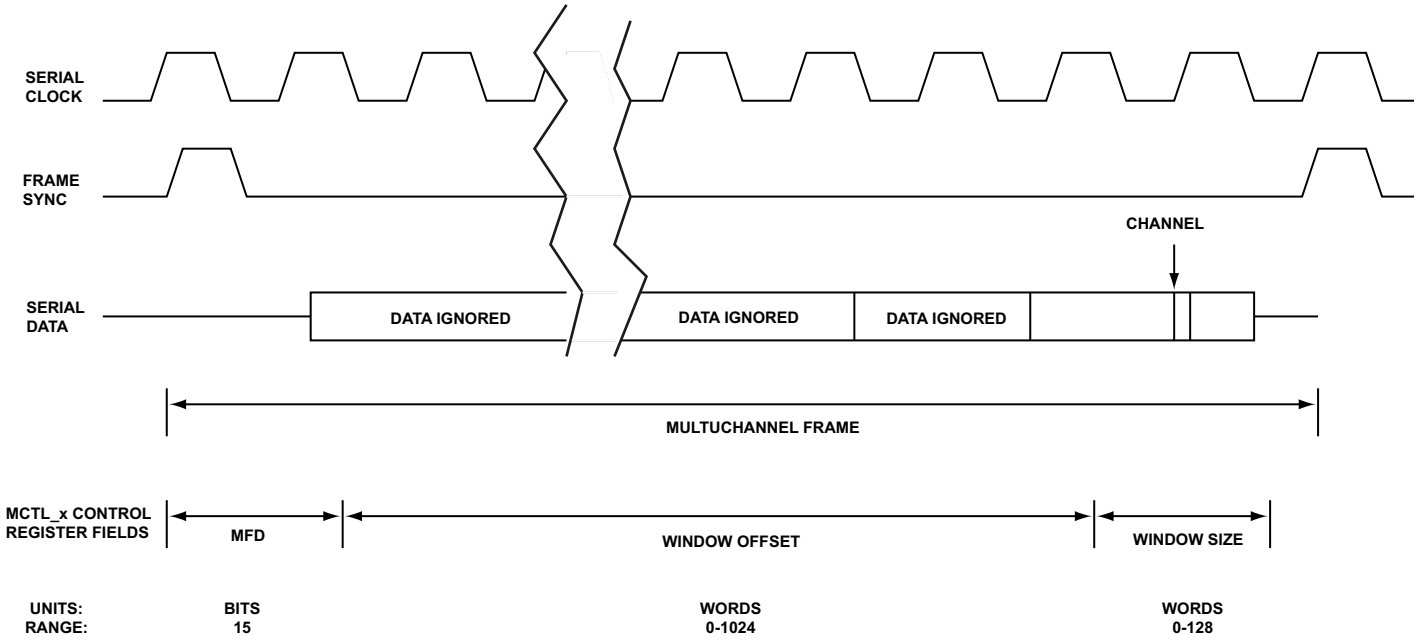


Figure 26-13: Relationships for Multichannel Parameter

Packed I2S Mode

A packed I^2S mode is available in the SPORT and used for audio codec communications using multiples channels. This mode allows applications to send more than the standard 32 bits per channel available

through standard I^{2}S mode. Packed mode is implemented using standard multichannel mode (and is therefore programmed similarly to multichannel mode).

The SPORT can be configured in packed I^{2}S mode by setting the SPORT_CTL_A.OPMODE bit and the SPORT_MCTL_A.MCE bit.

Similar to multichannel mode, packed I^{2}S mode also supports the maximum of 128 channels as the maximum of (128 x 32) bits per left or right channel.

As shown in the following figure, packed waveforms are the same as the wave forms used in multichannel mode, except that the frame sync is toggled for every frame, and therefore emulates I^{2}S mode. So it is a hybrid between multichannel and I^{2}S mode.

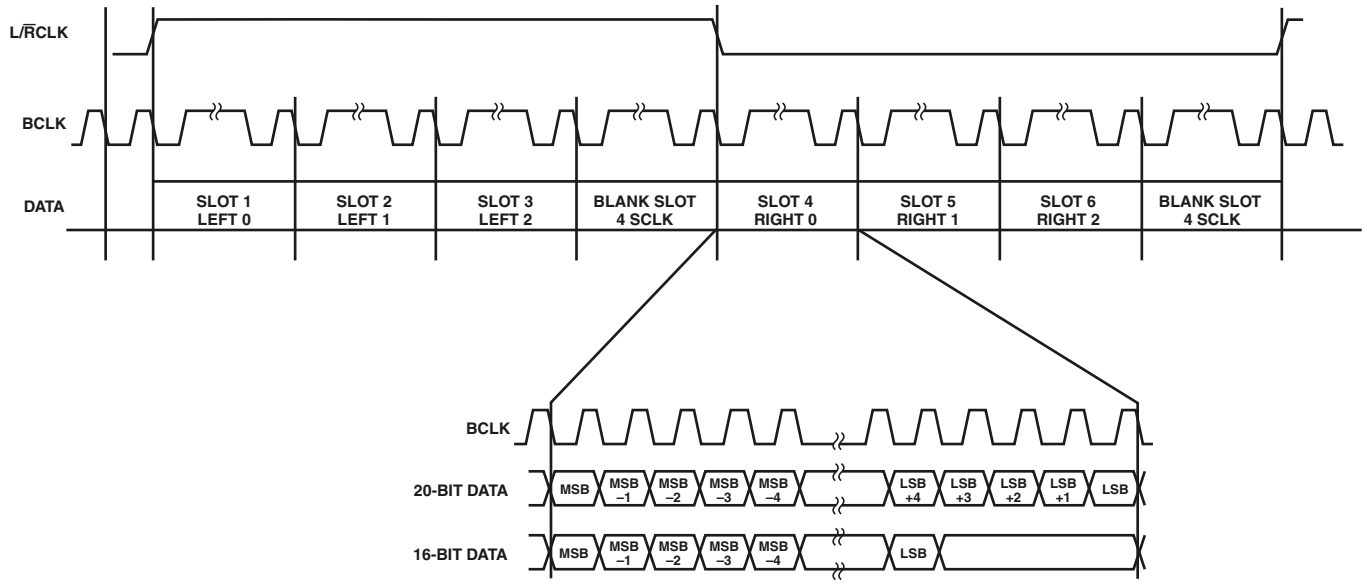


Figure 26-14: Packed I^{2}S Mode 128 Operation

Protocol Configuration Options

Several bits in the SPORT_CTL_A and SPORT_MCTL_A registers enable and configure packed I^{2}S mode.

- SLEN: serial word length select
- LSBF: little endian or big endian serial bit format
- ICLK: Internal clock generation or external clock mode
- CKRE: sampling edge as rising edge or falling edge
- IFS: Internal frame sync generation or external FS mode
- LFS: left-channel first or right channel first
- PACK: 16-bit to 32-bit packing enable or packing disable

- MFD: Multichannel frame delay
- WSIZE: Number of multichannel channels
- WOFFSET: window offset size
- MCPDE: Multichannel DMA packing enable

Clocking Options

In packed mode, the serial ports can either accept an external serial clock or generate it internally. The SPORT_CTL_A.ICLK register determines the selection of these options. For internally-generated serial clock (SPORT_CTL_A.ICLK = 1), the SPORT_DIV_A.CLKDIV bit field configures the serial clock rate from the system clock.

The programs can also select the serial clock edge that is used for sampling or driving serial data and/or frame syncs. This selection is performed using theSPORT_CTL_A.CKRE bit.

Frame Sync Options

The frame sync period in packed mode is defined as:

FS period = [(SPORT_CTL_A.SLEN + 1) × number of channels] - 1.

The frame sync can be configured in master or slave mode depending on the SPORT_CTL_A.IFS bit setting of serial port control register. Moreover the logic level can be changed with the SPORT_CTL_A.LFS bit setting.

Gated Clock Mode

Some of the ADC/DACs support the SPI compatible protocol for the interface. To communicate with such ADC/DACs, the serial port must support the gated clock mode of operation, in which the data valid information is embedded into the clock. Therefore, in gated clock mode, the clock should be active only when active data is being transmitted or received.

The processor features the gated clock function. The SPORT_CTL_A.GCLKEN (Gated clock mode select) control bit, is used to configure the serial port in gated clock mode. To enable gated clock mode of operation, SPORT must be programmed to comply with the following requirements.

- Gated clock mode feature is available in standard serial mode, left-justified and I^2S mode of operation only.

  [Note that among the stereo modes, right justified mode cannot be operated in gated clock mode because during the inactive period in a frame (the period between the leading edge of the frame sync and the first active data bit), there is a delay counter running inside the serial port. The counter operates on the serial clock and any interruption in clock makes the counter go out of sync].

- Gated clock mode has the following valid settings for other control bits.

  - Both serial clock and frame sync signals generated internally
  - Both serial clock and frame sync signals provided externally
  - Frame sync not required' mode (SPORT_CTL_A.FSR = 0) not supported
  - SPORT_CTL_A.DIFS should be programmed as 0 in transmit mode; while it should be programmed as 1 in receive mode.
- There are few necessary conditions to be satisfied when gated clock mode is enabled-
  - Need at least 7 serial clock cycles between enabling the SPORT and first frame sync. If this requirement is not met, the SPORT may drop the first data. (For subsequent data this requirement is not applicable).
  - Frame sync should be in the inactive (deasserted) state when the SPORT is enabled. Else, one extra cycle (in addition to the above mentioned) is needed before the frame sync can be applied. If this requirement is not met, the SPORT may drop the first data.
  - For edge detected frame sync, the frame sync should transition back to inactive state before the current word transmission/reception 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 when the clock is active.

Data Transfers

Serial port data can be transferred to/from internal or external memory in two different methods:

- Core-driven single word transfers
- DMA-driven multiple words transfers, with multiple work units.

DMA transfers can be set up to transfer a configurable number of serial words between the serial port transmit or receive data buffers and internal memory automatically. Core-driven transfers use SPORT interrupts to signal the processor core to perform single word transfers to/from the serial port data buffers.

The following sections provide information on core-driven and DMA-driven data transfers.

Data Buffers

When programming the serial port data channels (primary and/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 while 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 acti-

vated. 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 may 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 at the bit 0 location of the buffer). The upper unused bits may 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) data paths of each SPORT half have separate data buffers, referred to as SPORT_TXPRI_A and SPORT_TXSEC_A respectively.

These transmit data buffers are the 32 bits wide. These buffers must be loaded with the data to be transmitted on the primary and secondary data channels. The data is loaded automatically by the DMA controller or loaded manually by the program running on the processor core.

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 data path of a SPORT half is enabled, programs should not write to the inactive secondary transmit data buffer and vice-a-versa. If the core keeps writing to the inactive buffer, the status of that transmit buffer becomes full and this may 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) data paths 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 data path of a SPORT half is enabled, programs should not read from the inactive secondary receive data buffer and vice-a-versa. If the core keeps reading from the inactive buffer, the status of that receive buffer becomes empty and this may cause the core to hang indefinitely since new data is never received via the input shift register.

Data Buffer Status

Serial ports provide status information about data buffers via the SPORT_CTL_A.DXSPRI (primary channel data buffer status) and SPORT_CTL_A.DXSSEC (secondary channel data buffer status) bits and error status via SPORT_CTL_A.DERRPRI (primary channel error status) and SPORT_CTL_A.DERRSEC (secondary channel error status) bits. Depending on the SPORT_CTL_A.SPTRAN bit setting, these bits reflect the status of either SPORT_TXPRI_A and SPORT_TXSEC_A transmit data buffers or SPORT_RXPRI_A and SPORT_RXSEC_A receive data buffers. These bits indicate whether the buffers are full, partially full or empty.

When attempting to read from an empty receive buffer or to write to a full transmit buffer, the access is delayed until the buffer is ready. This delay is called a core processor hang. To avoid these conditions, always check the buffer status to determine if the access can be made. The status bits in the SPORT_CTL_A register are updated during reads and writes from the core processor even when the serial port is disabled.

Two complete 32-bit words can be stored in the receive buffer while a third word is being shifted 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, a shift register contents overwrite the second word if the first word has not been read out (by the processor core or the DMA controller). When this happens, the receive overflow status is flagged through the error status bits of the SPORT_CTL_A register. The overflow status is generated on the last bit of the third word. The SPORT_CTL_A.DERRPRI and SPORT_CTL_A.DXSSEC status bits are sticky read-only bits and are cleared by disabling the serial port.

NOTE: The status bits in the SPORT_CTL_A register are updated during reads and writes from the core processor even when the serial port is disabled.

Data Buffer Packing

When the SPORT is configured as a receiver with a serial data word length of 16 or less, then received data words may be packed into a 32-bit word. Similarly, if the SPORT is configured as transmitter with a serial data word length of 16 or less, then 32-bit words being transmitted may be unpacked into 16-bit words. This packing/unpacking feature is selected by the SPORT_CTL_A.PACK bit.

When SPORT_CTL_A.PACK = 1, two successive words received 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 applies to both receive (packing) and transmit (unpacking) operations. In this case, the transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.

Companding can be used with word packing or unpacking.

NOTE: When 16-bit received data is packed into 32-bit words and stored in normal word space in processor internal memory, the 16-bit words can be read or written with short word space addresses.

Single Word (Core) Transfers

Individual data words may also be transmitted or received by the serial ports, with interrupts occurring as each data word is transmitted or received. When a serial port is enabled and corresponding DMA is disabled, the SPORT interrupts are generated whenever a complete word has been received in the receive data buffer, or whenever the transmit data buffer is not full.

When performing core driven transfers, write to the buffer designated by the SPORT_CTL_A.SPTRAN bit setting. For DMA driven transfers, the SPORT logic performs the data transfer from internal memory to/from the appropriate buffer depending on the SPORT_CTL_A.SPTRAN bit setting. If the inactive SPORT data buffers are read or written to by the core while the SPORT is being enabled, the core may hang. For example, if a SPORT is programmed to be a transmitter, while at the same time the core reads from the receive buffer of the same SPORT, the core may hang just as it would if it were reading an empty buffer that is currently active and waits for the status to change. This may lock up the core until the SPORT is reset.

To avoid hanging the processor core, check the status of appropriate data buffers when the processor core tries to read a word from a serial port's receive buffer or writes a word to its transmit buffer. The full/empty status can be read using the SPORT_CTL_A.DXSPRI and SPORT_CTL_A.DXSSEC bits.

DMA Transfers

Direct memory access (DMA) provides a mechanism for receiving or transmitting an entire block of serial data before an interrupt is generated. When SPORT DMA is not enabled, the SPORT generates an interrupt every time it receives or starts to transmit a data word. The processor's on-chip DMA controller handles the DMA transfer, allowing the processor core to continue running until the entire block of data is transmitted or received. Service routines can then operate on the block of data rather than on single words, significantly reducing overhead.

Therefore, set the direction bit, DMA enable bit and the serial port enable bit before initiating any operations on the SPORT data buffers. Do not try to access data buffers when the associated DMA channel of serial port is enabled. If the processor operates on the inactive transmit or receive buffers while the SPORT is enabled, it can cause unpredictable results.

Each SPORT half has a dedicated DMA channel, which serves both primary and secondary data paths. In transmit mode, the DMA channel alternatively writes to the primary transmit data buffer and the secondary transmit data buffer. Software must interleave the data of primary and secondary channels into the DMA's transmit buffer. Similarly, in receive mode, the DMA channel alternatively reads from the primary receive data buffer and the secondary data buffer and software must de-interleave the data corresponding to the primary and secondary channels from the DMA's receive buffer.

If the SPORT is configured in stereo mode, the same DMA channel drives/receives both the left and right channels of the enabled data paths (primary and/or secondary). Therefore, in transmit mode, software must interleave the left and right channels' data (of the enabled data paths) into the DMA's transmit buffer. Similarly in receive mode, software should de-interleave the left and right channel data (of the enabled paths) from the DMA's receive buffer.

Since both primary and secondary data paths share the single DMA channel, each SPORT half share a common interrupt vector. Optionally the DMA controller can generate an interrupt at the end of the completion of a DMA transfer and at the end of each work unit of DMA.

The SPORT DMA channels are assigned a higher priority than all other DMA channels (for example, the SPI port) because of their relatively low service rate and their inability to hold off incoming data. Having higher priority causes the SPORT DMA transfers to be performed first when multiple DMA requests occur in the same cycle. The serial port DMA channels are numbered and prioritized in the processor's DMA Channel List table in the DMA chapter.

Although the DMA transfers are performed with 32-bit words, the SPORTs can handle word sizes from 4 to 32 bits (as defined by SPORT_CTL_A.SLEN field). If serial data length is 16 bits or smaller, two data can be packed into 32-bit words for each DMA transfer. This option is selected by setting the SPORT_CTL_A.PACK bit. When serial port data packing is enabled (SPORT_CTL_A.PACK = 1), the transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word. For more information, see *Data Buffer Status*.

NOTE: The DMA channel of a SPORT can access both internal memory and external memory of the processor without any core overhead.

Error Detection

When the serial port is configured as a transmitter, the SPORT_CTL_A.DERRPRI (primary channel error status) and SPORT_CTL_A.DERRSEC (secondary channel error status) bits provide transmit data buffer underflow status for the primary and secondary data paths respectively (it indicates that frame sync signal occurred when the transmit data buffer was empty). The serial port transmits data whenever it detects a framing signal.

- 0 = No frame sync signal occurred when TX data buffer is empty (no underflow).
- 1 = Framing signal occurred when TX buffer was empty (underflow).

Similarly, if SPORT configured as a receiver, the SPORT_CTL_A.DERRPRI and SPORT_CTL_A.DERRSEC bits provide receive overflow status of primary and secondary receive data buffers. In other words, the SPORT indicates that a channel has received new data when the receive buffer is full, so new data overwrites existing data. The serial port receives data whenever it detects a framing signal.

- 0 = No frame sync signal occurred when RX data buffer is full (no overflow).
- 1 = Frame sync signal occurred when RX data buffer was full (overflow).

Besides these status flagging for underflow and overflow errors, each SPORT half contains an error register (SPORT_ERR_A) and a dedicated interrupt channel, known as status interrupt. This interrupt can be optionally triggered based on the error status of primary and secondary data lines as reflected in SPORT_ERR_A.DERRPSTAT and SPORT_ERR_A.DERRSSTAT bits respectively. The SPORT_ERR_A.DERRPMSK (primary channel data error interrupt enable) and SPORT_ERR_A.DERRSMSK (secondary channel data error interrupt enable) bits can be used to unmask the status interrupt for primary and secondary data errors.

In addition to data underflow and data overflow errors, the status interrupt is also triggered optionally when frame sync error is detected. The SPORT_ERR_A.FSERRMSK (frame sync error interrupt enable) bit unmasks the status interrupt for this frame sync error. This frame sync error is generated because of premature frame sync, as explained in "Premature Frame sync error detection" section.

Note that a frame sync error is not detected in following cases.

- When there is no active data transmit/receive and the frame sync pulse occurs due to noise in the input signal.
- If there is an underflow or overflow error. SPORT error logic does not run (the bit count is not set and decremented) if there is an underflow error. Therefore, frame sync errors cannot be detected.
- When the frame sync pulse > system clock period.

The channel error status bits, SPORT_CTL_A.DERRPRI and SPORT_CTL_A.DERRSEC, in the control register are sticky read-only bits, which can be cleared in two ways:

- By disabling the SPORT (for frame sync error) or disabling the corresponding channel by itself (for SPORT_CTL_A.DERRPRI, SPORT_CTL_A.DERRSEC).
- By writing to R/W the SPORT_ERR_A.FSERRSTAT, SPORT_ERR_A.DERRPSTAT or SPORT_ERR_A.DERRSSTAT status bits.

When sticky bits are cleared, interrupts are also cleared.

Interrupts

This section describes the various scenarios in which an interrupt is generated. Both the core and DMA are able to generate data interrupts for receive or transmit operations. Moreover, the SPORT modules generate error conditions which have a separate status interrupt.

Internal Transfer Completion

Each SPORT half has an interrupt associated with it. Both primary and secondary data channels share the same interrupt vector, regardless of whether they are configured as a transmitter or receiver. To determine the source of an interrupt, applications should check the transmit or receive data buffer status (SPORT_CTL_A.DXSPRI, SPORT_CTL_A.DXSSEC) bits. In core mode, this interrupt signifies that either the transmit data buffer is empty (when SPORT_CTL_A.SPTRAN = 1) or new data is available in the receive data buffer (when SPORT_CTL_A.SPTRAN = 0). When serial port data packing is enabled (SPORT_CTL_A.PACK = 1), the transmit and receive interrupts are generated for 32-bit packed words, not for each 16-bit word.

The same interrupt can be used to indicate the completion of the transfer of a block of serial data when the serial ports are configured for DMA. The count register of DMA must be initialized with a word count that specifies the number of words to transfer. The count register decrements after each DMA transfer on the channel. When the word count reaches zero (or if a work unit has finished), the DMA completion interrupt is generated.

Multiple interrupts can occur if both data channels of serial port transmit or receive data in the same cycle. Any interrupt can be masked in the IMASK register.

Transfer Finish Interrupt (TFI)

When a serial port in DMA mode is configured as a transmitter (SPORT_CTL_A.SPTRAN = 1), the Transmit Finish Interrupt feature can be used to signal the end of the transmission in a particular work unit. This feature can be enabled by setting SPORT_CTL_A.TFIENbit. When DMA transfers the last word to the FIFO, it also gives a signal to the SPORT indicating DMA has finished. The SPORT uses this information and then waits until all the data in the FIFO is transmitted out (including the transmit shift register) and generates the Transmit Finish Interrupt. The Interrupt Type field in the DMA Configuration register should be configured for Peripheral Interrupt.

ADSP-BF60x SPORT Register Descriptions

Serial Port (SPORT) contains the following registers.

Table 26-15: ADSP-BF60x SPORT Register List

Name	Description
SPORT_CTL_A	Half SPORT 'A' Control Register
SPORT_DIV_A	Half SPORT 'A' Divisor Register
SPORT_MCTL_A	Half SPORT 'A' Multi-channel Control Register
SPORT_CS0_A	Half SPORT 'A' Multi-channel 0-31 Select Register
SPORT_CS1_A	Half SPORT 'A' Multi-channel 32-63 Select Register
SPORT_CS2_A	Half SPORT 'A' Multi-channel 64-95 Select Register
SPORT_CS3_A	Half SPORT 'A' Multi-channel 96-127 Select Register
SPORT_ERR_A	Half SPORT 'A' Error Register
SPORT_MSTAT_A	Half SPORT 'A' Multi-channel Status Register
SPORT_CTL2_A	Half SPORT 'A' Control 2 Register
SPORT_TXPRI_A	Half SPORT 'A' Tx Buffer (Primary) Register
SPORT_RXPRI_A	Half SPORT 'A' Rx Buffer (Primary) Register

Table 26-15: ADSP-BF60x SPORT Register List (Continued)

Name	Description
SPORT_TXSEC_A	Half SPORT 'A' Tx Buffer (Secondary) Register
SPORT_RXSEC_A	Half SPORT 'A' Rx Buffer (Secondary) Register
SPORT_CTL_B	Half SPORT 'B' Control Register
SPORT_DIV_B	Half SPORT 'B' Divisor Register
SPORT_MCTL_B	Half SPORT 'B' Multi-channel Control Register
SPORT_CS0_B	Half SPORT 'B' Multi-channel 0-31 Select Register
SPORT_CS1_B	Half SPORT 'B' Multi-channel 32-63 Select Register
SPORT_CS2_B	Half SPORT 'B' Multichannel 64-95 Select Register
SPORT_CS3_B	Half SPORT 'B' Multichannel 96-127 Select Register
SPORT_ERR_B	Half SPORT 'B' Error Register
SPORT_MSTAT_B	Half SPORT 'B' Multi-channel Status Register
SPORT_CTL2_B	Half SPORT 'B' Control 2 Register
SPORT_TXPRI_B	Half SPORT 'B' Tx Buffer (Primary) Register
SPORT_RXPRI_B	Half SPORT 'B' Rx Buffer (Primary) Register
SPORT_TXSEC_B	Half SPORT 'B' Tx Buffer (Secondary) Register
SPORT_RXSEC_B	Half SPORT 'B' Rx Buffer (Secondary) Register

Half SPORT 'A' Control Register

The SPORT_CTL_A 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 SPORT_CTL_A 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.

SPORT_CTL_A: Half SPORT 'A' Control Register - R/W

Reset = 0x0000 0000

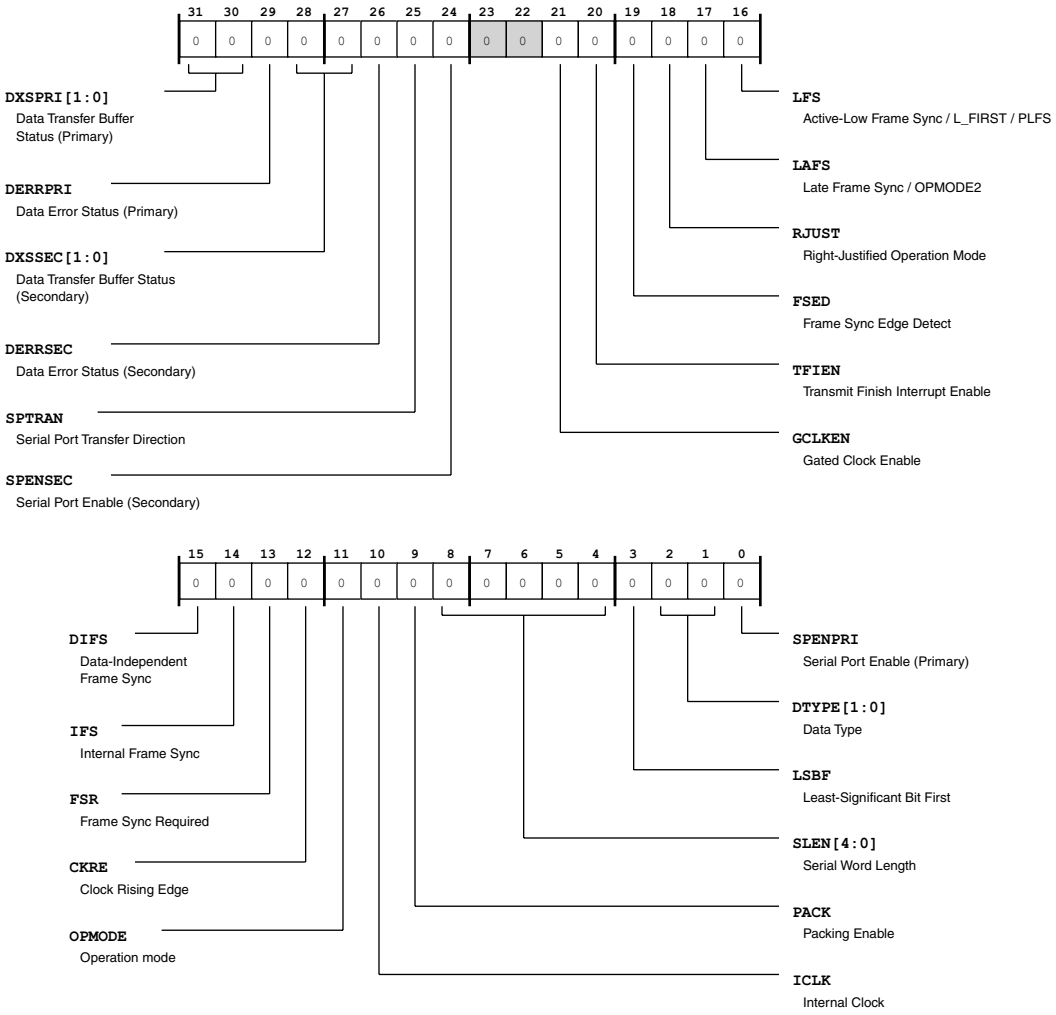


Figure 26-15: SPORT_CTL_A Register Diagram

Table 26-16: SPORT_CTL_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:30 (R/NW)	DXSPRI	Data Transfer Buffer Status (Primary). The SPORT_CTL_A.DXSPRI 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 SPORT_CTL_A.DERRPRI reports the half SPORT's primary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the SPORT_CTL_A.FSR bit =1, SPORT_CTL_A.DERRPRI indicates whether the SPT_AFS signal (from an internal or external source) occurred while the SPORT_TXPRI_A data buffer was empty (during transmit) or the SPORT_RXPRI_A data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the SPT_AFS signal. It is important to note that, as a receiver, the SPORT_CTL_A.DERRPRI indicates when the channel has received new data while the SPORT_RXPRI_A receive buffer is full. This new data overwrites existing data. If the SPORT_CTL_A.FSR bit =0, SPORT_CTL_A.DERRPRI is set whenever the SPORT is required to transmit while the SPORT_TXPRI_A transmit buffer is empty and is set whenever the SPORT is required to receive while the SPORT_RXPRI_A receive buffer is full. The SPORT clears the SPORT_CTL_A.DERRPRI bit if the SPORT_ERR_A.DERRPSTAT bit is cleared.	
		0	No error
		1	Error (Tx underflow or Rx overflow)
28:27 (R/NW)	DXSSEC	Data Transfer Buffer Status (Secondary). The SPORT_CTL_A.DXSSEC indicates the status of the half SPORT's secondary channel data buffer.	
		0	Empty
		1	Reserved
		2	Partially full
		3	Full

Table 26-16: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/NW)	DERRSEC	Data Error Status (Secondary). The SPORT_CTL_A.DERRSEC reports the half SPORT's secondary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the SPORT_CTL_A.FSR bit =1, SPORT_CTL_A.DERRSEC indicates whether the SPT_AFS signal (from an internal or external source) occurred while the SPORT_TXSEC_A data buffer was empty (during transmit) or the SPORT_RXSEC_A data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the SPT_AFS signal. It is important to note that, as a receiver, the SPORT_CTL_A.DERRSEC indicates when the channel has received new data while the SPORT_RXSEC_A receive buffer is full. This new data overwrites existing data. If the SPORT_CTL_A.FSR bit =0, SPORT_CTL_A.DERRSEC is set whenever the SPORT is required to transmit while the SPORT_TXSEC_A transmit buffer is empty and is set whenever the SPORT is required to receive while the SPORT_RXSEC_A receive buffer is full. The SPORT clears the SPORT_CTL_A.DERRSEC bit if the SPORT_ERR_A.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_A.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 SPT_ACLK and SPT_AFS 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 SPT_ACLK and SPT_AFS pins control the transmit shift registers. The receive buffers are inactive when the half SPORT's transfer direction is transmit.	
		0	Receive
		1	Transmit

Table 26-16: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
24 (R/W)	SPENSEC	Serial Port Enable (Secondary). The SPORT_CTL_A.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
21 (R/W)	GCLKEN	Gated Clock Enable. The SPORT_CTL_A.GCLKEN bit enables gated clock operation for the half SPORT when in DSP serial mode or left-justified stereo modes (SPORT_CTL_A.OPMODE = 0 or 1). This bit is ignored when the half SPORT is in right-justified mode (SPORT_CTL_A.RJUST =1) or multi-channel mode (SPORT_MCTL_A.MCE =1). When SPORT_CTL_A.GCLKEN 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 SPORT_CTL_A.TFIEN bit selects when the half SPORT issues its transmission complete interrupt if a DMA complete interrupt is enabled by the DDE_CFG_INT configuration. When enabled (SPORT_CTL_A.TFIEN =1), the DMA complete peripheral interrupt is generated when the last bit of last word in the DMA is shifted out. When disabled (SPORT_CTL_A.TFIEN =0), the DMA interrupt 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

Table 26-16: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	FSED	Frame Sync Edge Detect. The SPORT_CTL_A.FSED bit enables the half SPORT to start transmitting or receiving after detecting an active edge of an external frame sync. The SPORT_CTL_A.FSED 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 (SPORT_CTL_A.FSED =0), 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
18 (R/W)	RJUST	Right-Justified Operation Mode. The SPORT_CTL_A.RJUST bit enables the half SPORT (if SPORT_CTL_A.OPMODE =1) 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 SPORT_MCTL_A.WOFFSET 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 (SPORT_CTL_A.OPMODE =0) or in right-justified mode (SPORT_CTL_A.RJUST =1), the SPORT_CTL_A.LAFS bit selects whether the half SPORT generates a late frame sync (SPT_AFS during first data bit) or generates an early frame sync signal (SPT_AFS during serial clock cycle before first data bit). When the half SPORT is in I2S / left-justified mode (SPORT_CTL_A.OPMODE =1), the SPORT_CTL_A.LAFS bit acts as OPMODE2, selecting whether the half SPORT is in left-justified mode or I2S mode. When the half SPORT is in multi-channel mode (SPORT_MCTL_A.MCE =1), the SPORT_CTL_A.LAFS bit is reserved.	
		0	Early frame sync (or I2S mode)
		1	Late frame sync (or left-justified mode)

Table 26-16: SPORT_CTL_A 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_A.OPMODE =0), the SPORT_CTL_A.LFS bit selects whether the half SPORT uses active low or active high frame sync. When the half SPORT is in I2S / packed / left-justified mode (SPORT_CTL_A.OPMODE =1), the SPORT_CTL_A.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 (multi-channel mode) or right channel first (I2S/packed mode) or left channel first (left-justified mode)
		1	Active low frame sync (DSP standard mode) or falling edge frame sync (multi-channel mode) or left channel first (I2S/packed mode) or right channel first (left-justified mode)
15 (R/W)	DIFS	Data-Independent Frame Sync. The SPORT_CTL_A.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_A.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_A.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_A.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 26-16: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	FSR	Frame Sync Required. The SPORT_CTL_A.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 I2S / packed / left-justified mode (SPORT_CTL_A.OPMODE =1) or is in multi-channel mode (SPORT_MCTL_A.MCE =1).	
		0	No frame sync required
		1	Frame sync required
12 (R/W)	CKRE	Clock Rising Edge. The SPORT_CTL_A.CKRE selects the rising or falling edge of the SPT_ACLK 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 SPT_ACLK. Also note that the transmit and receive related SPORT halves (A and B) should be programmed with the same value for SPORT_CTL_A.CKRE. This programming drives the internally-generated signals on one edge of SPT_ACLK 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_A.OPMODE bit selects whether the half SPORT operates in DSP standard / multi-channel mode or operates in I2S / 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/multi-channel mode
		1	I2S/packed/left-justified mode

Table 26-16: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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 SPT_ACLK clock signal, and the SPT_ACLK is an output. The SPORT_DIV_A.CLKDIV serial clock divisor value determines the clock frequency. For internal clock disabled, the SPT_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
8:4 (R/W)	SLEN	Serial Word Length. The SPORT_CTL_A.SLEN 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_A.SLEN = (serial word length in bits) - 1 For DSP standard mode (SPORT_CTL_A.OPMODE =0), use SPORT_CTL_A.SLEN of 3 to 31 bits. For I2S / packed / left-justified mode (SPORT_CTL_A.OPMODE =1), use SPORT_CTL_A.SLEN of 4 to 31 bits.	

Table 26-16: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	LSBF	Least-Significant Bit First. The SPORT_CTL_A.LSBF 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 SPORT_CTL_A.DTYPE bits selects the data type formatting for the half SPORT's data transfers in DSP standard mode (SPORT_CTL_A.OPMODE =0).	
		0	Right-justify data, zero-fill unused MSBs
		1	Right-justify data, sign-extend unused MSBs
		2	µ-law compand data
		3	A-law compand data
0 (R/W)	SPENPRI	Serial Port Enable (Primary). The SPORT_CTL_A.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 contains divisor values that determine frequencies of internally-generated clocks and frame syncs for half SPORT 'A'.

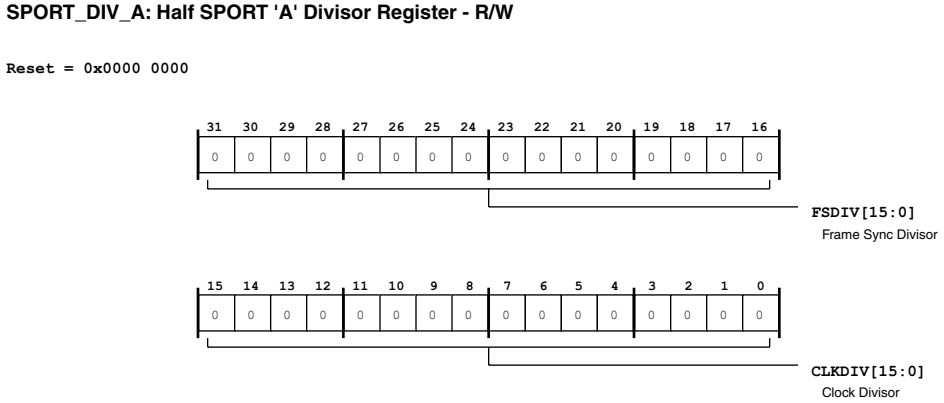


Figure 26-16: SPORT_DIV_A Register Diagram

Table 26-17: SPORT_DIV_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	FSDIV	Frame Sync Divisor. The SPORT_DIV_A.FSDIV 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 SPORT_DIV_A.FSDIV to the number of cycles between frame sync pulses is: SPORT_DIV_A.FSDIV = (number of serial clocks between frame syncs) - 1 Use the following equation to determine the value of SPORT_DIV_A.FSDIV, given the serial clock frequency and desired frame sync frequency: FSDIV = (SCLK ÷ FSCLK) - 1 Note that the frame sync is continuously active when SPORT_DIV_A.FSDIV = 0. The value of SPORT_DIV_A.FSDIV should not be less than the serial word length (SPORT_CTL_A.SLEN), as this may cause an external device to abort the current operation or cause other unpredictable results.
15:0 (R/W)	CLKDIV	Clock Divisor. The SPORT_DIV_A.CLKDIV bits select the divisor that the half SPORT uses to calculate the serial clock (SPT_ACLK) from the processor system clock (SCLK). The divisor is a 16-bit value, allowing a wide range of serial clock rates. When configured for internal clock (SPORT_CTL_A.ICLK =1), legal SPORT_DIV_A.CLKDIV 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 SPORT_DIV_A.CLKDIV: CLKDIV = (SCLK ÷ SPT_ACLK) - 1 For the maximum serial clock frequency, see the processor data sheet.

Half SPORT 'A' Multi-channel Control Register

The SPORT_MCTL_A register controls the half SPORT's multi-channel operations. This register enables multi-channel operation, enables multi-channel data packing, selects the multi-channel frame delay, selects the number of multi-channel slots, and selects the multi-channel window offset size.

SPORT_MCTL_A: Half SPORT 'A' Multi-channel Control Register - R/W

Reset = 0x0000 0000

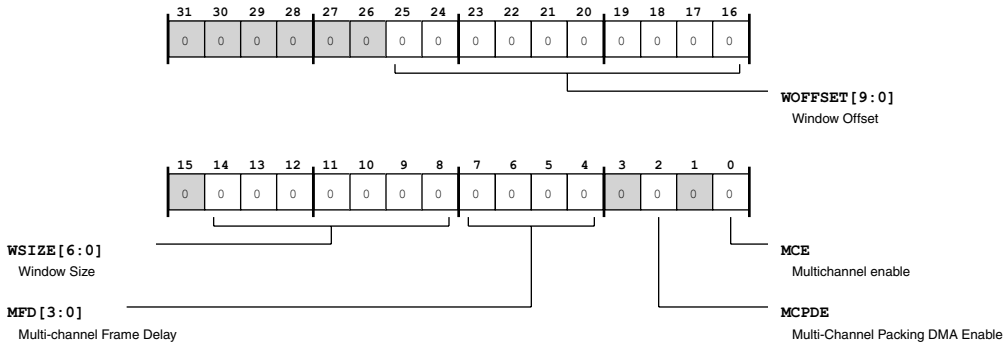


Figure 26-17: SPORT_MCTL_A Register Diagram

Table 26-18: SPORT_MCTL_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:16 (R/W)	WOFFSET	Window Offset. The SPORT_MCTL_A.WOFFSET 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 multi-channel mode is disabled (SPORT_MCTL_A.MCE =0)and right-justified mode is enabled (SPORT_CTL_A.RJUST =1), the least significant 6 bits of SPORT_MCTL_A.WOFFSET 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 SPORT_MCTL_A.WSIZE 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 = (number of channel slots) -1
7:4 (R/W)	MFD	Multi-channel Frame Delay. The SPORT_MCTL_A.MFD bits select the delay (in serial clock cycles) between the half SPORT's multi-channel frame sync pulse and channel 0. The 4-bit field allows selecting Multichannel Frame delay of 0-15 serial clocks.

Table 26-18: SPORT_MCTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	MCPDE	Multi-Channel Packing DMA Enable. The SPORT_MCTL_A.MCPDE bit enables DMA data packing for transmit and enables DMA data unpacking for the half SPORT's multi-channel data transfers.	
		0	Disable
		1	Enable
0 (R/W)	MCE	Multichannel enable. The SPORT_MCTL_A.MCE bit enables multi-channel operations for the half SPORT. The Half SPORT is configured in normal multichannel mode if SPORT_CTL_A.OPMODE=0; while it is configured in Packed mode if SPORT_CTL_A.OPMODE=1. When Configuring in these modes, the Multichannel Enable bit (SPORT_MCTL_A.MCE) should be set before enabling SPORT data channel enable bits (SPORT_CTL_A.SPENPRI and/or SPORT_CTL_A.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_A.DERRPRI and SPORT_CTL_A.DERRSEC bits are cleared.	
		0	Disable
		1	Enable

Half SPORT 'A' Multi-channel 0-31 Select Register

The SPORT_CS0_A register's bits (when set, =1) each corresponds to an active channel for the half SPORT in multi-channel 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 de-activates 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).

SPORT_CS0_A: Half SPORT 'A' Multi-channel 0-31 Select Register - R/W

Reset = 0x0000 0000

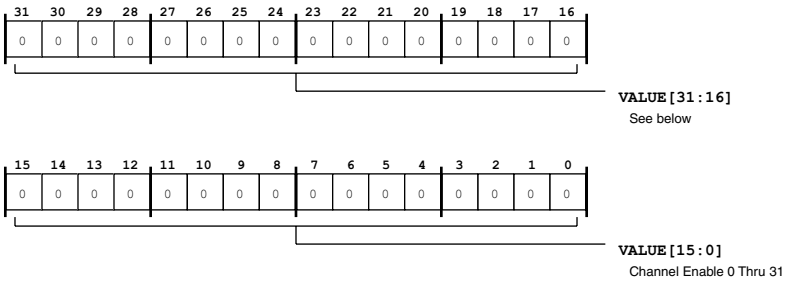


Figure 26-18: SPORT_CS0_A Register Diagram

Table 26-19: SPORT_CS0_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 0 Thru 31.

Half SPORT 'A' Multi-channel 32-63 Select Register

The SPORT_CS1_A register's bits (when set, =1) each corresponds to an active channel for the half SPORT in multi-channel 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 de-activates 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).

SPORT_CS1_A: Half SPORT 'A' Multi-channel 32-63 Select Register - R/W

Reset = 0x0000 0000

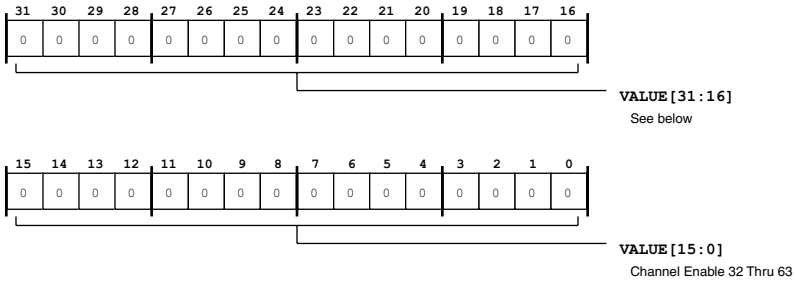


Figure 26-19: SPORT_CS1_A Register Diagram

Table 26-20: SPORT_CS1_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 32 Thru 63.

Half SPORT 'A' Multi-channel 64-95 Select Register

The SPORT_CS2_A register's bits (when set, =1) each corresponds to an active channel for the half SPORT in multi-channel 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 de-activates 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).

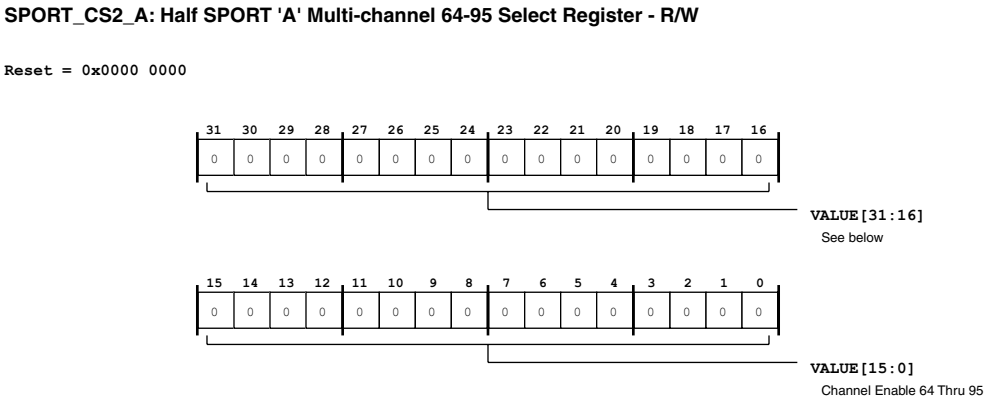


Figure 26-20: SPORT_CS2_A Register Diagram

Table 26-21: SPORT_CS2_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 64 Thru 95.

Half SPORT 'A' Multi-channel 96-127 Select Register

The SPORT_CS3_A register's bits (when set, =1) each corresponds to an active channel for the half SPORT in multi-channel 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 de-activates 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).

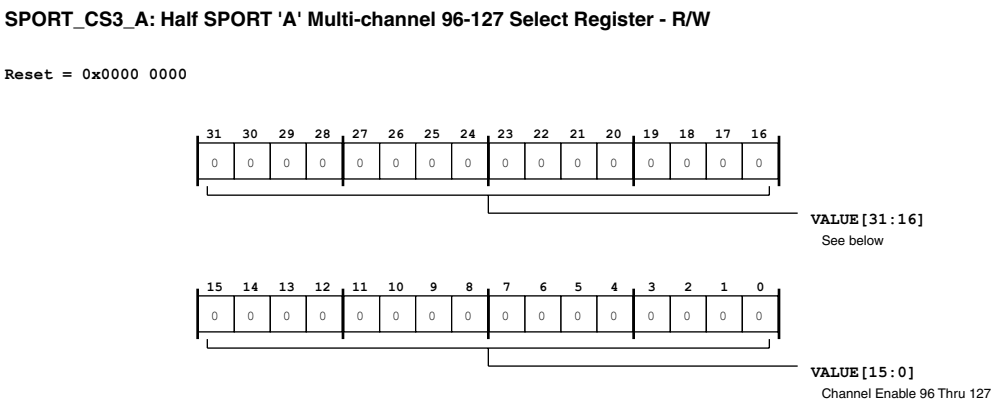


Figure 26-21: SPORT_CS3_A Register Diagram

Table 26-22: SPORT_CS3_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 96 Thru 127.

Half SPORT 'A' Error Register

The SPORT_ERR_A 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.

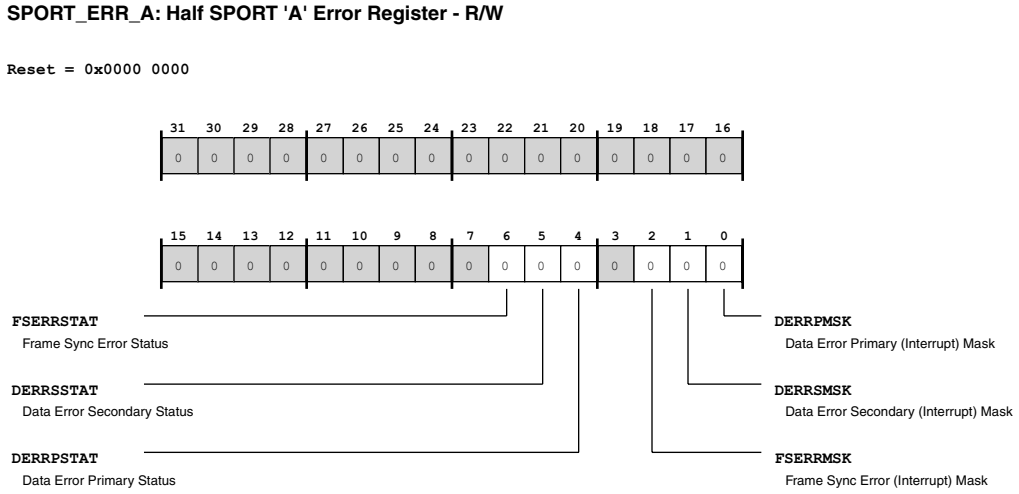


Figure 26-22: SPORT_ERR_A Register Diagram

Table 26-23: SPORT_ERR_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	FSERRSTAT	Frame Sync Error Status. The SPORT_ERR_A.FSERRSTAT 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, SPORT_CTL_A.SLEN = 32). 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 during a zero bit count.	
		0	No error
		1	Error (non-zero bit count at frame sync)
5 (R/W)	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), SPORT_ERR_A.DERRSSTAT indicates transmit underflow status. During receive (SPORT_CTL_A.SPTRAN =0), SPORT_ERR_A.DERRSSTAT indicates receive overflow status. This bit is used to clear the latch of SPORT status interrupt when triggered by 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/W)	DERRPSTAT	Data Error Primary Status. The SPORT_ERR_A.DERRPSTAT bit indicates the error status for the half SPORT's primary channel data buffers. During transmit (SPORT_CTL_A.SPTRAN =1), SPORT_ERR_A.DERRPSTAT indicates transmit underflow status. During receive (SPORT_CTL_A.SPTRAN =0), SPORT_ERR_A.DERRPSTAT indicates receive overflow status. This bit is used to clear the latch of SPORT status interrupt when triggered by Primary Data Error. This bit can also be used to clear the read-only SPORT_CTL_A.DERRPRI status bit.	
		0	No error
		1	Error (transmit underflow or receive overflow)

Table 26-23: SPORT_ERR_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	FSERRMSK	Frame Sync Error (Interrupt) Mask. The SPORT_ERR_A.FSERRMSK unmasks (enables) the half SPORT to generate the frame sync error interrupt.	
		0	Mask (disable)
		1	Unmask (enable)
1 (R/W)	DERRSMSK	Data Error Secondary (Interrupt) Mask. The SPORT_ERR_A.DERRSMSK unmasks (enables) the half SPORT to generate the data error interrupt for the secondary channel.	
		0	Mask (disable)
		1	Unmask (enable)
0 (R/W)	DERRPMSK	Data Error Primary (Interrupt) Mask. The SPORT_ERR_A.DERRPMSK unmasks (enables) the half SPORT to generate the data error interrupt for the primary channel.	
		0	Mask (disable)
		1	Unmask (enable)

Half SPORT 'A' Multi-channel Status Register

The SPORT_MSTAT_A register indicates the current multi-channel being serviced among the half SPORT's active channels in multi-channel 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.

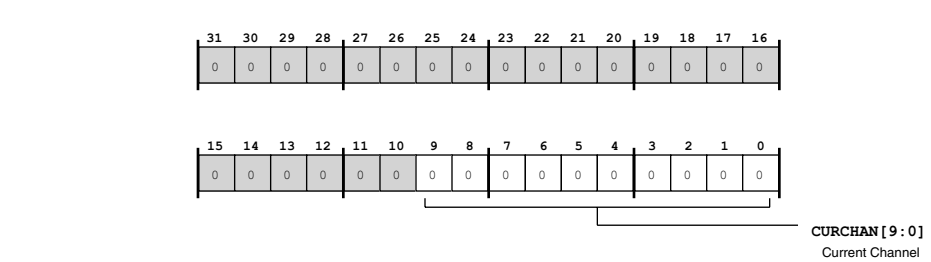


Figure 26-23: SPORT_MSTAT_A Register Diagram

Table 26-24: SPORT_MSTAT_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/NW)	CURCHAN	Current Channel. The SPORT_MSTAT_A.CURCHAN bits indicate the half SPORT's current channel being serviced in multi-channel mode.

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.

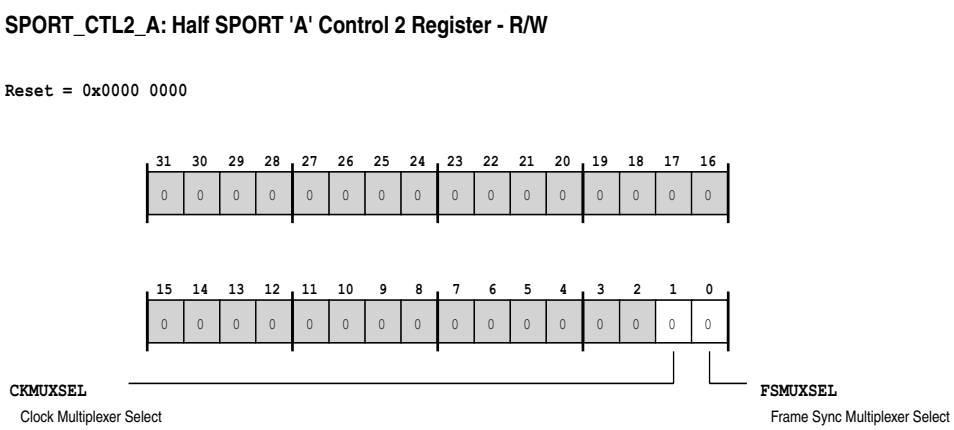


Figure 26-24: SPORT_CTL2_A Register Diagram

Table 26-25: SPORT_CTL2_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	CKMUXSEL	Clock Multiplexer Select. The SPORT_CTL2_A.CKMUXSEL bit enables multiplexing of the half SPORT' 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 SPORT_CTL2_A.CKMUXSEL is enabled, half SPORT 'A' uses SPT_BCLK instead of SPT_ACLK.	
		0	Disable serial clock multiplexing
		1	Enable serial clock multiplexing

Table 26-25: SPORT_CTL2_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	FSMUXSEL	Frame Sync Multiplexer Select. The SPORT_CTL2_A.FSMUXSEL bit enables multiplexing of the half SPORT' 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 SPORT_CTL2_A.FSMUXSEL is enabled, half SPORT 'A' uses SPT_BFS instead of SPT_AFS.	
		0	Disable frame sync multiplexing
		1	Enable frame sync multiplexing

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 may load the data into the buffer (word-by-word process) or the DMA controller may automatically load 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 two location buffer when packing is enabled (SPORT_CTL_A.PACK =1). So depending on PACK bit setting, two 32-bit words or three 32-bit words may 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 may issue an interrupt (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 does not occur when the half SPORT is executing a DMA-based transfer.

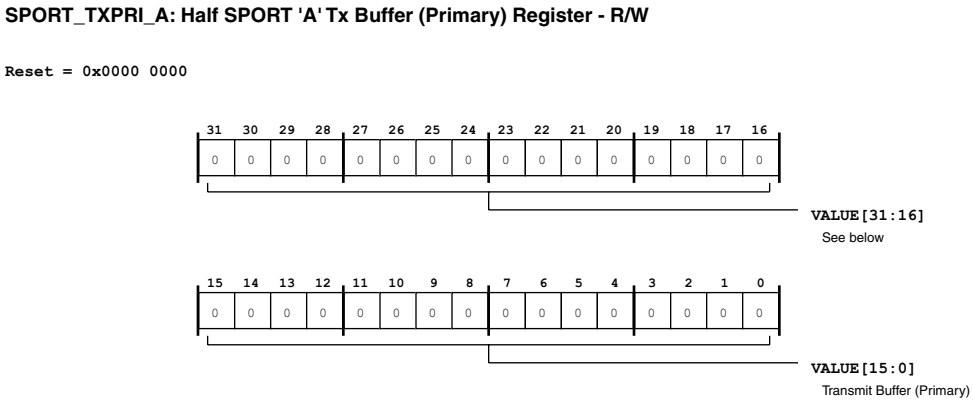


Figure 26-25: SPORT_TXPRI_A Register Diagram

Table 26-26: SPORT_TXPRI_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Primary). The SPORT_TXPRI_A.VALUE bits hold the half SPORT's primary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_A.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_A and SPORT_MCTL_A register descriptions.

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 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 DMA'd into processor memory using DMA controller. With a data buffer and input shift register, the SPORT_RXPRI_A register acts as a two-location buffer. So, the SPORT can keep Two 32-bit received words at max in any one time (independent of the SPORT_CTL_A.PACK bit setting).

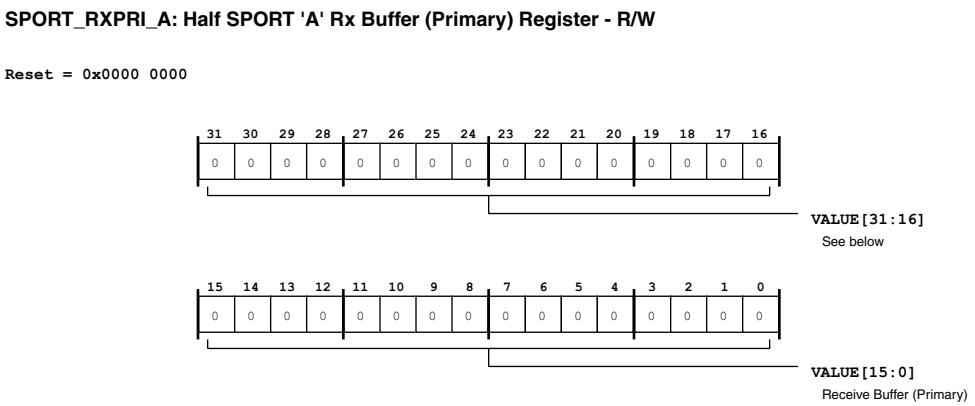


Figure 26-26: SPORT_RXPRI_A Register Diagram

Table 26-27: SPORT_RXPRI_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Primary). The SPORT_RXPRI_A.VALUE bits hold the half SPORT's primary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_A.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_A and SPORT_MCTL_A 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 may load the data into the buffer (word-by-word process) or the DMA controller may automatically load 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 two location buffer when packing is enabled (SPORT_CTL_A.PACK =1). So depending on PACK bit setting, two 32-bit words or three 32-bit words may 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 may issue an interrupt (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 does not occur when the half SPORT is executing a DMA-based transfer.

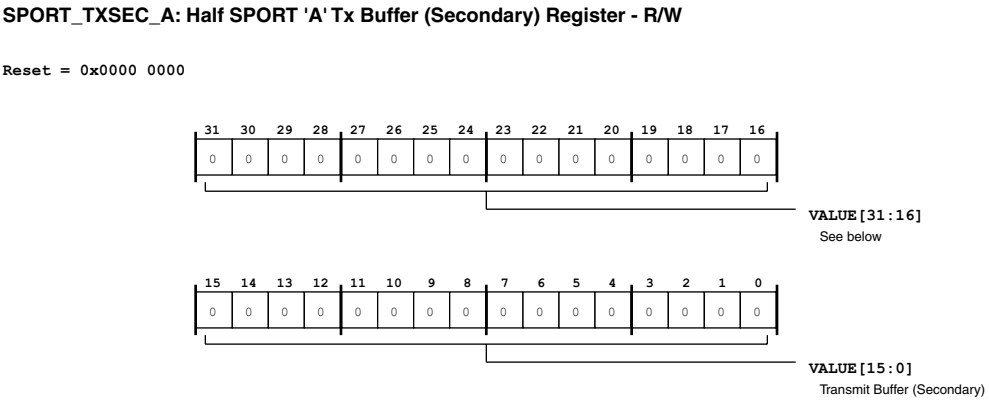


Figure 26-27: SPORT_TXSEC_A Register Diagram

Table 26-28: SPORT_TXSEC_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Secondary). The SPORT_TXSEC_A.VALUE bits hold the half SPORT's secondary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_A.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_A and SPORT_MCTL_A register descriptions.

Half SPORT 'A' Rx Buffer (Secondary) Register

The SPORT_RXSEC_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 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 DMA'd into processor memory using DMA controller. With a data buffer and input shift register, the SPORT_RXSEC_A register acts as a two-location buffer. So, the SPORT can keep Two 32-bit received words at max in any one time (independent of the SPORT_CTL_A.PACK bit setting).

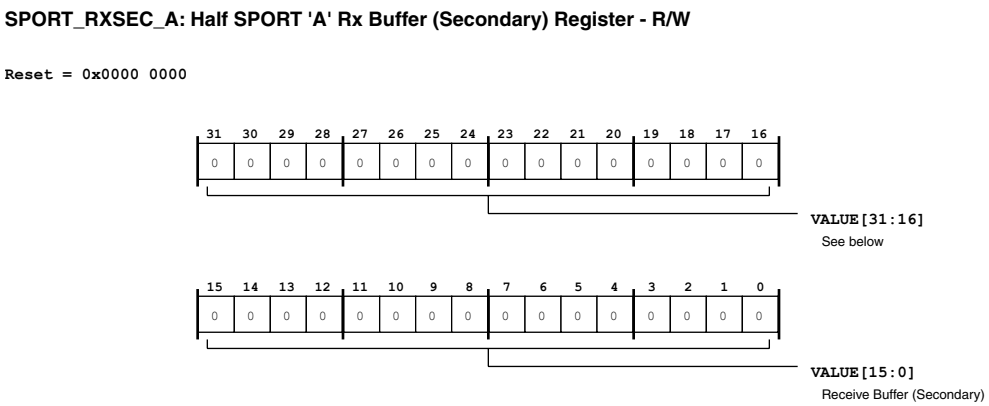


Figure 26-28: SPORT_RXSEC_A Register Diagram

Table 26-29: SPORT_RXSEC_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Secondary). The SPORT_RXSEC_A.VALUE bits hold the half SPORT's secondary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_A.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_A and SPORT_MCTL_A register descriptions.

Half SPORT 'B' Control Register

The SPORT_CTL_B 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 SPORT_CTL_B 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.

SPORT_CTL_B: Half SPORT 'B' Control Register - R/W

Reset = 0x0000 0000

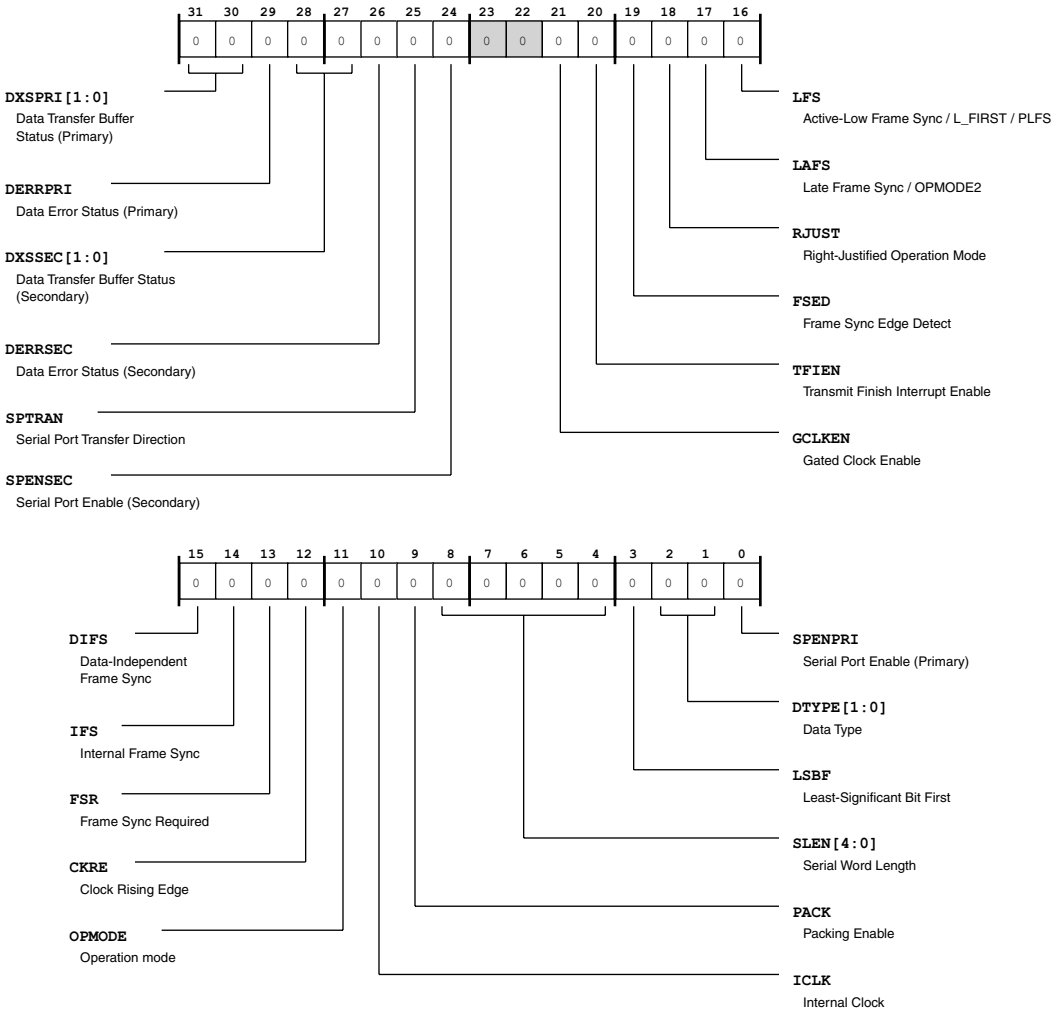


Figure 26-29: SPORT_CTL_B Register Diagram

Table 26-30: SPORT_CTL_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:30 (R/NW)	DXSPRI	Data Transfer Buffer Status (Primary). The SPORT_CTL_B.DXSPRI 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 SPORT_CTL_B.DERRPRI reports the half SPORT's primary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the SPORT_CTL_B.FSR bit =1, SPORT_CTL_B.DERRPRI indicates whether the SPT_BFS signal (from an internal or external source) occurred while the SPORT_TXPRI_B data buffer was empty (during transmit) or the SPORT_RXPRI_B data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the SPT_BFS signal. It is important to note that, as a receiver, the SPORT_CTL_B.DERRPRI indicates when the channel has received new data while the SPORT_RXPRI_B receive buffer is full. This new data overwrites existing data. If the SPORT_CTL_B.FSR bit =0, SPORT_CTL_B.DERRPRI is set whenever the SPORT is required to transmit while the SPORT_TXPRI_B transmit buffer is empty and is set whenever the SPORT is required to receive while the SPORT_RXPRI_B receive buffer is full. The SPORT clears the SPORT_CTL_B.DERRPRI bit if the SPORT_ERR_B.DERRPSTAT bit is cleared.	
		0	No error
		1	Error (Tx underflow or Rx overflow)
28:27 (R/NW)	DXSSEC	Data Transfer Buffer Status (Secondary). The SPORT_CTL_B.DXSSEC indicates the status of the half SPORT's secondary channel data buffer.	
		0	Empty
		1	Reserved
		2	Partially full
		3	Full

Table 26-30: 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 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, SPORT_CTL_B.DERRSEC indicates whether the SPT_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 SPT_BFS signal. It is important to note that, as a receiver, the SPORT_CTL_B.DERRSEC 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, SPORT_CTL_B.DERRSEC is set whenever the SPORT is required to transmit while the SPORT_TXSEC_B transmit buffer is empty and is 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 SPT_BCLK and SPT_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 SPT_BCLK and SPT_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

Table 26-30: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
21 (R/W)	GCLKEN	Gated Clock Enable. The SPORT_CTL_B.GCLKEN bit enables gated clock operation for the half SPORT when in DSP serial mode or left-justified stereo modes (SPORT_CTL_B.OPMODE = 0 or 1). This bit is ignored when the half SPORT is in right-justified mode (SPORT_CTL_B.RJUST =1) or multi-channel mode (SPORT_MCTL_B.MCE =1). When SPORT_CTL_B.GCLKEN 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 SPORT_CTL_B.TFIEN bit selects when the half SPORT issues its transmission complete interrupt if a DMA complete interrupt is enabled by the DDE_CFG_INT configuration. When enabled (SPORT_CTL_B.TFIEN =1), the DMA complete peripheral interrupt is generated when the last bit of last word in the DMA is shifted out. When disabled (SPORT_CTL_B.TFIEN =0), the DMA interrupt 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

Table 26-30: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	FSED	Frame Sync Edge Detect. The SPORT_CTL_B.FSED bit enables the half SPORT to start transmitting or receiving after detecting an active edge of an external frame sync. The SPORT_CTL_B.FSED 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 (SPORT_CTL_B.FSED =0), 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
18 (R/W)	RJUST	Right-Justified Operation Mode. The SPORT_CTL_B.RJUST bit enables the half SPORT (if SPORT_CTL_B.OPMODE =1) 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 SPORT_MCTL_B.WOFFSET field). For information about appropriate delay selections, see the SPORT operating modes section.	
		0	Disable
		1	Enable

Table 26-30: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	LAFS	Late Frame Sync / OPMODE2. When the half SPORT is in DSP standard mode (SPORT_CTL_B.OPMODE =0) or in right-justified mode (SPORT_CTL_B.RJUST =1), the SPORT_CTL_B.LAFS bit selects whether the half SPORT generates a late frame sync (SPT_BFS during first data bit) or generates an early frame sync signal (SPT_BFS during serial clock cycle before first data bit). When the half SPORT is in I2S / left-justified mode (SPORT_CTL_B.OPMODE =1), the SPORT_CTL_B.LAFS bit acts as OPMODE2, selecting whether the half SPORT is in left-justified mode or I2S mode. When the half SPORT is in multi-channel mode (SPORT_MCTL_B.MCE =1), the SPORT_CTL_B.LAFS bit is reserved.	
		0	Early frame sync (or I2S 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 (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 I2S / 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 (multi-channel mode) or right channel first (I2S/packed mode) or left channel first (left-justified mode)
		1	Active low frame sync (DSP standard mode) or falling edge frame sync (multi-channel mode) or left channel first (I2S/packed mode) or right channel first (left-justified mode)

Table 26-30: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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 I2S / packed / left-justified mode (SPORT_CTL_B.OPMODE =1) or is in multi-channel mode (SPORT_MCTL_B.MCE =1).	
		0	No frame sync required
		1	Frame sync required

Table 26-30: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	CKRE	Clock Rising Edge. The SPORT_CTL_B.CKRE selects the rising or falling edge of the SPT_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 SPT_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 SPT_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 / multi-channel mode or operates in I2S / 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/multi-channel mode
		1	I2S/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 SPT_BCLK clock signal, and the SPT_BCLK is an output. The SPORT_DIV_B.CLKDIV serial clock divisor value determines the clock frequency. For internal clock disabled, the SPT_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 26-30: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	PACK	Packing Enable. The SPORT_CTL_B.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
8:4 (R/W)	SLEN	Serial Word Length. The SPORT_CTL_B.SLEN 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 = (serial word length in bits) - 1 For DSP standard mode (SPORT_CTL_B.OPMODE =0), use SPORT_CTL_B.SLEN of 3 to 31 bits. For I2S / packed / left-justified mode (SPORT_CTL_B.OPMODE =1), use SPORT_CTL_B.SLEN of 4 to 31 bits.	
3 (R/W)	LSBF	Least-Significant Bit First. The SPORT_CTL_B.LSBF 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)

Table 26-30: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2:1 (R/W)	DTYPE	Data Type. The SPORT_CTL_B.DTYPE bits selects the data type formatting for the half SPORT's data transfers in DSP standard mode (SPORT_CTL_B.OPMODE =0).	
		0	Right-justify data, zero-fill unused MSBs
		1	Right-justify data, sign-extend unused MSBs
		2	µ-law compand data
		3	A-law compand data
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 'B' Divisor Register

The SPORT_DIV_B contains divisor values that determine frequencies of internally-generated clocks and frame syncs for SPORT half 'B'.

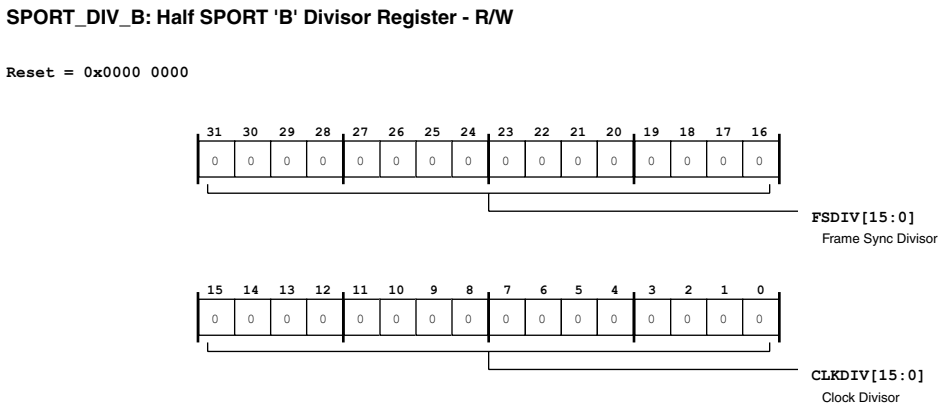


Figure 26-30: SPORT_DIV_B Register Diagram

Table 26-31: SPORT_DIV_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	FSDIV	Frame Sync Divisor. The SPORT_DIV_B.FSDIV 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 SPORT_DIV_B.FSDIV to the number of cycles between frame sync pulses is: SPORT_DIV_B.FSDIV = (number of serial clocks between frame syncs) - 1 Use the following equation to determine the value of SPORT_DIV_B.FSDIV, given the serial clock frequency and desired frame sync frequency: FSDIV = (SCLK ÷ FSCLK) - 1 Note that the frame sync is continuously active when SPORT_DIV_B.FSDIV = 0. The value of SPORT_DIV_B.FSDIV should not be less than the serial word length (SPORT_CTL_B.SLEN), as this may cause an external device to abort the current operation or cause other unpredictable results.
15:0 (R/W)	CLKDIV	Clock Divisor. The SPORT_DIV_B.CLKDIV bits select the divisor that the half SPORT uses to calculate the serial clock (SPT_BCLK) from the processor system clock (SCLK). The divisor is a 16-bit value, allowing a wide range of serial clock rates. When configured for internal clock (SPORT_CTL_B.ICLK =1), legal SPORT_DIV_B.CLKDIV 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 SPORT_DIV_B.CLKDIV: CLKDIV = (SCLK ÷ SPT_BCLK) - 1 For the maximum serial clock frequency, see the processor data sheet.

Half SPORT 'B' Multi-channel Control Register

The SPORT_MCTL_B register controls the half SPORT's multi-channel operations. This register enables multi-channel operation, enables multi-channel data packing, selects the multi-channel frame delay, selects the number of multi-channel slots, and selects the multi-channel window offset size.

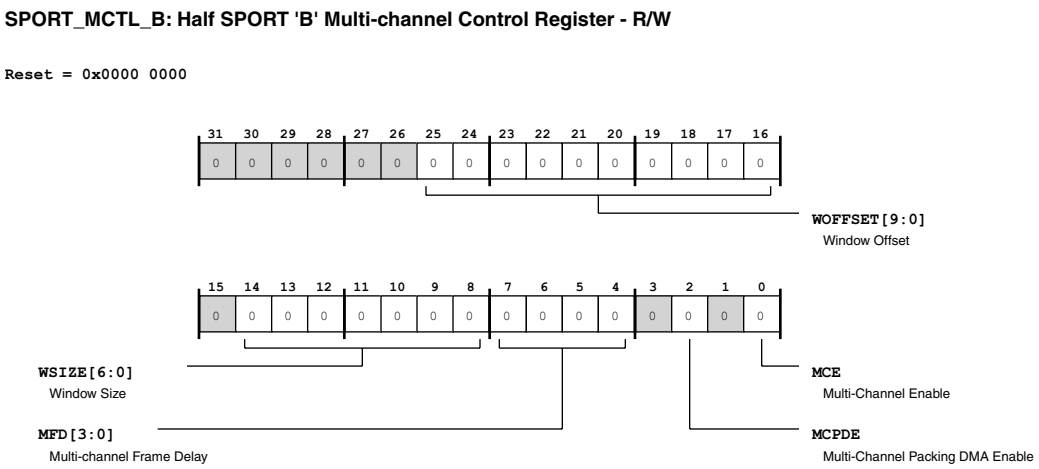


Figure 26-31: SPORT_MCTL_B Register Diagram

Table 26-32: SPORT_MCTL_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:16 (R/W)	WOFFSET	Window Offset. The SPORT_MCTL_B.WOFFSET 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 multi-channel mode is disabled (SPORT_MCTL_B.MCE =0)and right-justified mode is enabled (SPORT_CTL_B.RJUST =1), the least significant 6 bits of SPORT_MCTL_B.WOFFSET 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 SPORT_MCTL_B.WSIZE 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 = (number of channel slots) -1
7:4 (R/W)	MFD	Multi-channel Frame Delay. The SPORT_MCTL_B.MFD bits select the delay (in serial clock cycles) between the half SPORT's multi-channel frame sync pulse and channel 0. The 4-bit field allows selecting Multichannel Frame delay of 0-15 serial clocks.

Table 26-32: SPORT_MCTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	MCPDE	Multi-Channel 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 multi-channel data transfers.	
		0	Disable
		1	Enable
0 (R/W)	MCE	Multi-Channel Enable. The SPORT_MCTL_B.MCE bit enables multi-channel 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 'B' Multi-channel 0-31 Select Register

The SPORT_CS0_B register's bits (when set, =1) each corresponds to an active channel for the half SPORT in multi-channel 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 de-activates 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).

SPORT_CS0_B: Half SPORT 'B' Multi-channel 0-31 Select Register - R/W

Reset = 0x0000 0000

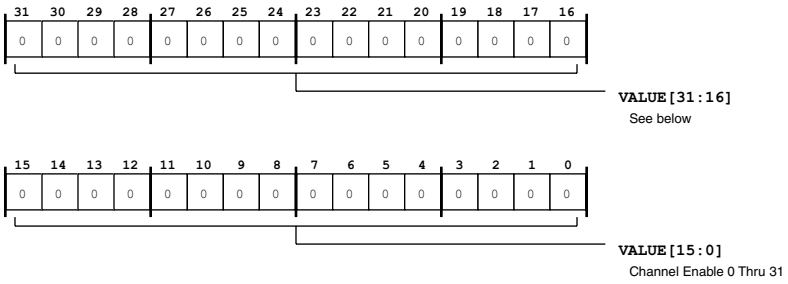


Figure 26-32: SPORT_CS0_B Register Diagram

Table 26-33: SPORT_CS0_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 0 Thru 31.

Half SPORT 'B' Multi-channel 32-63 Select Register

The SPORT_CS1_B register's bits (when set, =1) each corresponds to an active channel for the half SPORT in multi-channel 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 de-activates 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).

SPORT_CS1_B: Half SPORT 'B' Multi-channel 32-63 Select Register - R/W

Reset = 0x0000 0000

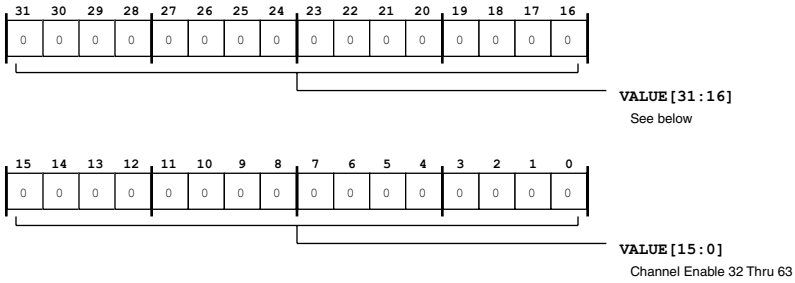


Figure 26-33: SPORT_CS1_B Register Diagram

Table 26-34: SPORT_CS1_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 32 Thru 63.

Half SPORT 'B' Multichannel 64-95 Select Register

The SPORT_CS2_B register's bits (when set, =1) each corresponds to an active channel for the half SPORT in multi-channel 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 de-activates 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).

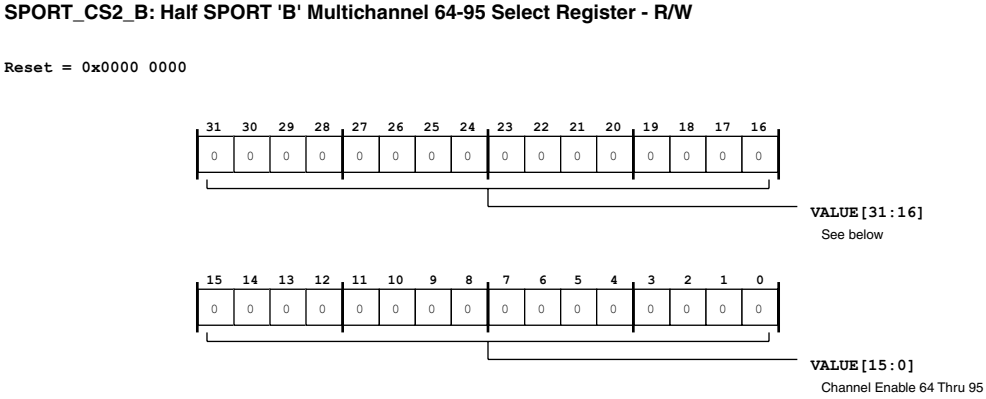


Figure 26-34: SPORT_CS2_B Register Diagram

Table 26-35: SPORT_CS2_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 64 Thru 95.

Half SPORT 'B' Multichannel 96-127 Select Register

The SPORT_CS3_B register's bits (when set, =1) each corresponds to an active channel for the half SPORT in multi-channel 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 de-activates 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).

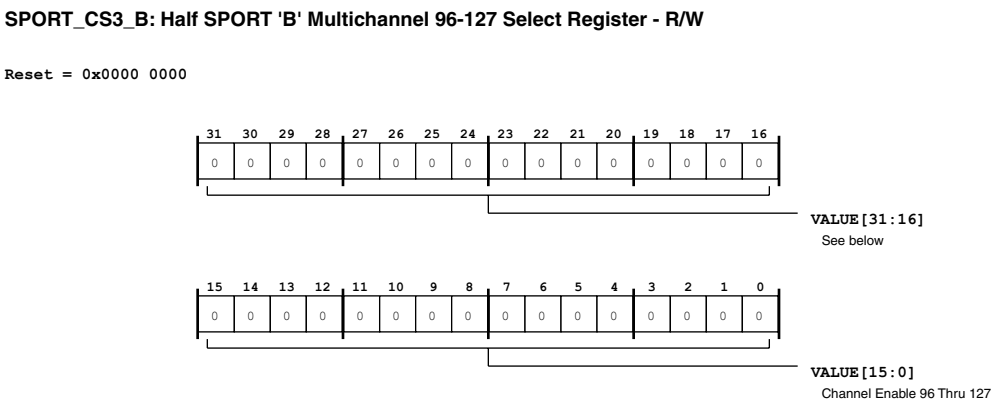


Figure 26-35: SPORT_CS3_B Register Diagram

Table 26-36: SPORT_CS3_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 96 Thru 127.

Half SPORT 'B' Error Register

The SPORT_ERR_B 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.

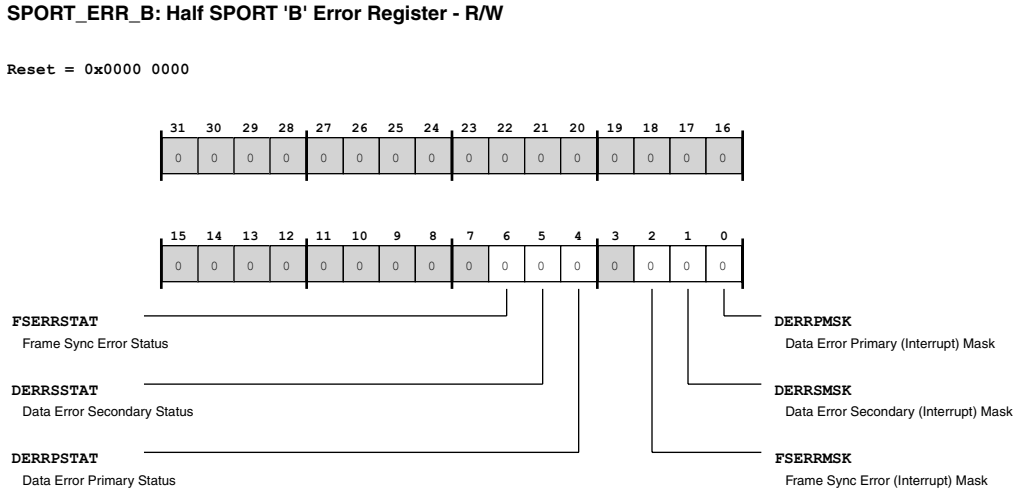


Figure 26-36: SPORT_ERR_B Register Diagram

Table 26-37: SPORT_ERR_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	FSERRSTAT	Frame Sync Error Status. The SPORT_ERR_B.FSERRSTAT 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, SPORT_CTL_B.SLEN = 32). 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 during a zero bit count.	
		0	No error
		1	Error (non-zero bit count at frame sync)
5 (R/W)	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 transmit underflow status. During receive (SPORT_CTL_B.SPTRAN =0), SPORT_ERR_B.DERRSSTAT indicates receive overflow status. This bit is used to clear the latch of SPORT status interrupt when triggered by 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/W)	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), SPORT_ERR_B.DERRPSTAT indicates transmit underflow status. During receive (SPORT_CTL_B.SPTRAN =0), SPORT_ERR_B.DERRPSTAT indicates receive overflow status. This bit is used to clear the latch of SPORT status interrupt when triggered by 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)

Table 26-37: SPORT_ERR_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	FSERRMSK	Frame Sync Error (Interrupt) Mask. The SPORT_ERR_B.FSERRMSK unmasks (enables) the half SPORT to generate the frame sync error interrupt.	
		0	Mask (disable)
		1	Unmask (enable)
1 (R/W)	DERRSMSK	Data Error Secondary (Interrupt) Mask. The SPORT_ERR_B.DERRSMSK unmasks (enables) the half SPORT to generate the data error interrupt for the secondary channel.	
		0	Mask (disable)
		1	Unmask (enable)
0 (R/W)	DERRPMSK	Data Error Primary (Interrupt) Mask. The SPORT_ERR_B.DERRPMSK unmasks (enables) the half SPORT to generate the data error interrupt for the primary channel.	
		0	Mask (disable)
		1	Unmask (enable)

Half SPORT 'B' Multi-channel Status Register

The SPORT_MSTAT_B register indicates the current multi-channel being serviced among the half SPORT's active channels in multi-channel 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.

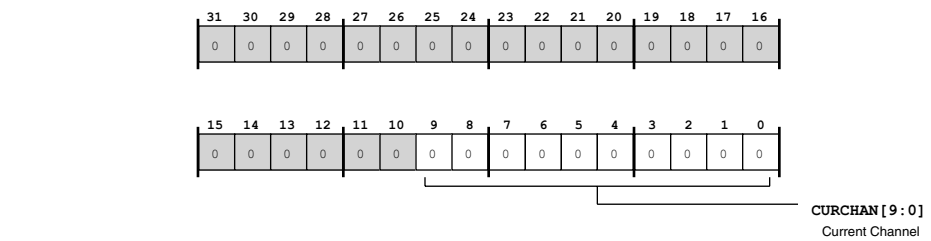


Figure 26-37: SPORT_MSTAT_B Register Diagram

Table 26-38: SPORT_MSTAT_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/NW)	CURCHAN	Current Channel. The SPORT_MSTAT_B.CURCHAN bits indicate the half SPORT's current channel being serviced in multi-channel mode.

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.

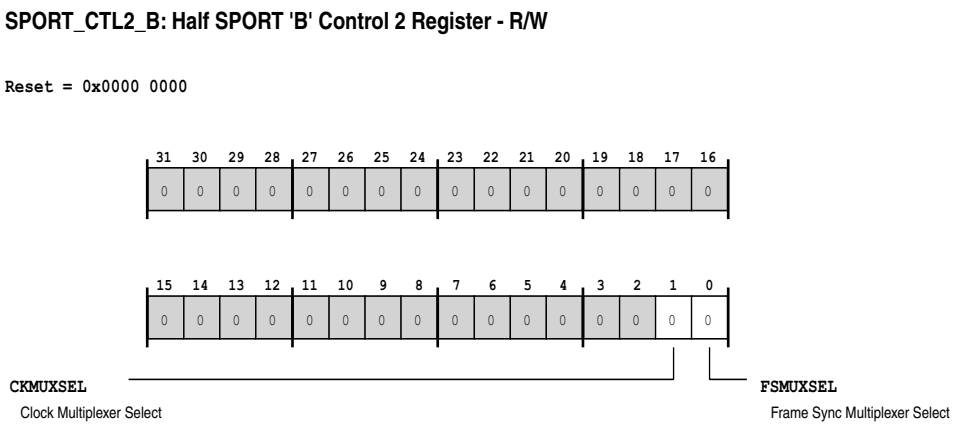


Figure 26-38: SPORT_CTL2_B Register Diagram

Table 26-39: SPORT_CTL2_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	CKMUXSEL	Clock Multiplexer Select. The SPORT_CTL2_B.CKMUXSEL bit enables multiplexing of the half SPORT' 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 SPORT_CTL2_B.CKMUXSEL is enabled, half SPORT 'B' uses SPT_ACLK instead of SPT_BCLK.	
		0	Disable serial clock multiplexing
		1	Enable serial clock multiplexing

Table 26-39: SPORT_CTL2_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	FSMUXSEL	Frame Sync Multiplexer Select. The SPORT_CTL2_B.FSMUXSEL bit enables multiplexing of the half SPORT' 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 SPORT_CTL2_B.FSMUXSEL is enabled, half SPORT 'B' uses SPT_AFS instead of SPT_BFS.	
		0	Disable frame sync multiplexing
		1	Enable frame sync multiplexing

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 may load the data into the buffer (word-by-word process) or the DMA controller may automatically load 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 two location buffer when packing is enabled (SPORT_CTL_B.PACK =1). So depending on PACK bit setting, two 32-bit words or three 32-bit words may 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 may issue an interrupt (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 does not occur when the half SPORT is executing a DMA-based transfer.

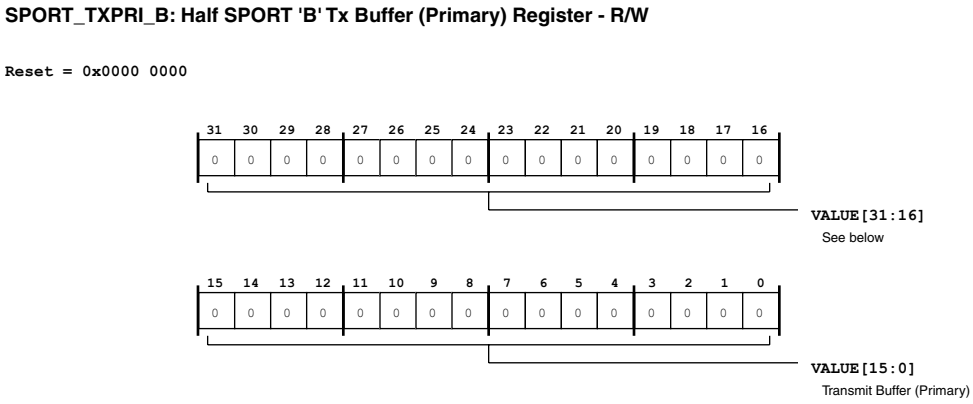


Figure 26-39: SPORT_TXPRI_B Register Diagram

Table 26-40: SPORT_TXPRI_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Primary). The SPORT_TXPRI_B.VALUE bits hold the half SPORT's primary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_B.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_B and SPORT_MCTL_B 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 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 DMA'd into processor memory using DMA controller. With a data buffer and input shift register, the SPORT_RXPRI_B register acts as a two-location buffer. So, the SPORT can keep Two 32-bit received words at max in any one time (independent of the SPORT_CTL_A.PACK bit setting).

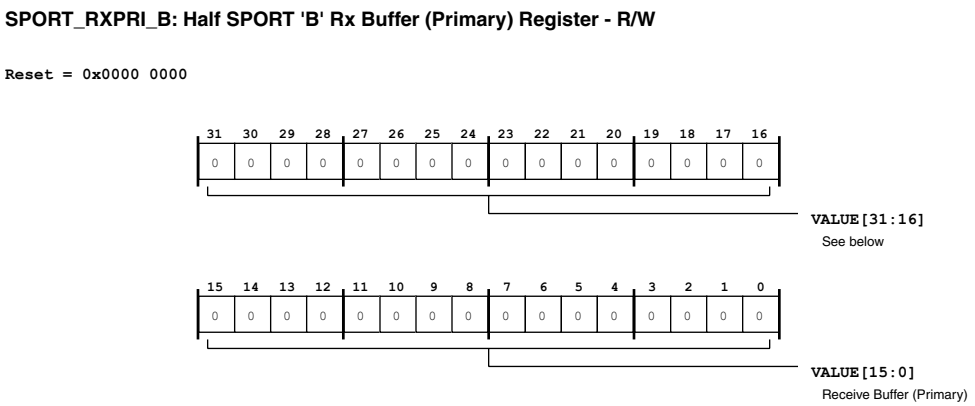


Figure 26-40: SPORT_RXPRI_B Register Diagram

Table 26-41: SPORT_RXPRI_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Primary). The SPORT_RXPRI_B.VALUE bits hold the half SPORT's primary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_B.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_B and SPORT_MCTL_B 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 may load the data into the buffer (word-by-word process) or the DMA controller may automatically load 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 PACK bit setting, two 32-bit words or three 32-bit words may 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 may issue an interrupt (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 does not occur when the half SPORT is executing a DMA-based transfer.

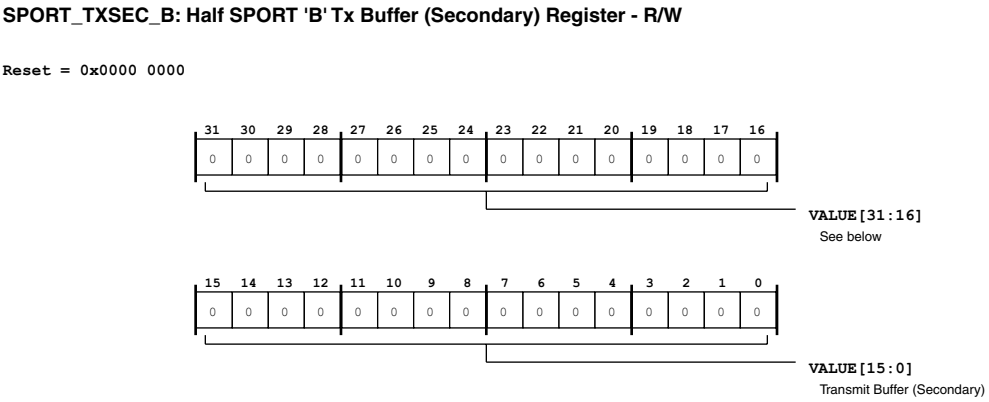


Figure 26-41: SPORT_TXSEC_B Register Diagram

Table 26-42: SPORT_TXSEC_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Secondary). The SPORT_TXSEC_B.VALUE bits hold the half SPORT's secondary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_B.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_B and SPORT_MCTL_B register descriptions.

Half SPORT 'B' Rx Buffer (Secondary) Register

The SPORT_RXSEC_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 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 DMA'd into processor memory using DMA controller. With a data buffer and input shift register, the SPORT_RXSEC_B register acts as a two-location buffer. So, the SPORT can keep Two 32-bit received words at max in any one time (independent of the SPORT_CTL_A.PACK bit setting).

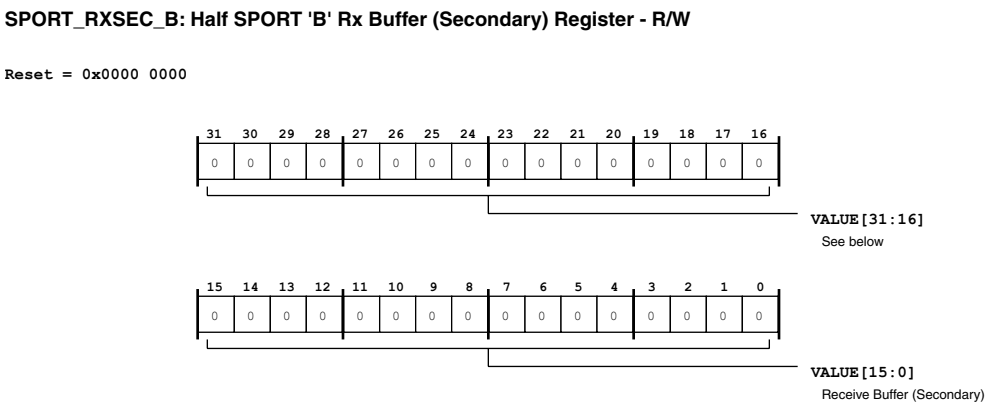


Figure 26-42: SPORT_RXSEC_B Register Diagram

Table 26-43: SPORT_RXSEC_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Secondary). The SPORT_RXSEC_B.VALUE bits hold the half SPORT's secondary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_B.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_B and SPORT_MCTL_B register descriptions.

27 ADC Control Module (ACM)

The processor includes an ADC control module (ACM) that provides an interface that synchronizes the controls between the processor and an Analog-to-Digital Converter (ADC). The analog-to-digital conversions are initiated by the processor, based on either external or internal events.

Traditionally, ADC sampling uses processor interrupts (initiated by the events) and the interrupt service routine programming of the appropriate peripheral (usually SPORT or SPI) for initiating the ADC conversion process. This traditional approach has some limiting factors:

- The ADC sampling instances are not precisely controlled due to interrupt latencies (which can vary) or due to variable instruction execution cycles
- Consumption of processor MIPS can be prohibitive, especially for high frequency of conversion related events.
- If the ADC requires any control signals (such as channel select pins, ADC mode select, ADC Range pin) with some specified Set-up, Hold or Zero time requirements with respect to sampling time, then providing such signals with GP flags may be difficult to implement in the application.

The ADC control module (ACM) answers the limitations of the traditional approach to sampling by providing a dedicated hardware, which samples the events and provides sampling signals with required timings to the ADC in real-time. It permits flexible scheduling of sampling instants and provides precise sampling signals to the ADC. The ACM approach both saves processor bandwidth and provides precise control for ADC sampling time. Furthermore, the processor can be interfaced directly to many ADCs without any glue logic required.

The ACM synchronizes the ADC conversion process (by providing ADC clock, ADC conversion start signal, and related ADC controls), but the actual data acquisition from the ADC is accomplished by other peripheral such as SPORT. On the ADSP-BF60x processors, the ADC module can be used in conjunction with either halves of SPORT1. The processor does not support ACM operation with the SPI. The following figure shows how an ADC can be interfaced using ACM and SPORT peripherals of processor.

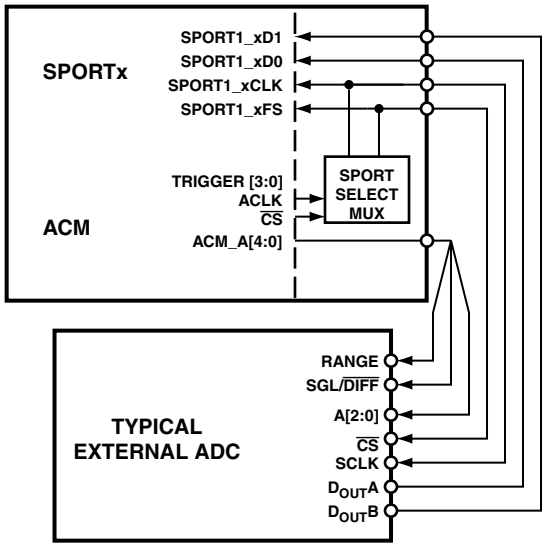


Figure 27-1: ADC/SPORT Interface

NOTE: The ADSP-BF60x processor does not include an on-chip, internal ADC.

ACM Features

The ADC Control Module (ACM) offers following features and capabilities:

- Provides serial clock, chip select and five general-purpose control lines to control the ADC operations. Internally routes clock and CS signals to selected serial port.
- ACM can accept four trigger inputs, based on which it can precisely initiate the ADC sampling events. The Trigger inputs may be internally generated or externally supplied. Further, polarity of trigger inputs is configurable.
- ACM can handle 16 ADC sampling events per valid trigger received. Each event can be independently programmed to specify when to initiate ADC sampling with respect to trigger input.
- Two independent 32-bit ACM Timers can be used to divide 16 events into two groups of 8 events each.
- Automatically stops the ACM Timer after completion of associated events in order to save the power.
- Four-deep pending FIFO to queue the active events when ACM is busy.
- ACM can internally generate serial clock up to `SCLK/2` rate. Improved granularity for internal clock generation, allowing both odd and even `SCLK:ACLK` ratios.
- ACM clock can be gated (active only during enabled events) to interface it with SPI-compatible ADC's.
- When initiating ADC sampling cycle, the width of chip select signal-which can be also used by ADC as start of conversion-, can be configured from 1 ACLK to 256 ACLKs. Further the polarity of this signal can be configured as active-high or active-low signal.

- Auto ACLK adjustment at the time of CS assertion. After assertion of CS signal, the first edge of ACLK can be configured to be either rising edge or falling edge.
- The five general-purpose control lines provided by ACM can be programmed for required Set-up and Hold time with respect to ADC sampling cycle. Additionally, Zero time can be inserted between two successive sampling cycles.
- ACM provides 16 Event Order registers (one per each event) which indicates the order at which events were handled. Optionally these registers can be automatically cleared by ACM's trigger input.
- The ACM hardware flags the appropriate event completion status bit upon completion of an event. If an event got missed, appropriate event missed status bit is flagged. Each event has separate bits. Optionally Event Completion Interrupt and Event Missed Interrupt can be triggered upon these respective conditions.
- Predictable latency between the internal occurrence of an event and the assertion of a sampling event.
- ACM can operate as Trigger Master to provide signal to TRU upon completion of events.

ACM Functional Description

The ADC Control Module uses internal ACM timers and the event time register to create events. The user has to enable one of the timer (or both timers) for the ACM operation. Appropriate event control register and event time register values also have to be programmed. After receiving valid trigger on selected trigger input, the timer starts counting. If at anytime the timer count matches with the time specified in the event time register (ACM_ETx) of an enabled event (associated with that timer), the comparators generate an active event signal to the timing generation unit to start the ADC access. The counter continues counting, and for each matching with enabled event time, the ACM gives an event signal to the timing generation unit.

Figure shows the ACM operation where only two events (`Event0` and `Event3`) are enabled. The line labeled "ADC Controls" depicts the timing of the ADC control signals: `A[4:0]`.

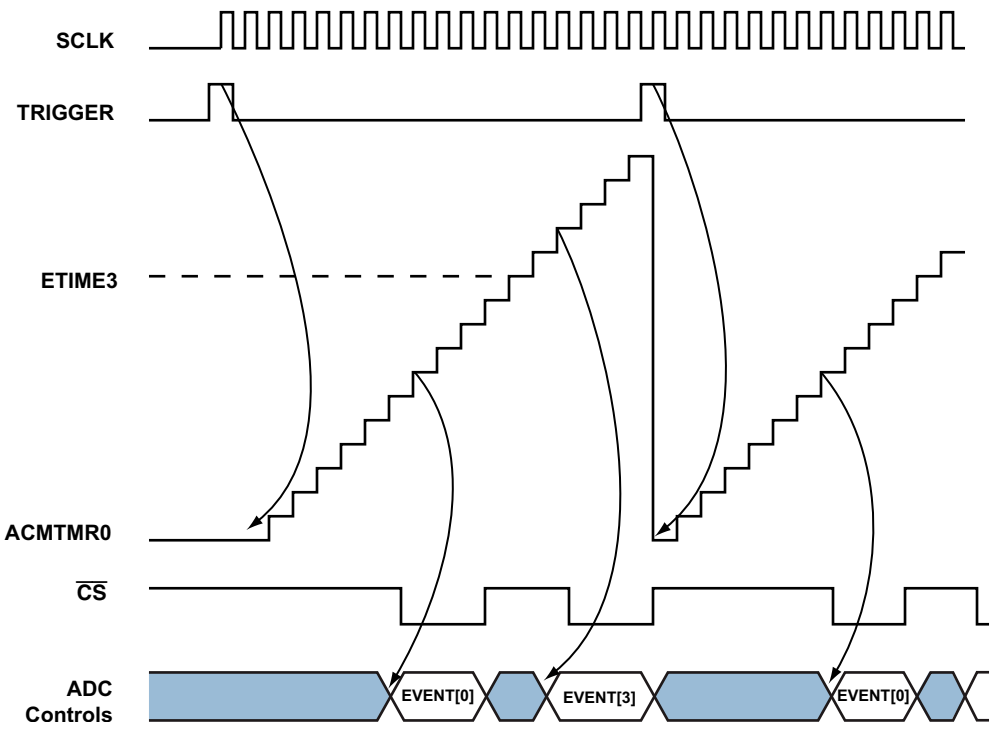


Figure 27-2: ACM Operation, Two Events

Note that, this figure depicts a usage case in which ACM Event3 Time register is programmed with a count value that is greater than that programmed in Event0. So, Event3 occurs after Event0. There are, however, no restrictions on the order of the different events. Event0 time can be greater than, less then or equal to Event3 time.

If value in Event0 Time register is less than the value in Event3 Time, Event3 occurs after Event0; while if the value in Event0 is greater than the value in Event3, then Event3 occurs before Event0. Whereas if Event0 time is equal to the Event3 time, the events are processed according to their priorities. Only the highest priority event is processed, and other lower priority event is missed (even if there is a space in pending event FIFO), as both events, are handled by same ACM Timer. Event0 has the highest priority. So in this case, the timing generation unit processes Event0, while Event3 is missed. In this case, the Event3 missed status bit (EM3) is set in the missed event status register (ACM_ MEVSTAT), and the Event Missed bit (EMISS) is set in the ACM Status (ACM_STAT) register, indicating that an event has been missed.

If event times are not sufficiently spaced apart, an event could occur while a previous event is underway (while the CS of the previous event is asserted). In this situation, the second event is queued in the pending event FIFO. If the pending event FIFO is full, the event will not get queued and will be missed. This can happen mostly when enabling both ACM Timers with different trigger inputs and the sources of these triggers are not synchronized together. In this case, it is possible that the events controlled by the two timers to overlap. It is therefore important to consider the possibility of events occurring either simultaneously or being missed when enabling events on two asynchronously-triggered timers. It is programmer's responsibility to ensure that the values in the event time registers do not lead to event misses. The appropriate event miss bit in the ACM_ MEVSTAT and ACM_STAT registers will be flagged, upon missing any event.

When both ACM timers are enabled, and if they triggered the events simultaneously, the event triggered by ACMTMR0 is given higher priority. For example, when an ACMTMR0 event (one of events 0 through 7) and ACMTMR1 event (one of events 8 through 15) occur simultaneously, the ACMTMR0 event is processed by the timing generation unit or is queued in the pending event FIFO before the processing or the queuing of the ACMTMR1 event.

When all the events enabled for a given ACM timer (ACMTMRx) are processed, the ACM timer stops incrementing. (Note that this timer action is not reflected in Figure). And corresponding Event Completion bit (ECOMPx) in the ACM Status register is flagged. The same bit also reflects in Event Compilation Status register (ACM_EVSTAT), which can optionally generate Event Completion interrupt if the corresponding bit in the Event Completion Interrupt mask register (ACM_EVMSK) is set. Two separate bits are available, one for each ACM Timer.

The ACM can be used to generate various sequences of ADC sampling events through appropriate programming of event time registers, event control registers, and triggers. For more information, see the usage cases described in *Emulation Mode Use Case*.

ADSP-BF60x ACM Register List

The ADC control module (ACM) provides an interface that synchronizes the controls between the processor and an analog-to-digital converter (ADC). The analog-to-digital conversions are initiated by the processor, based on external or internal events. A set of registers govern ACM operations. For more information on ACM functionality, see the ACM register descriptions.

Table 27-1: ADSP-BF60x ACM Register List

Name	Description
ACM_CTL	Control Register
ACM_TC0	Timing Configuration 0 Register
ACM_TC1	Timing Configuration 1 Register
ACM_STAT	Status Register
ACM_EVSTAT	Event Complete Status Register
ACM_EVMSK	Event Complete Interrupt Mask Register
ACM_MEVSTAT	Missed Event Status Register
ACM_MEVMSK	Missed Event Interrupt Mask Register
ACM_EVCTLn	Event N Control Register

Table 27-1: ADSP-BF60x ACM Register List (Continued)

Name	Description
ACM_EVTIMEn	Event N Time Register
ACM_EVORDn	Event N Order Register
ACM_TMR0	Timer 0 Register
ACM_TMR1	Timer 1 Register

ADSP-BF60x ACM Interrupt List

Table 27-2: ADSP-BF60x ACM Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
ACM0 Event Miss	38		LEVEL
ACM0 Event Complete	39		LEVEL

ADSP-BF60x ACM Trigger List

Table 27-3: ADSP-BF60x ACM Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
ACM0 Event Complete	19	LEVEL

Table 27-4: ADSP-BF60x ACM Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
ACM0 Trigger Input 2	33	
ACM0 Trigger Input 3	34	

ACM Event Handling Latency

The ACM ensures a predictable latency between the internal occurrence of an event (event time value matching the ACM timer count value) and the assertion of a sampling event by the timing generation unit (the assertion of CS and other ACM signals as appropriate).

Latency between occurrence of Event to CS assertion = (t_S + t_{ED}) SCLK cycles, where:

- t_S= ADC control setup cycles programmed in ACM_TC0 register
- t_{ED} = 1 SCLK cycle latency

This predictable latency is applicable only when events are generated when the Timing Generation Unit is idle. If this Timing Generation Unit was processing a prior sampling event, then the new event is held in the Pending Event FIFO, and the latency is increase by the duration that the new event is held in the pending event FIFO.

If an external trigger input is selected as a trigger input of the ACM, then synchronization to this external signal leads to a 3 SCLK cycle fixed delay and 1 SCLK cycle variability due to delays in latching asynchronous external triggers. When the external trigger is synchronous to SCLK, the 1 SCLK cycle variability is eliminated and the latency from the external trigger to the start of the count of an ACM timer becomes fixed at 3 SCLK cycles. This latency is denoted as t_{TRIG}.

As a result, the total latency between an external trigger and the assertion of an ADC sampling event, assuming that the sampling event does is not queued in the pending event FIFO, is:

Total Latency = t_{TRIG} + t_{ED} + t_{PD} + t_S

The following figure shows latency details from the occurrence of external triggers to the assertion of ADC sampling events.

In the following figure, observe the following timing definitions:

- t_{TRIG} = trigger to timer start delay (3 to 4 SCLK)
- t_{PD} = Event Time (programmed in ACM_ETx register of event).
- t_{ED} = internal event delay (1 SCLK)
- t_S = set up time (programmed in ACM_TC0 register)
- t_{CSW} = CS width (programmed in ACM_TC0 register)
- t_H = hold time (programmed in ACM_TC1 register)

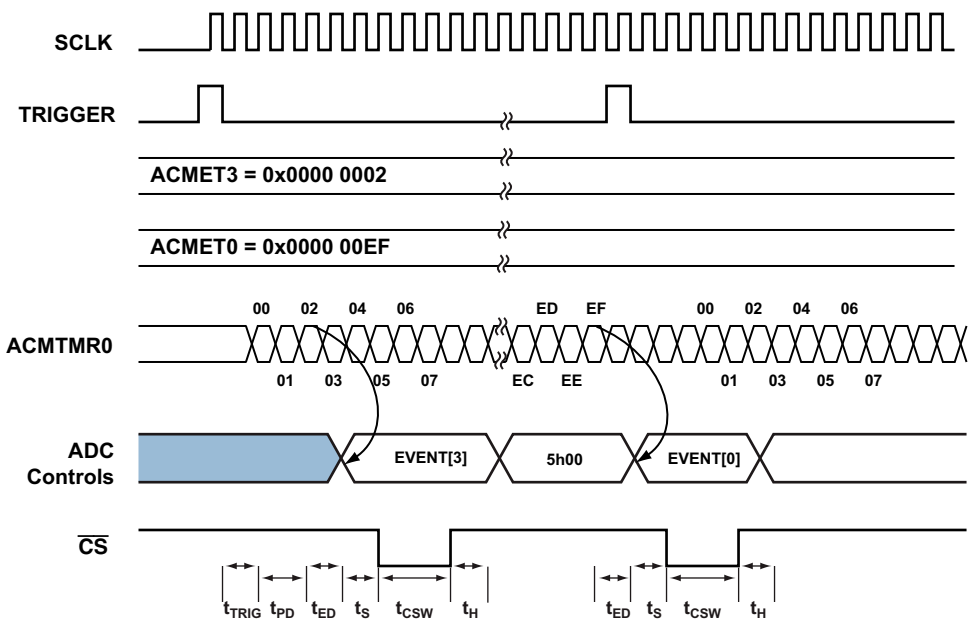


Figure 27-3: Latency of External Triggers to Assertion of ADC Sampling Events

ACM Timing Specifications

The AC timing of the ACM signals are specified in the ADSP-BF60x Embedded Processor Data Sheet. When trigger sources external to the processor are used for triggering the ACM (for example, external signals on the GPIO, timer or PWM sync pins), the minimum pulse-width for such trigger sources needs to be greater than one SCLK period in order to detect it as a valid trigger by the ACM trigger logic.

- When the ACM is used in conjunction with the SPORT, the setup and hold timing requirements for the SPORT data signals with respect to ACLK are different from those requirements with respect to internally-generated or externally-supplied SPORT clock. Consult the ADSP-BF60x Embedded Processor Data Sheet for information on these timing requirements.

When using Gated clock mode (ACM_CTL.CLKMOD=1), the interfaced serial mode should also be configured in Gated clock mode (SPORT_CTL.GCLKEN=1). In this case, it is required to satisfy some conditions in order to set-up the serial port in Gated clock mode. These conditions are:

- The serial port needs at least 7 serial clock cycles between enabling the SPORT and first frame sync. If this requirement is not met, the SPORT may drop the first data. (For subsequent data this requirement is not applicable).
- The frame sync should be in the inactive (de-asserted) state when the SPORT is enabled. Otherwise one extra cycle (in addition to the above mentioned) is needed before the frame sync can be applied. If this requirement is not met, the SPORT may drop the first data.

ACM External Pin Timing

The ACM clock (ACLK) is derived internally from SCLK clock using the CKDIV divider specified in the ACM_TCO register. The other output signals such as ADC control pins (ACM_A[4:0]) and CS are driven on the rising edge of SCLKclock. As a result, these signals may not synchronous to ACLK. The setup, hold, and other timing parameters of the ADC control signals, width of CS signal and the frequency of ACLK can be configured in the ACM timing configuration registers (ACM_TCx). The polarity of CS and ACLK can be configured in the ACM control register (ACM_CTL). The timing parameters of the ADC control pins (ACM_A[4:0]) cannot be individually specified.

The inactive period of CS (t_{CSIW} as shown in the following figure) is the sum of the three timing parameters – Setup Time (t_s), Zero Time (t_z) and the Hold Time (t_H):

$t_{CSIW} = t_S + t_Z + t_H$.

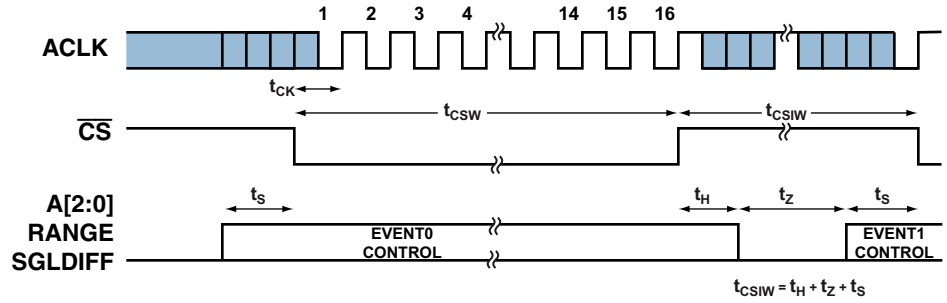


Figure 27-4: ACLK Timing Reference

Appropriate specification of values of these three parameters can yield the desired inactive period of CS.

In order to provide a predictable latency from the occurrence of an internal event to the assertion of an external ADC sampling event, the ADC controls and CS must be driven on the rising edge of SCLK. Therefore, the Setup Time (T_s) of these signals is specified in terms of SCLK. However, the hold-time and zero-time are specified in terms of ACLK cycles.

To achieve accurate timing relationship between CS and ACLK (which is a normally a free running clock), the ACLK signal is re-aligned with the active edge of CS. This realignment of ACLK ensures that the setup time of the first active edge of ACLK, with respect to the active edge of CS, is at least 1 ACLK cycle.

The figures in the following sections show various scenarios of ACLK re-alignment. All of these figures assume an ACLK:SCLK ratio of 1:4.

- **Case 1—Chip Select Asserted during the High Phase of ACLK**
- **Case 2—Chip Select Asserted During the Low Phase of ACLK**
- **Case 3—Chip Select Asserted Right Before the Falling Edge of ACLK**
- **Case 4—Chip Select Asserted Right Before the Rising Edge of ACLK**
- **Case 5—ACLK Polarity Set to 1 (CLKPOL=1)**

Notice that re-alignment of ACM clock causes suppression or extension of clock edges, leading to duty cycle variation. It is important to ensure that systems interfacing with the ACM can tolerate such duty cycle variation.

The figures show both the ACM-generated CS signal, which is output externally onto the appropriate ACMOFS pin, and the serial port receive frame sync (SPORT1_xFS) signal, which is an internal signal that is routed to the frame sync input of the appropriate SPORT.

Please note that ACM clock polarity can be configured using the CLKPOL bit of ACM control register. After assertion of CS signal, the first edge of ACLK can be configured to be either rising edge or falling edge. Also, by default the clock is free running; it is possible to gate the ACM clock during inactive CS period using the CLKMOD bit of the same register.

Case 1—Chip Select Asserted During the High Phase of ACLK (CLKPOL=0)

The following figure shows the realignment of ACLK when CS is asserted during the high phase of ACLK. The first edge of ACLK after the assertion of CS is the falling edge.

The two reference clock signals (RefACLK1 and RefACLK2) are shown to illustrate how the ACLK signal can be generated from a free running clock (RefACLK1) in order to meet the timing requirements between ACLK and CS. RefACLK2 is based on the free running clock RefACLK1, but is adjusted such that its period is immediately reset upon the assertion of CS. The resulting ACLK signal, shown in the figure, is such that the time from the active edge of CS to the falling edge of ACLK is constant at a period of 1ACLK cycle.

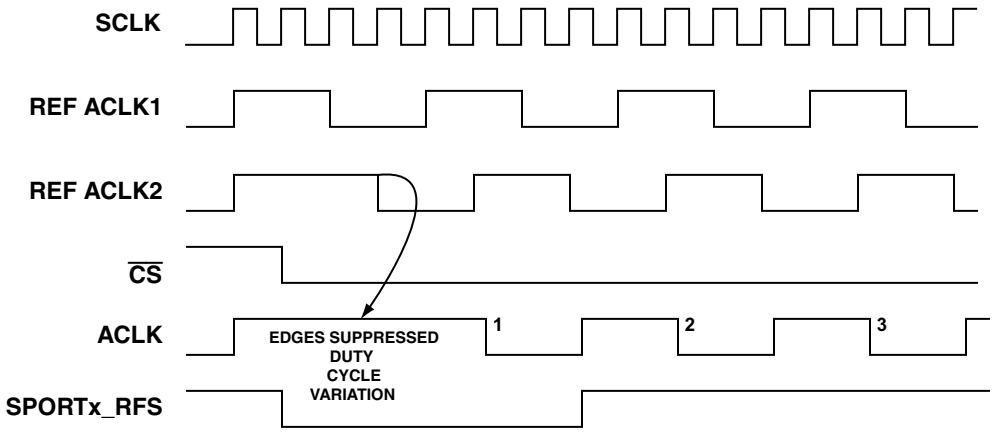


Figure 27-5: Chip Select Asserted During the High Phase of ACLK

Case 2—Chip Select Asserted During the Low Phase of ACLK (CLKPOL=0)

When CS is asserted during the low phase of ACLK, as shown in the following figure, ACLK is immediately pulled high causing a duty cycle variation. In this case, similar to Case 1, the time from the active edge of CS to the falling edge of ACLK is 1ACLK period.

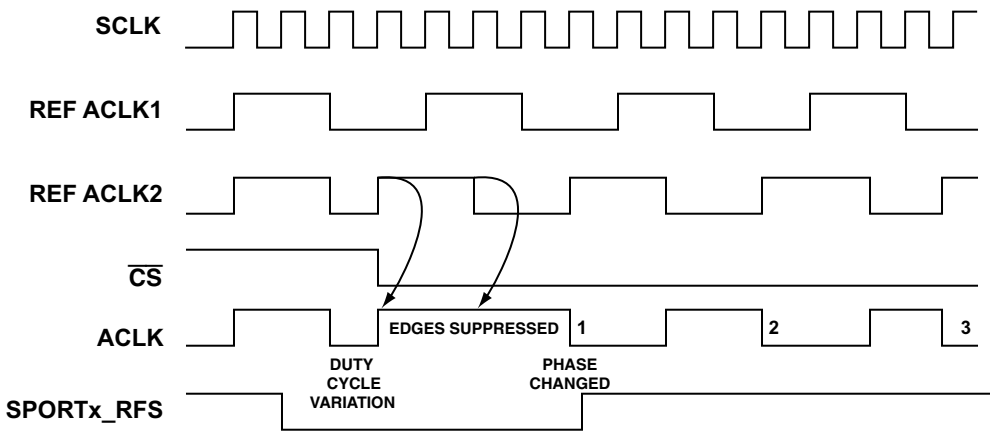


Figure 27-6: Chip Select Asserted During the Low Phase of ACLK

Case 3—Chip Select Asserted Right Before the Falling Edge of ACLK (CLKPOL=1)

When CS is asserted right before the falling edge of ACLK, the falling edge of ACLK is suppressed, as shown in the figure. This ensures that the time from the active edge of CS to the falling edge of ACLK is constant at a period of 1ACLK cycle.

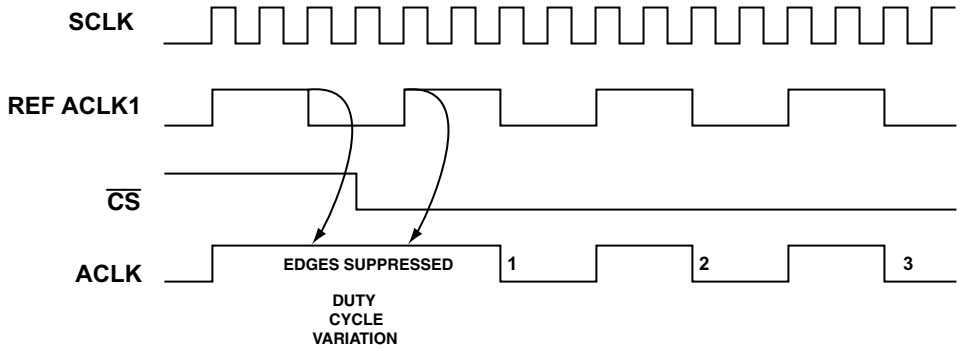


Figure 27-7: Chip Select Asserted Right Before the Falling Edge of ACLK

Case 4—Chip Select Asserted Right Before the Rising Edge of ACLK (CLKPOL=0)

When CS is asserted right before the rising edge of ACLK, the high phase of ACLK is extended, as shown in Figure. This extension ensures that the time from the active edge of CS to the falling edge of ACLK is constant at a period of 1ACLK cycle.

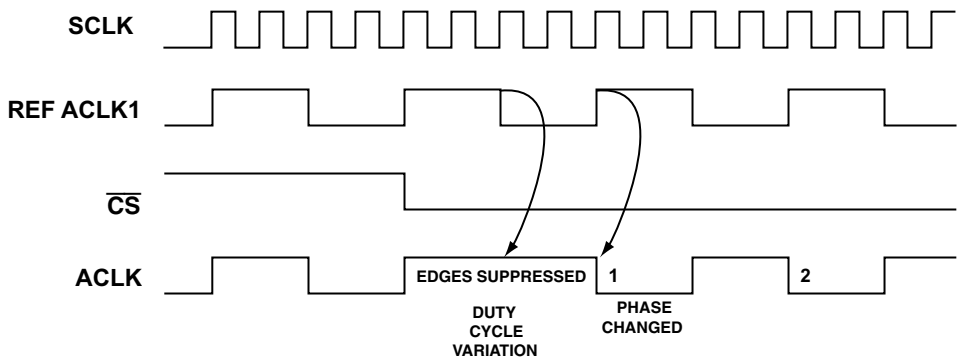


Figure 27-8: Chip Select Asserted Right Before the Rising Edge of ACLK

Case 5—ACLK Polarity Set to 1 (CLKPOL=1)

When the ACLK polarity is set to 1 (bitCLKPOL is set to 1 in the ACM Control register), the first ACLK edge after the assertion of CS is the rising edge. The ACM ensures that the time from the active edge of CS to the rising edge of ACLK has a constant duration of 1 ACLK cycle. The figure shows an example diagram of the case where CLKPOL=1.

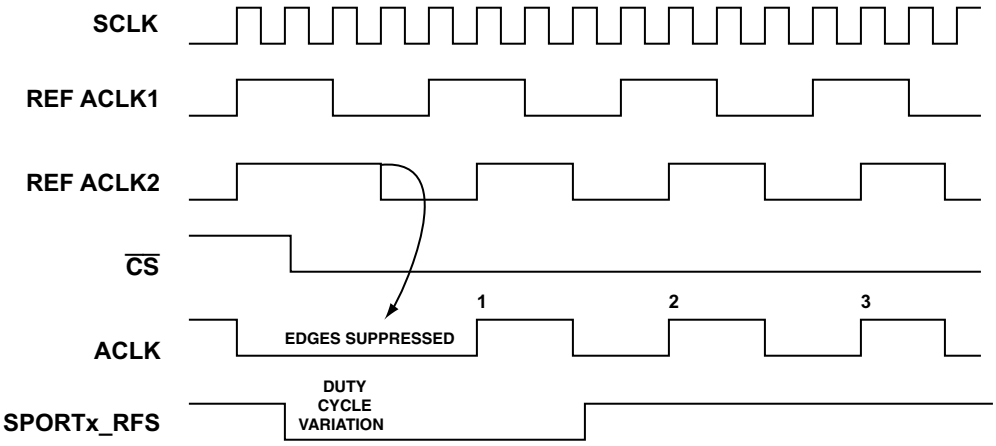


Figure 27-9: ACLK Polarity Set to 1

ACM Architectural Concepts

The following sections provide information on the architecture of the ACM module.

ACM Block Diagram

The ADC Control Module consists of two independent 32-bit ACM Timers, 16 Event Register pairs, 16 Event Comparators and a Timing Generation Unit.

ACM can accept four trigger inputs (internal as well as external signals); and on receiving a valid trigger on selected trigger input, the ACM timer/s start counting (based on the Mode of ACM). The trigger input can be independently selected for each timer.

Two sets of 8 event register pairs (total 16 event register pairs) determines the ADC controls for each ADC sampling, and also determines when the sampling happens. The Event register pair consists of: Event Control Register (ACM_ERx) and the Event Time Register (ACM_ETx). The Event Comaparators Unit, compares the ACM Timer's count with Event Time of associated enabled events and, upon matching the count, signals appropriately to Timing Generation Unit. The Timing Generation Unit starts handling the events by driving the CS and ACM[4:0] signals accordingly.

The **ACM Block Diagram** shows the structure of the ACM. The following sections discuss these blocks in detail.

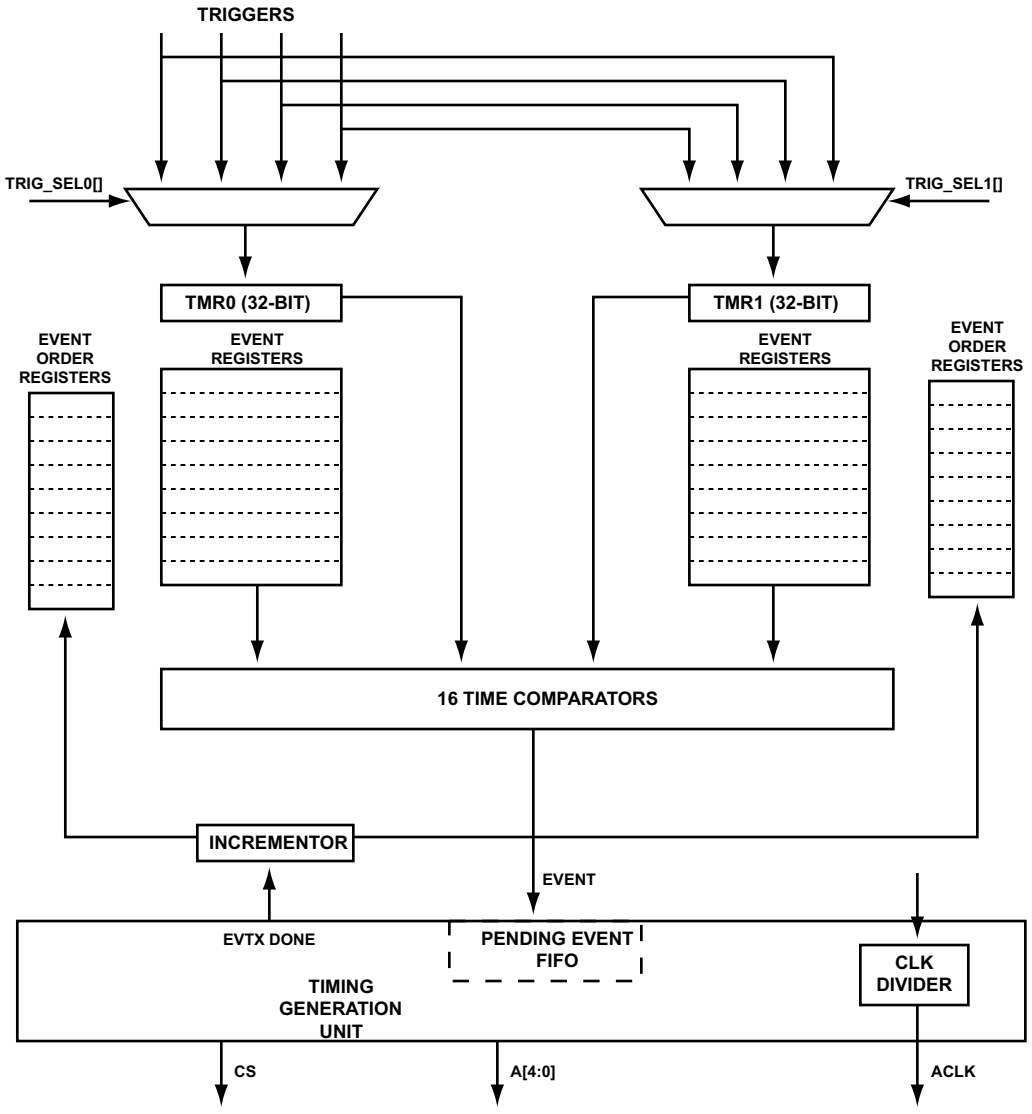


Figure 27-10: ACM Block Diagram

ACM Trigger Inputs

ACM can accept four trigger inputs, based on which ACM timers start running at SCLK rate. ACM contains two 32-bit internal timers; each can be independently configured to use one of these trigger input. The two separate fields, TRGSEL0 and TRGSEL1, of ACM Control register selects the trigger input for ACMTIMER0 and ACMTIMER1 respectively.

Therefore, ACM uses two trigger inputs when both ACM timers are enabled for different trigger inputs. However, it uses only one trigger if both ACM Timers are enabled for same trigger input or if single ACM Timer is enabled. The non-selected trigger inputs are 'don't care' for ACM. So, at most two and at least one selected trigger input should be active in the system for ACM to start it's operation.

The four possible trigger inputs are briefly described below:

- ACMTriggerinput0 (ACM_T0) – PE8:

  Trigger input- 0 is sourced from PE8 pin of the PORTE (sometimes also referred as GPIO[72]). When ACM is enabled, input tap on PE8 pin is enabled; so, ACM trigger input can be from any source (internal or external) depending on PE8 pin configuration.

  When PE8 pin is configured in GPIO mode (FER=0), the source of GPIO signal may be either internal or external depending on the GPIO direction configured in PORTE_DIR register. When PE8 pin is configured in function mode (FER=1), the trigger- 0 input is sourced in from peripheral signals based on PORTE_MUX register setting for PE8 pin. For example the ACM can source this trigger input from PWM0_SYNC signal (internally generated by PWM unit or externally provided), if MUX bits for PE8 pin are set to b#00. Whereas, if MUX bits for PE8 are configured to b#01, trigger input is sourced from PPI0_FS1 signal (internally generated by EPPI0 unit or externally-generated).

  So, FER and MUX bits for PE8 pin must be programmed appropriately considering the source of trigger input. Enabling input tap ensures that ACM operation would not interfere with module driving the PE8 pin.

- ACMTriggerinput1 (ACM_T1) – PG5:

  Trigger input- 1 is sourced from PG5 pin of the PORTG (sometimes also referred as GPIO[101]). When ACM is enabled, input tap on PG5 pin is enabled; so, ACM trigger input can be from any source (internal or external) depending on PG5 pin configuration.

  When PG5 pin is configured in GPIO mode (FER=0), the source of GPIO signal may be either internal or external depending on the GPIO direction configured in PORTG_DIR register. When PG5 pin is configured in function mode (FER=1), the trigger- 1 input can be sourced in from peripheral signals based on PORTG_MUX register setting for PG5 pin. E.g. ACM can source this trigger input from PWM1_SYNC signal (internally generated by PWM unit or externally generated), when MUX bits for PG5 pin are set to b#10.

  So, FER and MUX bits for PG5 pin must be programmed appropriately considering the source of trigger input. Enabling input tap ensures that ACM operation would not interfere with module driving the PG5 pin.

- ACMTriggerinput2 (ACM_T2) – ACMSlaveTriggerID33:

  The Trigger Routing Unit of processor provides system-level sequence control without core intervention. When this trigger input mode is selected, ACM acts as Trigger Slave and accepts trigger through Trigger Slave ID- 33. The 'Slave Select' (SSR) field of TRU_SSR33 register should be configured to receive triggers from a specific trigger Master. In this way, ACM_T2 trigger input can accept triggers asserted by that particular trigger Master or through software by writing ID of that trigger Master to one of the four fields in the TRU_MTR register. The trigger response from selected master is internally routed to ACM trigger input. This way, ACM slave trigger ID- 33 is capable of receiving any one of the 86 internal triggers available.

- ACMTriggerinput3 (ACM_T3) – A CMSlaveTriggerID34:

  Similar to ACM_T2 input, ACMTriggerInput3 (ACM_T3) is internally provided by Trigger Routing Unit of the processor. When this trigger input mode is selected, ACM acts as Trigger Slave and accepts trigger through Trigger Slave ID- 34. The 'Slave Select' (SSR) field of TRU_SSR34 register should be configured to receive triggers from a specific trigger Master. In this way, ACM slave trigger ID 34 can accept triggers asserted by that particular trigger Master or through software by writing ID of that trigger Master to one of the four fields in the TRU_MTR register. The trigger response from selected master is internally routed to ACM trigger input. This way, ACM slave trigger ID 34 is capable of receiving any one of the 86 internal triggers available.

  Refer Trigger Routing Unit chapter for more details about Trigger slaves and Trigger masters.

For all trigger input signals, the active edge of the trigger is programmable in the ACM Control register as either rising edge or falling edge trigger.

The following figure shows the detailed ACM trigger generation logic.

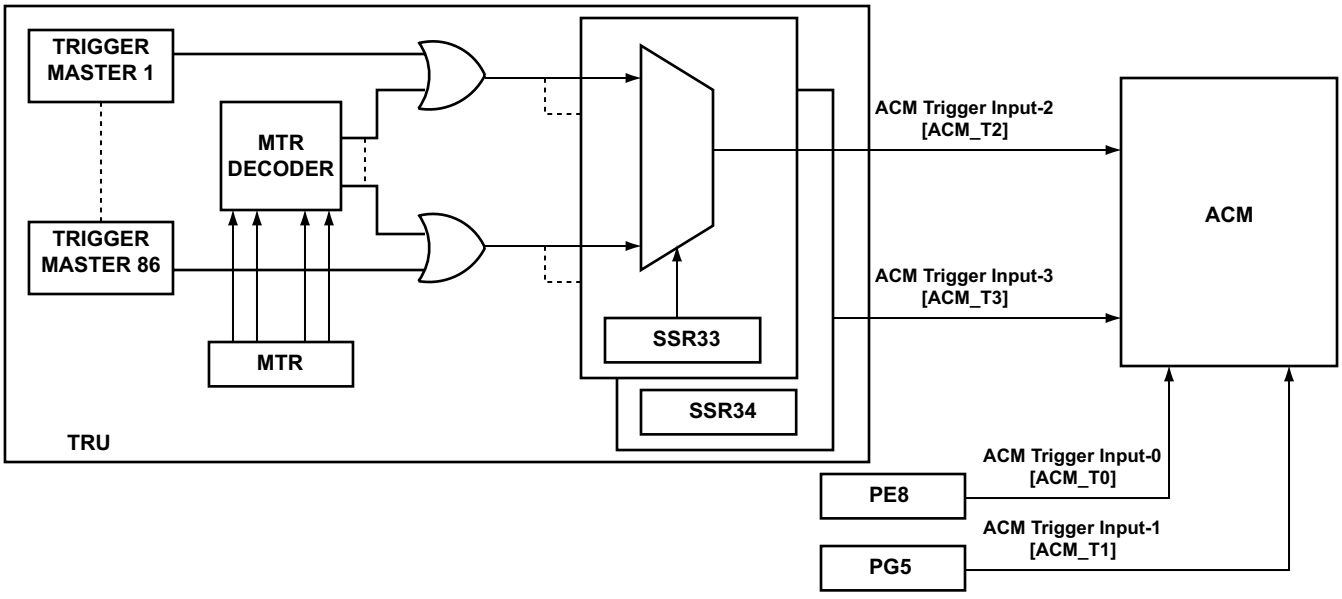


Figure 27-11: Detailed ACM Trigger Generation Logic

When trigger sources external to the processor (for example ACM_T[1:0]) are used for triggering the ACM Timers, the minimum pulse width for such trigger sources needs to be greater than one SCLK period.

NOTE: A latency of no more than four SCLK cycles exists between external trigger and ACM Timer starts counting. Please refer to *ACM Event Handling Latency* section for further details.

ACM Timers

The ACM module has two independent 32-bit timers (ACMTMR0 and ACMTMR1) which start counting at system clock (SCLK) rate upon detecting valid trigger on selected trigger input. The timers can be independently enabled using the timer enable bits (ACMTMR0EN, ACMTMR1EN) in the ACM Control register; and can be independently configured for one of the four trigger inputs with configurable polarity of the signal. At least one ACM timer should be enabled for proper ACM operation.

By default, each ACM timer has 8 event associated with it. If both timers are enabled, the Event[7:0] are associated with ACMTMR0; while the Event[15:8] are associated with ACMTMR1. However, if only one timer is enabled, all of the event registers are associated with that particular timer. For example, if only ACMTMR1 is enabled, (if ACMTMR0EN=0 and ACMTMR1EN=1), all 16 events, Event[15:0], will be handled by ACMTMR1.

These timers start counting when a trigger input occurs that is selected for that particular timer. The timer only stops counting under one of the following conditions:

1. A timer rollover occurs.
2. All the events associated with the trigger have completed.

Note that a timer rollover can never happen, unless the Event Time Register of an event is programmed at some point after the trigger occurs; this is a practice that is contrary to ACM programming guidelines.

In the second case, the exact time at which the Timer stops counting depends on the FIFO state when the last event occurred internally.

If a trigger occurs while the timer is counting, the time resets and starts counting again. In this case, some of the events may miss resulting in flagging appropriate status bit and optionally an Event Missed interrupt.

When an ACM timer is disabled or the ACM itself is disabled, the timer resets to zero.

Event Register Pairs

An Event, for the ACM, is a point in time where ADC sampling has to happen on a particular channel of the ADC with the specified control settings of the ADC.

ACM can handle total 16 events which are grouped into two sets of 8 events. As explained in ACM Timer sections, either 8 events can be assigned to each of the timers (if both timers are enabled) or all 16 events can be assigned to one particular timer (if only one timer is enabled).

All events can be independently configured and enabled. The enabled events determine the ADC controls and timing for each ADC sampling interval. Each event consist a register pair comprising an event control

register (ACM_ERx) and an event time register (ACM_ETx). The ACM_ERx register enables a particular event and determines settings for the ADC control lines (ACM_A[4:0]) for that particular ADC conversion. The ACM_ETx register determines the time offset from the corresponding ACM timer trigger input to the start of that particular event (i.e. when the event has to occur w.r.t. trigger input). This time offset should be specified in terms of system clock 1 of the processor.

At least one event, associated with enabled ACM Timer, should be enabled for ACM to execute ADC sampling.

Event Comparators Unit

The Event Comparator block consists of 16 event time comparators which determine when an enabled event should happen. After detecting valid trigger on selected trigger input, ACM timer starts running at system clock 1 rate. The comparators compare the ACM timer count with the event time specified in the ACM_ETx register of the enabled event. If the time value matches, the comparators indicate an active event signal to the timing generation unit.

If an event happens when another event is ongoing, the occurred event is stored in the pending event FIFO of Timing Generation Unit. If more than one event associated with a ACM Timer are active during the same SCLK cycle, only the highest priority event is processed, and all other events are missed (even if there was space in the pending event FIFO). However, if both Timers are enabled and if multiple events associated with both ACM Timers are active at the same SCLK cycle, then two events, one highest priority active event per each timer, are signaled.

The priority of events is fixed; event with lowest event ID has higher priority compared other events.

When both ACM Timers are enabled, Event0 has the highest priority and Event7 has the lowest priority in the Event[0:7] group associated with ACMTIMER0. Similarly Event8 has the highest priority and Event15 has the lowest priority in the Event[8:15] group associated with ACMTIMER1. So if Event1 and Event5 occur simultaneously, then Event5 is missed, even if there is space in the pending FIFO. But between the Event[0:7] and Event[8:15] group, simultaneous events can be written into the FIFO. For example, if Event0 and Event9 occur together, then both are written into the FIFO, Event[0:7] group has higher priority, so the order of Events in the FIFO is Event0 first and then Event9. If Event1, Event5, Event9, Event15 happened together, then Event5, and Event15 are missed and Event1 and Event9 are put into the FIFO (Event1 first followed by Event9). When events that are triggered by both timers occur simultaneously, the event triggered by ACMTMR0 is given higher priority.

When only single ACM Timer is enabled, then all 16 events, Event[0:15], are assigned to that timer. So, Event0 has highest priority; while Event15 has lowest priority. So, in this case, if Event1, Event5, Event9, Event15 happened together, then only Event1 will be placed in forwarded, while Event5, Event9 and Event15 will be missed, even if there is space in the pending FIFO.

If an event is missed, the EMISS bit in the ACM Status register (ACM_STAT) and the corresponding bit in the ACM Event Missed Status register (ACM_MEVSTAT) are set.

Timing Generation Unit

After event signaling from event comparators, the timing generation unit initiates an ADC sampling interval as per settings of that particular event. It generates ADC control signals based on the Event parameter field of the ACM_ERx register setting. The timings of output signals (ACLK, CS, A[4:0]) are determined by the ACM Timing registers (ACM_TCx) which contains the fields like Set-up time, Hold time and Zero time for these signals in addition to ACM clock divider.

The Pending FIFO is part of Timing Generation Unit. If an event happens when ACM is busy with another event, the occurred event is stored in the pending event FIFO. This pending event is serviced (for example, the ACM starts an ADC conversion for the event that occurred), after completion of ongoing event.

The pending event FIFO has a depth of 4, so it can hold up to four pending events. If an event occurs when the pending event FIFO is full, that event is missed. If an event is missed, the EMISS bit is set in the ACM_STAT register and the corresponding bit in the ACM_MEVSTAT register also is set.

On disabling the ACM, all the pending entries in the pending FIFO are flushed.

Status Flags and Interrupts

The ACM provides a read-only Status register (ACM_STAT) to check the module activities such as which event is currently being serviced, whether any event has been missed events or all the events has been serviced for the current trigger.

In addition to this Status register, ACM also provides two general-purpose status registers, ACM Event Completion Status register (ACM_EVSTAT) and ACM Missed Event Status register (ACM_MEVSTAT). The ACM_EVSTAT register specifies servicing of which enabled events has completed for a particular trigger cycle; while ACM_MEVSTAT register specifies which enabled event has been missed for that particular trigger cycle. This information is provided for all the 16 events via individual bits.

Based on these status bits, ACM can generate two interrupts, event completed or event missed, for each event. These interrupts can be selectively enabled for particular ACM events via ACM Completed Event Interrupt Mask Register (ACM_EVMSK) and ACM Missed Event Interrupt Mask Register (ACM_MEVMSK) registers.

The Event Completion interrupt is generated only after the entire event completes externally (for example, when CS signal goes inactive, and Hold Time (TH), Zero Time (TZ) periods are completed for that particular event). The ACM_EVSTAT register provides the status of each event indicating which event has caused the interrupt. It is also possible to generate this interrupt when all the events associated with an ACM Timer are completed for an ACM trigger cycle. This interrupt can be cleared by writing to the relevant W1C (write 1 to clear) bit in ACM_EVSTAT register.

The Event Missed Interrupt is generated when an enabled event is missed for a trigger cycle and the corresponding mask bit of ACM_MEVMSK register is set. The event might be missed in Event comparator unit if more than one event, related to same Timer, are active during the same SCLK cycle; or it can be missed in Timing Generation Unit if an event occur when event pending FIFO is full. The ACM_MEVSTAT register provides the status of each missed event indicating which event miss caused the interrupt. This interrupt can be cleared by writing the relevant W1C bit in the ACM_MEVSTAT register.

NOTE: A Status bit set either in ACM_EVSTAT or ACM_MEVSTAT registers triggers an interrupt only if the corresponding bit in the ACM_EVMSK or ACM_MEVMSK registers is enabled.

In addition to Event Completion interrupt (upon completion of particular event or all events), ACM can also provide an trigger output to Trigger Routing Unit of the processor. This trigger output can be used by Trigger slaves for their operations without requiring core intervention.

Event Order Registers

For debugging purpose, BF60x ACM hardware includes 16 Event Order registers-one per each event-which indicates the order in which the events were completed externally.

These registers are denoted as ACM_EVORDx, where x stands for event ID, 0 to 15. The 8-bit EVT_ORDER field of this register indicates in which order the ADC data has been captured corresponding to the event. This field accumulates the order count every trigger cycle, unless it is cleared in the software. At each trigger cycle, the values of the register will be updated, so it must be read at the end of the each trigger cycle (as it will write the new order value of the event in the next trigger cycle). Thus, the 8-bit field can store the order of 256 data captures at a stretch, after which it will start the order count from zero again.

All the event registers can be reset in software by setting ORDR_RST bit of the ACM control register. When set, it clears all the Event Order Register value to zero. This bit auto-clears to 0 after all ACM_EVORDx registers are cleared. These registers can also be cleared automatically by selected trigger input of ACM Timers, if AOREN bit is set. The OTSEL bit of the ACM Control register determines which trigger input to select for this auto-clearing.

The Event Order functionality is explained below with an example where ACM has only three events enabled (let it be Event1, Event7 and Event13). Following figure shows how the Order register value of these events will be at different stages.

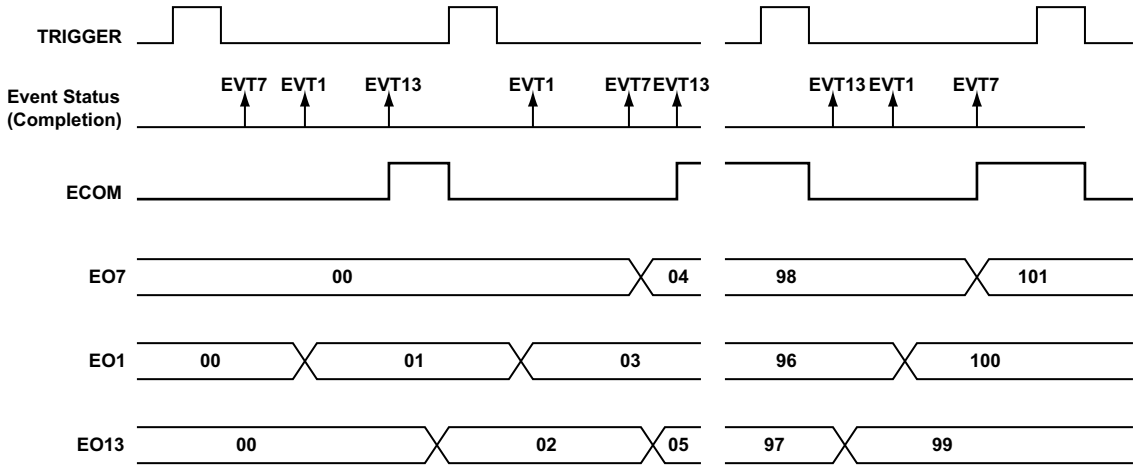


Figure 27-12:

ACM Programming Concepts

Since ACM module is being used with the SPORT, PWM, GPTIMER and GPIO; the programming must comply with the following guidelines for reliable operation of the ACM.

- ACM is control module and provides clock and chip select and control signals with required timing. But for capturing the data from ADC, one of the halve of SPORT1 is used.
- The ACM should be enabled before enabling the SPORT. But SPORT can be configured before enabling ACM. SPORT should be configured in slave mode [external clock (ICLK=0), external frame sync(IFS=0)] as receiver.

  The timings of external ADC decide the settings of LFS, LAFS, CKRE bits of SPORT Control register. Generally SPORT is configured in DSP serial mode to receive the ADC samples, but other operating mode (such as multichannel) may be possible.

  If ACM is supposed to be programmed to gated clock mode (CLKMOD=1), the serial port should also be set in gated clock mode(GCLKEN=1).
- DMA mode of SPORT operation is preferred, as it saves the processor MIPS when receiving chunk of data. However, receiving ADC sample in core mode is also possible. So, when using DMA mode, DMA registers of selected SPORT should be configured appropriately; and DMA must be enabled before enabling SPORT. When using both primary and secondary channels of SPORT to receive data from two ADC channels, the 2D feature of DMA can be effectively used to de-interleave the data from two channels. When using Core mode of SPORT operation, core handler should be registered to handle the data read requests from SPORT receiver.
- In addition to SPORT register settings, the PORT registers should also be set properly to enable SPORT Data pins, ACM clock, CS and Control pins. When using either of the ACM_T[1:0] trigger inputs for ACM Timers, FER and MUX bits of corresponding pins (PE8 or PG5) must be configured properly according to source of trigger input.
- When using ACM_T[2:3] trigger inputs for ACM timers, the Trigger Routing Unit can be configured and enabled at this step. The corresponding Slave Trigger ID should be programmed properly using Slave Select Register to select the required master trigger. When using these trigger inputs, the Slave select register must not be configured when ACM is enabled, as default value of this register zero and Master Trigger ID-0 is system reserved.
- Before enabling the ACM (by setting the ACMEN bit), all the control bits of ACM Control register should be programmed properly. These control bits includes ACM trigger selects (TRGSELx), trigger input polarities (TRGPOLx), CS signal polarity (CSPOL), ACM clock polarity (CLKPOL), ACM clock mode (CLKMOD) and Serial port unit selection, Event order register settings.
- Configure the ACM Timing Control registers to define the ACM clock frequency and setup, hold & zero time of ACM control signals.
- The Timer Enabled (TMRENx) bits, however, should be programmed together only after ACM is enabled but once the bits are programmed it should not be changed later. Modifying these enable bits in the ACM Control register is not recommended while the ACM is in operation. Doing so can cause events

to change dependency from one timer to the other and can cause the values in the ACM status registers (ACM_EVSTAT and ACM_STAT) to be inaccurate. This means that if both timers are required for use, then enable them together after ACM is being enabled. If one timer is already enabled, then disable ACM, re-enable ACM and then program both timer enable bits together. Similarly when both timers are running, they should disabled together.

- Once configuration peripherals are done, ACM should be enabled first and then SPORT module (SPORT DMA should be enabled before enabling SPORT, can be in previous steps also). Ideally trigger should not active when enabling the ACM.
- After enabling ACM, the event register pairs (Event Control and Event Time registers) should be configured and enabled to create required events.
- It should be noted that ACLK is an external clock relative to the SPORT peripheral. Therefore, any SPORT requirements around a minimum number of stable external clock cycles before assertion of the first SPORT frame sync need to observed. The SPORT requires a minimum of 3 clock cycles before it is able to recognize a frame sync. When SPORT is configured in gated clock mode, this requirement becomes a minimum of 7 SPORT clock cycles. Therefore the required number of ACLK cycles should elapse before first assertion of CS. This can be guaranteed by any of the following methods:
  - Ensuring that ACM triggers are generated at least 3ACLK cycles after the ACM is enabled.
  - Ensuring that the event time value (ACM_ETx) of the first active event is such that 3ACLK cycles would elapse before the event is processed.
- When the minimum number of ACLK cycles before the assertion of CS is not observed, the SPORT may miss the data of the first ADC sampling event. There can be a software workaround for fulfilling this requirement. Program the ACM_TCO register (CKDIV value) after enabling the SPORT (subsequently after enabling ACM) but before the trigger is applied. Since the default value of CLKDIV is 1, the ACLK frequency is higher.Therefore the SPORT may receive the required clock cycles within short period of time (before the Frame Sync arrives).
- When using ACM_T[2:3] trigger inputs, the master can enabled at last (if it configured only to provide triggers to ACM) to generate the triggers.
- While disabling the ACM system, the SPORT should be disabled first, then DMA and finally the ACM should be disabled.

Emulation Mode Use Case

This section describes the usage modes of the ACM by illustrating how to implement various sequencing ADC sampling modes.

Single-Shot Sequencing Mode Emulation

In single-shot sequencing mode, all enabled events are sequentially issued one after the other on the occurrence of an ACM trigger. The sequence of events is fixed, starting with Event0 and ending with Event15.

The following figure shows an example of single-shot sequencing mode where only Event0 and Event1 are enabled. ETIME0 is the value written into the ACM_ET0 register, and ACMTMR0 is enabled in this mode. To emulate this mode of operation using the ACM.

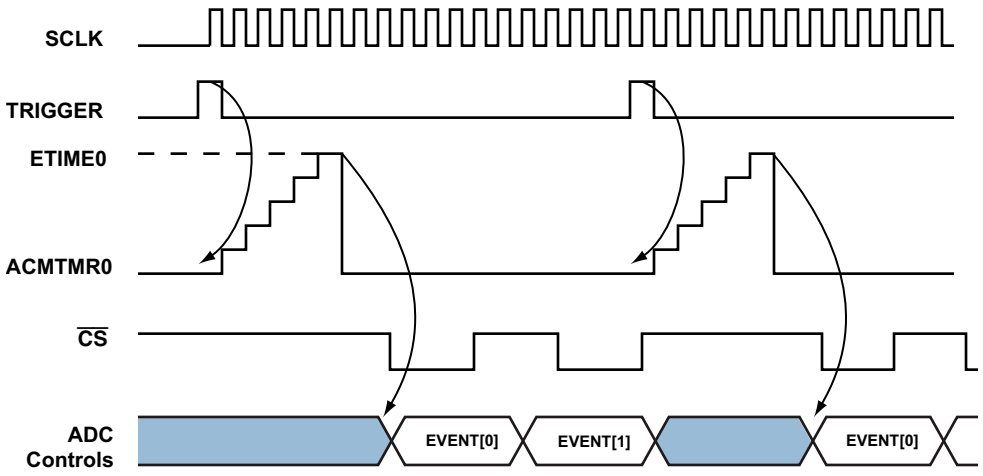


Figure 27-13: Single Shot Sequencing

- Configure the appropriate trigger source for initiating ACM activity. Please refer to "Interface Overview" for information on signals that can trigger the ACM counters.
- Enable only one ACM timer (ACMTMR0)
- Enable events and program the event time values as: Event0 time = X, Event1 time = X + Y, Event2 time = X + 2Y where:

  X = ETIME0, the initial time offset from trigger (if needed)

  Y = $t_H + t_{CSW} + t_S + t_Z$, where t_H is the hold time, t_Z is the zero t time, and t_S is the setup time for ACM Control lines as specified in ACM Timing Registers. For more information, refer "ACM External Pin Timing" section.

NOTE: Y has to be slightly less than the above value to ensure that the next event occurs before the first event completes, so that the next event is in the pending FIFO and enables the transitions between events without a break.

Continuous Sequencing Mode Emulation

Continuous sequencing mode is similar to single-shot sequencing mode, except in continuous sequencing the event sequencing is continuously repeated. As in single-shot mode, the time offset is programmable in continuous mode. The trigger in continuous mode is relevant only for the first time. Therefore, any subsequent triggers after the first active edge of the trigger are neglected.

The following figure shows an example of continuous sequencing mode with only two events – Event0, Event1 enabled. To emulate continuous sequencing mode using ACM:

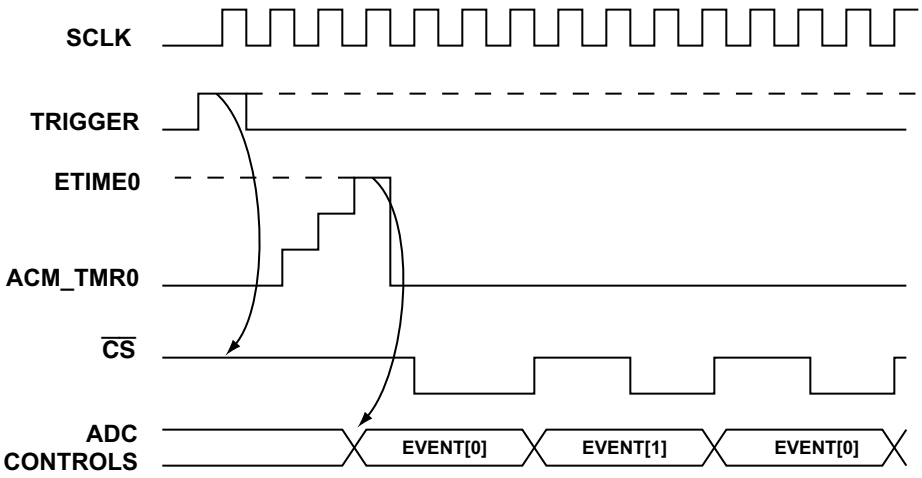


Figure 27-14: Continuous Sequencing

- Enable only one ACM Timer (say ACMTIMER0). Select Timer trigger input as from TRU of processor i. e. either of the ACM_T[3:2].

  If ACM_T2 is selected, program ACM Slave Trigger ID-33 to select ACM Event Completion (whose master Trigger ID is 19) as trigger input. If ACM_T3 is selected as trigger input, configure ACM Slave Trigger ID-34.

- Configure rest of the settings required by programming ACM Control and Timing Registers.

- Enable ACM Events with required ACM Control lines settings.

  Program the Event time registers as: Event0 time = X, Event1 time = X + Y, Event 2 time = X + 2Y where:

  X and Y values, in terms of SCLK, are as described in the single-shot case (Y can be slightly less than $t_H + t_{CSW} + t_S + t_Z$ to avoid any break between Events)

- Configure and enable System event controller.

  Also configure, map and enable the ACM Event Completion interrupt which will interrupt upon completion of all enabled ACM Events for the current trigger. That means, set only ECOMP0 (or ECOMP1, if using ACMTIMER1) bit of ACM Event Interrupt Mask (ACM_EVMSK) register.

- Enable ACM, SPORT and SPORT DMA as per the guidelines given in "Programming Concepts" section.

- Since we have configured ACM Trigger input as ACM Event Completion trigger output, the first trigger is necessary to start the ACM operation. We can provide this dummy trigger by writing Master Trigger ID into Master Trigger register (TRU0_MTR).

  So, write ACM Event Completion Trigger ID (19) into TRU0_MTR register. This will trigger the ACM Timers and ACM will start handling the events.

- After completing all events, ACM will provide Event Completion Trigger and corresponding interrupt will be generated. The ACM Trigger output will be provided to ACM Timers which will reset it's counter and start running from zero, which will cause ACM to re-handle all the enabled ACM events. And this sequence continues.

  However, in order to provide the Trigger outputs properly, the interrupt latch must be cleared in the ISR by clearing ECOMP0S (or ECOMP1S, if using ACMTIMER1) bit of the ACM Event Completion Status register (ACM_EVSTAT).

ADSP-BF60x ACM Register Descriptions

ADC Control Module (ACM) contains the following registers.

Table 27-5: ADSP-BF60x ACM Register List

Name	Description
ACM_CTL	Control Register
ACM_TC0	Timing Configuration 0 Register
ACM_TC1	Timing Configuration 1 Register
ACM_STAT	Status Register
ACM_EVSTAT	Event Complete Status Register
ACM_EVMSK	Event Complete Interrupt Mask Register
ACM_MEVSTAT	Missed Event Status Register
ACM_MEVMSK	Missed Event Interrupt Mask Register
ACM_EVCTLn	Event N Control Register
ACM_EVTIMEn	Event N Time Register
ACM_EVORDn	Event N Order Register
ACM_TMR0	Timer 0 Register
ACM_TMR1	Timer 1 Register

Control Register

The ACM_CTL register enables and selects the various modes of operation of the ACM.

ACM_CTL: Control Register - R/W

Reset = 0x0000 0000

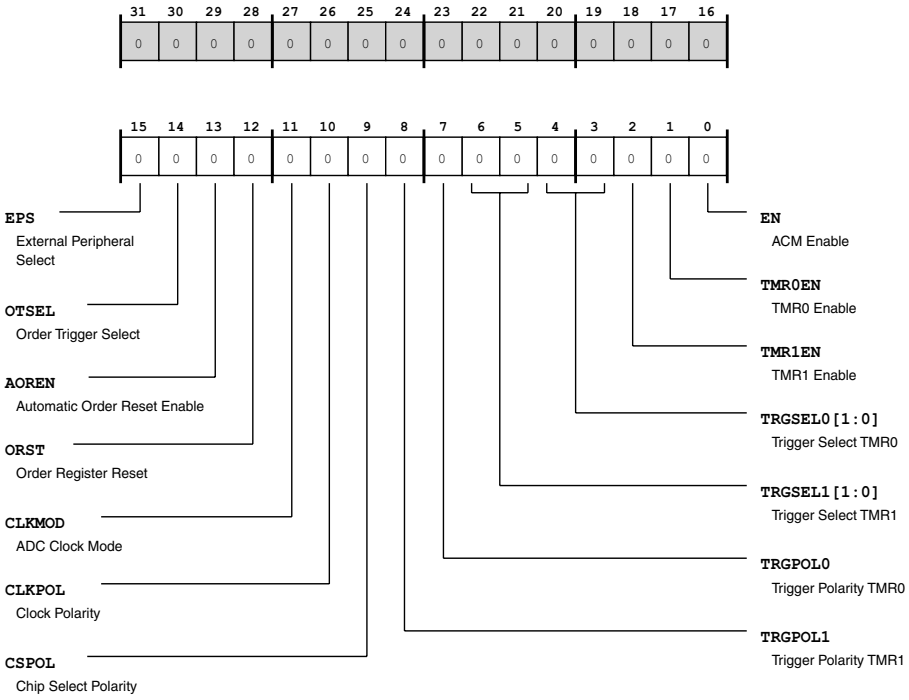


Figure 27-15: ACM_CTL Register Diagram

Table 27-6: ACM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	EPS	External Peripheral Select. The ACM_CTL.EPS bit selects whether the ACM interfaces to half SPORT1 A or half SPORT1 B.	
		0	Half SPORT1 A Interfaces to ACM
		1	Half SPORT1 B Interfaces to ACM

Table 27-6: ACM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	OTSEL	Order Trigger Select. The ACM_CTL.OTSEL bit selects whether TMR0 or TMR1 triggers a reset of the event order (ACM_EVORDn) registers. This bit is applicable only if the ACM_CTL.AOREN bit is set.	
		0	ACM TMR0 Triggers Reset of Order Registers
		1	ACM TMR1 Triggers Reset of Order Registers
13 (R/W)	AOREN	Automatic Order Reset Enable. The ACM_CTL.AOREN bit enables automatic reset of the event order (ACM_EVORDn) registers, based on the selected timer trigger. The ACM_CTL.OTSEL bit selects the trigger.	
		0	Disable Automatic Order Reset
		1	Enable Automatic Order Reset
12 (R/W)	ORST	Order Register Reset. The ACM_CTL.ORST bit resets the event order (ACM_EVORDn) registers' value to 0. This bit auto-clears to 0 after the ACM_EVORDn registers are cleared.	
11 (R/W)	CLKMOD	ADC Clock Mode. The ACM_CTL.CLKMOD bit selects whether the ADC clock mode is gated (ACM_CLK is gated when the ADC CS is inactive) or continuous (ACM generates continuous ACM_CLK).	
		0	Continuous Clock Mode
		1	Gated Clock Mode
10 (R/W)	CLKPOL	Clock Polarity. The ACM_CTL.CLKPOL bit selects whether the rising or falling edge of ACM_CLK comes after ADC CS becomes active.	
		0	Falling Edge of Clock After CS
		1	Rising Edge of Clock After CS
9 (R/W)	CSPOL	Chip Select Polarity. The ACM_CTL.CSPOL bit selects whether ADC CS is active high or low.	
		0	Active Low CS
		1	Active High CS

Table 27-6: ACM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	TRGPOL1	Trigger Polarity TMR1. The ACM_CTL.TRGPOL1 bit selects whether the trigger polarity for ACM TMR1 is falling or rising edge.	
		0	Rising Edge Trigger
		1	Falling Edge Trigger
7 (R/W)	TRGPOL0	Trigger Polarity TMR0. The ACM_CTL.TRGPOL0 bit selects whether the trigger polarity for ACM TMR0 is falling or rising edge.	
		0	Rising Edge Trigger
		1	Falling Edge Trigger
6:5 (R/W)	TRGSEL1	Trigger Select TMR1. The ACM_CTL.TRGSEL1 bits selects the external trigger for ACM TMR1.	
		0	Trigger 0 (ACM_T0 Pin)
		1	Trigger 1 (ACM_T1 Pin)
		2	Trigger 2 (Trigger Input 2 - TRU Slave)
		3	Trigger 3 (Trigger Input 3 - TRU Slave)
4:3 (R/W)	TRGSEL0	Trigger Select TMR0. The ACM_CTL.TRGSEL0 bits selects the external trigger for ACM TMR0.	
		0	Trigger 0 (ACM_T0 Pin)
		1	Trigger 1 (ACM_T1 Pin)
		2	Trigger 2 (Trigger Input 2 - TRU Slave)
		3	Trigger 3 (Trigger Input 3 - TRU Slave)
2 (R/W)	TMR1EN	TMR1 Enable. The ACM_CTL.TMR1EN bit enables ACM TMR1.	
		0	Disable ACM TMR1
		1	Enable ACM TMR1
1 (R/W)	TMR0EN	TMR0 Enable. The ACM_CTL.TMR0EN bit enables ACM TMR0.	
		0	Disable ACM TMR0
		1	Enable ACM TMR0

Table 27-6: ACM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	ACM Enable. The ACM_CTL.EN bit enables ACM operation.	
		0	Disable ACM
		1	Enable ACM

Timing Configuration 0 Register

The ACM_TC0 register determines the frequency of ACM_CLK (using the ACM_TC0.CKDIV field) and the setup cycles (using the ACM_TC0.SC field) for the ADC controls. Note that the setup cycles are specified in terms of SCLK.

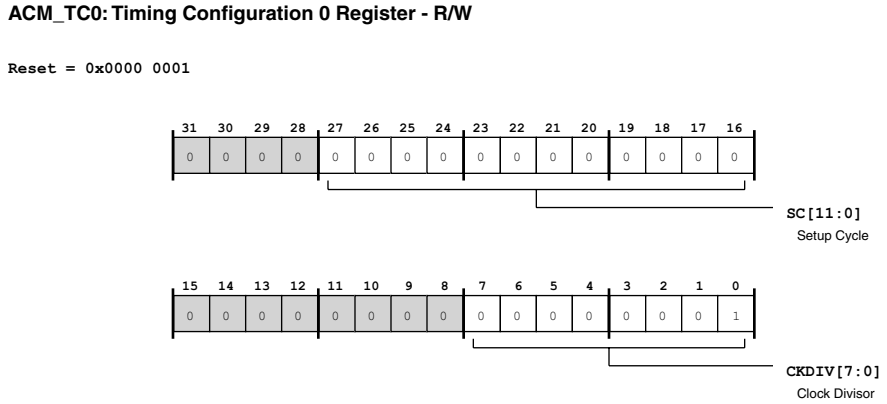


Figure 27-16: ACM_TC0 Register Diagram

Table 27-7: ACM_TC0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
27:16 (R/W)	SC	Setup Cycle. The ACM_TC0.SC bits select the ADC control (ACM_A0, ACM_A1, ACM_A2, ACM_A3, and others) setup time in SCLK cycles with respect to ADC CS active edge. The setup time may be calculated from: Setup Time = ACM_TC0.SC + 1 The maximum setup cycle time is 4096*SCLK, and the minimum setup cycle time is 1 SCLK.	
		0	1 SCLK Cycle Setup Time
		1	2 SCLK Cycles Setup Time
		4095	4096 SCLK Cycles Setup Time

Table 27-7: ACM_TC0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	CKDIV	Clock Divisor. The ACM_TC0.CKDIV bits select the frequency of ACM_CLK as a function of the system clock frequency (SCLK) and the value of the CKDIV field according to the formula: ACM_CLK frequency = (SCLK frequency)/(ACM_TC0.CKDIV + 1) The maximum ACM_CLK frequency is SCLK/2, and the minimum ACM_CLK frequency is SCLK/256. For example, for a 100 MHz SCLK, the ACM_CLK frequency range is from 390 KHz to 50 MHz. Note that the value ACM_TC0.CKDIV =0 is reserved.

Timing Configuration 1 Register

The ACM_TC1 register provides programmability for the active duration of chip select (T_{CSW}), Hold Cycles (T_H), and Zero Cycles (T_Z) for ADC controls.

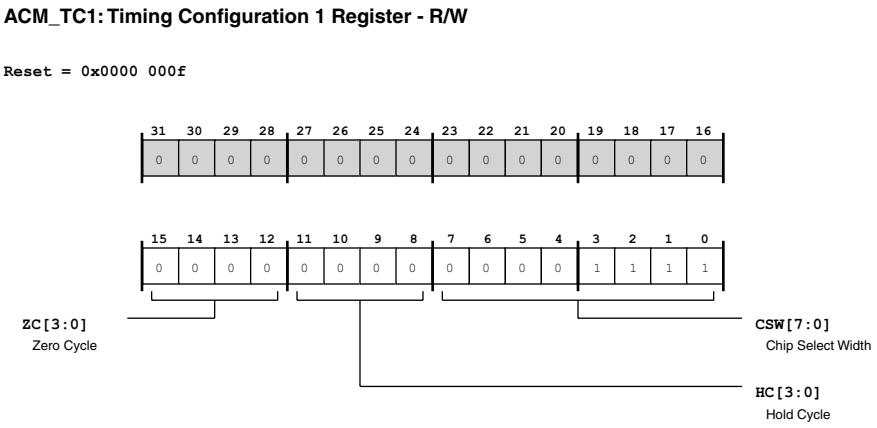


Figure 27-17: ACM_TC1 Register Diagram

Table 27-8: ACM_TC1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:12 (R/W)	ZC	Zero Cycle. The ACM_TC1.ZC bits select the ADC control zero duration. All ADC controls are driven low for ACM_TC1.ZCACM_CLK cycles.	
		0	0 Zero Cycles
		1	1 Zero Cycle
		15	15 Zero Cycles

Table 27-8: ACM_TC1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11:8 (R/W)	HC	Hold Cycle. The ACM_TC1.HC bits select the ADC control hold duration. All ADC controls are held after the inactive edge of CS for ACM_TC1.HCACM_CLK cycles.	
		0	0 Hold Cycles
		1	1 Hold Cycle
		15	15 Hold Cycles
7:0 (R/W)	CSW	Chip Select Width. The ACM_TC1.CSW bits select the active duration of CS. The CS is active for ACM_TC1.CSWACM_CLK +1 cycles.	
		0	1 Active CS Cycle
		1	2 Active CS Cycles
		15	16 Active CS Cycles
		255	256 Active CS Cycles

Status Register

The ACM_STAT register indicates the ACM event currently being serviced, any pending events, any missed events, and any missed triggers.

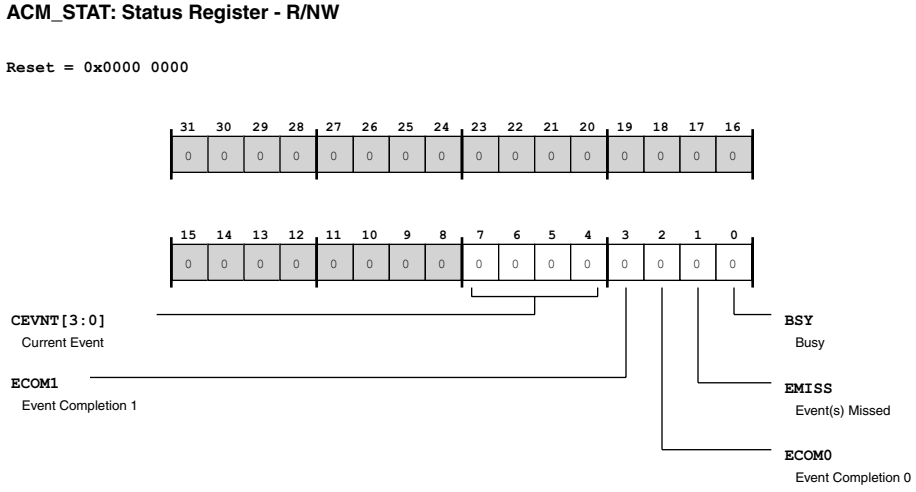


Figure 27-18: ACM_STAT Register Diagram

Table 27-9: ACM_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:4 (R/NW)	CEVNT	Current Event. The ACM_STAT.CEVNT bits indicates to which event (0 through 15) the ongoing access (current event, if any) corresponds.	
		0	Current Event Correspond to Event 0
		1	Current Event Correspond to Event 1
		15	Current Event Correspond to Event 15
3 (R/NW)	ECOM1	Event Completion 1. The ACM_STAT.ECOM1 bit indicates TMR1 event completion for all enabled ACM TMR1 events and the current trigger. The ACM clears this bit with each trigger.	
		0	No Status
		1	ACM TMR1 Events Complete
2 (R/NW)	ECOM0	Event Completion 0. The ACM_STAT.ECOM0 bit indicates TMR0 event completion for all enabled ACM TMR0 events and the current trigger. The ACM clears this bit with each trigger.	
		0	No Status
		1	ACM TMR0 Events Complete
1 (R/NW)	EMISS	Event(s) Missed. The ACM_STAT.EMISS bit indicates when an event is missed (any bits in ACM_MEVSTAT set). This bit is cleared by writing into the ACM_MEVSTAT register.	
		0	No Missed Event(s)
		1	Missed Event(s)
0 (R/NW)	BSY	Busy. The ACM_STAT.BSY bit indicates when the ACM is busy (an external sampling event in progress; CS is active or about to go active).	
		0	Idle
		1	Busy

Event Complete Status Register

The ACM_EVSTAT register identifies which enabled event has occurred for a particular trigger cycle. When an ACM_EVSTAT bit is cleared (=0), this status indicates that the ACM has not begun or completed conver-

sion for the corresponding event (conversion not done). When an ACM_EVSTAT bit is set (=1), this status indicates that the ACM has completed conversion for the corresponding event (conversion done).

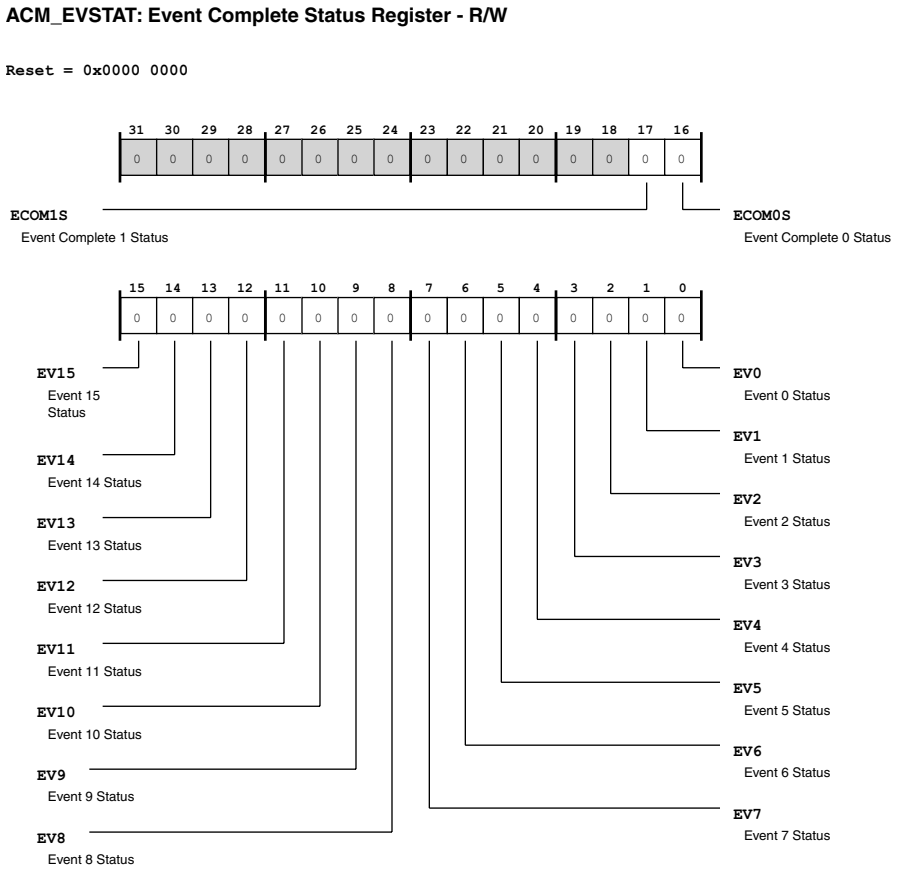


Figure 27-19: ACM_EVSTAT Register Diagram

Table 27-10: ACM_EVSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W1C)	ECOM1S	Event Complete 1 Status. The ACM_EVSTAT.ECOM1S bit indicates the state of the ACM_STAT.ECOM1 bit. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C and is not cleared by trigger.	
		0	No Status
		1	ACM_STAT.ECOM1 =1 Occurred

Table 27-10: ACM_EVSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W1C)	ECOM0S	Event Complete 0 Status. The ACM_EVSTAT.ECOM0S bit indicates the state of the ACM_STAT.ECOM0 bit. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C and is not cleared by trigger.	
		0	No Status
		1	ACM_STAT.ECOM0 =1 Occurred
15 (R/W1C)	EV15	Event 15 Status. The ACM_EVSTAT.EV15 bit indicates when the ACM has completed the conversion for event 15. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 15 Conversion
		1	Event 15 Conversion Done
14 (R/W1C)	EV14	Event 14 Status. The ACM_EVSTAT.EV14 bit indicates when the ACM has completed the conversion for event 14. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 14 Conversion
		1	Event 14 Conversion Done
13 (R/W1C)	EV13	Event 13 Status. The ACM_EVSTAT.EV13 bit indicates when the ACM has completed the conversion for event 13. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 13 Conversion
		1	Event 13 Conversion Done
12 (R/W1C)	EV12	Event 12 Status. The ACM_EVSTAT.EV12 bit indicates when the ACM has completed the conversion for event 12. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 12 Conversion
		1	Event 12 Conversion Done

Table 27-10: ACM_EVSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W1C)	EV11	Event 11 Status. The ACM_EVSTAT.EV11 bit indicates when the ACM has completed the conversion for event 11. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 11 Conversion
		1	Event 11 Conversion Done
10 (R/W1C)	EV10	Event 10 Status. The ACM_EVSTAT.EV10 bit indicates when the ACM has completed the conversion for event 10. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 10 Conversion
		1	Event 10 Conversion Done
9 (R/W1C)	EV9	Event 9 Status. The ACM_EVSTAT.EV9 bit indicates when the ACM has completed the conversion for event 9. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 9 Conversion
		1	Event 9 Conversion Done
8 (R/W1C)	EV8	Event 8 Status. The ACM_EVSTAT.EV8 bit indicates when the ACM has completed the conversion for event 8. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 8 Conversion
		1	Event 8 Conversion Done
7 (R/W1C)	EV7	Event 7 Status. The ACM_EVSTAT.EV7 bit indicates when the ACM has completed the conversion for event 7. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 7 Conversion
		1	Event 7 Conversion Done

Table 27-10: ACM_EVSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1C)	EV6	Event 6 Status. The ACM_EVSTAT.EV6 bit indicates when the ACM has completed the conversion for event 6. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 6 Conversion
		1	Event 6 Conversion Done
5 (R/W1C)	EV5	Event 5 Status. The ACM_EVSTAT.EV5 bit indicates when the ACM has completed the conversion for event 5. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 5 Conversion
		1	Event 5 Conversion Done
4 (R/W1C)	EV4	Event 4 Status. The ACM_EVSTAT.EV4 bit indicates when the ACM has completed the conversion for event 4. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 4 Conversion
		1	Event 4 Conversion Done
3 (R/W1C)	EV3	Event 3 Status. The ACM_EVSTAT.EV3 bit indicates when the ACM has completed the conversion for event 3. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 3 Conversion
		1	Event 3 Conversion Done
2 (R/W1C)	EV2	Event 2 Status. The ACM_EVSTAT.EV2 bit indicates when the ACM has completed the conversion for event 2. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 2 Conversion
		1	Event 2 Conversion Done

Table 27-10: ACM_EVSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	EV1	Event 1 Status. The ACM_EVSTAT.EV1 bit indicates when the ACM has completed the conversion for event 1. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 1 Conversion
		1	Event 1 Conversion Done
0 (R/W1C)	EV0	Event 0 Status. The ACM_EVSTAT.EV0 bit indicates when the ACM has completed the conversion for event 0. If set and the corresponding bit in ACM_EVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 0 Conversion
		1	Event 0 Conversion Done

Event Complete Interrupt Mask Register

The ACM_EVMSK register enables interrupts corresponding to status bits in the ACM_EVSTAT register. When an ACM_EVMSK bit is set (=1), an interrupt is generated when the corresponding event complete bit is set (bit in ACM_EVSTAT is set).

ACM_EVMSK: Event Complete Interrupt Mask Register - R/W

Reset = 0x0000 0000

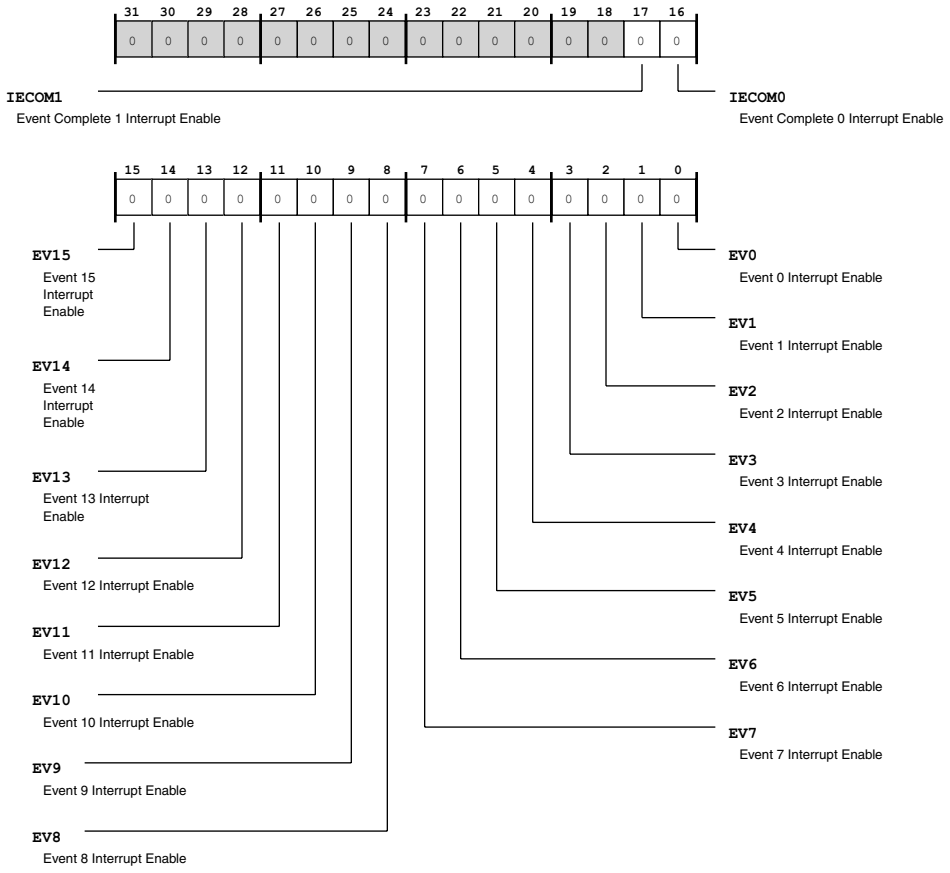


Figure 27-20: ACM_EVMSK Register Diagram

Table 27-11: ACM_EVMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	IECOM1	Event Complete 1 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
16 (R/W)	IECOM0	Event Complete 0 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Table 27-11: ACM_EVMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	EV15	Event 15 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
14 (R/W)	EV14	Event 14 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
13 (R/W)	EV13	Event 13 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
12 (R/W)	EV12	Event 12 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
11 (R/W)	EV11	Event 11 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
10 (R/W)	EV10	Event 10 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
9 (R/W)	EV9	Event 9 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
8 (R/W)	EV8	Event 8 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
7 (R/W)	EV7	Event 7 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
6 (R/W)	EV6	Event 6 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Table 27-11: ACM_EVMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	EV5	Event 5 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
4 (R/W)	EV4	Event 4 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
3 (R/W)	EV3	Event 3 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
2 (R/W)	EV2	Event 2 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
1 (R/W)	EV1	Event 1 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
0 (R/W)	EV0	Event 0 Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Missed Event Status Register

The ACM_MEVSTAT register indicates which enabled event has been missed for a particular trigger cycle. When an ACM_MEVSTAT bit is set (=1), this status indicates that corresponding event was missed. This status generates an interrupt if the corresponding bit in the ACM_MEVMSK register is set.

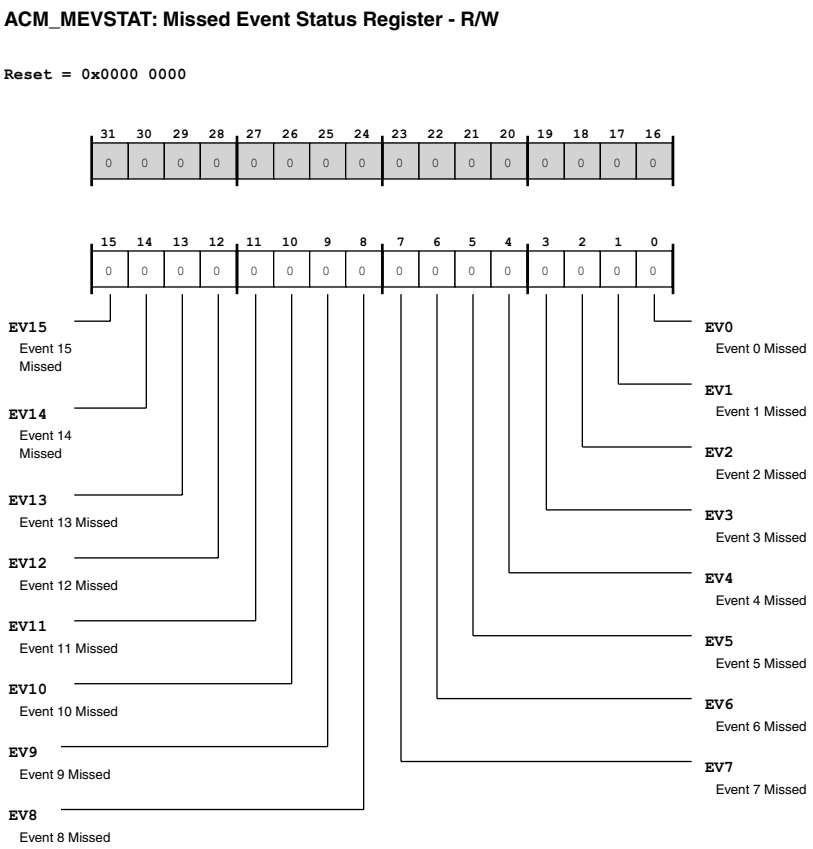


Figure 27-21: ACM_MEVSTAT Register Diagram

Table 27-12: ACM_MEVSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	EV15	Event 15 Missed. The ACM_MEVSTAT.EV15 bit indicates when an instance of event 15 has been missed since the last trigger. If set and the corresponding bit in ACM_MEVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 15 Missed Status
		1	Event 15 Missed
14 (R/W1C)	EV14	Event 14 Missed. The ACM_MEVSTAT.EV14 bit indicates when an instance of event 14 has been missed since the last trigger. If set and the corresponding bit in ACM_MEVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 14 Missed Status
		1	Event 14 Missed

Table 27-12: ACM_MEVSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1C)	EV13	Event 13 Missed. The ACM_MEVSTAT.EV13 bit indicates when an instance of event 13 has been missed since the last trigger. If set and the corresponding bit in ACM_MEVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 13 Missed Status
		1	Event 13 Missed
12 (R/W1C)	EV12	Event 12 Missed. The ACM_MEVSTAT.EV12 bit indicates when an instance of event 12 has been missed since the last trigger. If set and the corresponding bit in ACM_MEVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 12 Missed Status
		1	Event 12 Missed
11 (R/W1C)	EV11	Event 11 Missed. The ACM_MEVSTAT.EV11 bit indicates when an instance of event 11 has been missed since the last trigger. If set and the corresponding bit in ACM_MEVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 11 Missed Status
		1	Event 11 Missed
10 (R/W1C)	EV10	Event 10 Missed. The ACM_MEVSTAT.EV10 bit indicates when an instance of event 10 has been missed since the last trigger. If set and the corresponding bit in ACM_MEVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 10 Missed Status
		1	Event 10 Missed
9 (R/W1C)	EV9	Event 9 Missed. The ACM_MEVSTAT.EV9 bit indicates when an instance of event 9 has been missed since the last trigger. If set and the corresponding bit in ACM_MEVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 9 Missed Status
		1	Event 9 Missed

Table 27-12: ACM_MEVSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1C)	EV8	Event 8 Missed. The ACM_MEVSTAT.EV8 bit indicates when an instance of event 8 has been missed since the last trigger. If set and the corresponding bit in ACM_MEVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 8 Missed Status
		1	Event 8 Missed
7 (R/W1C)	EV7	Event 7 Missed. The ACM_MEVSTAT.EV7 bit indicates when an instance of event 7 has been missed since the last trigger. If set and the corresponding bit in ACM_MEVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 7 Missed Status
		1	Event 7 Missed
6 (R/W1C)	EV6	Event 6 Missed. The ACM_MEVSTAT.EV6 bit indicates when an instance of event 6 has been missed since the last trigger. If set and the corresponding bit in ACM_MEVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 6 Missed Status
		1	Event 6 Missed
5 (R/W1C)	EV5	Event 5 Missed. The ACM_MEVSTAT.EV5 bit indicates when an instance of event 5 has been missed since the last trigger. If set and the corresponding bit in ACM_MEVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 5 Missed Status
		1	Event 5 Missed
4 (R/W1C)	EV4	Event 4 Missed. The ACM_MEVSTAT.EV4 bit indicates when an instance of event 4 has been missed since the last trigger. If set and the corresponding bit in ACM_MEVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 4 Missed Status
		1	Event 4 Missed

Table 27-12: ACM_MEVSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W1C)	EV3	Event 3 Missed. The ACM_MEVSTAT.EV3 bit indicates when an instance of event 3 has been missed since the last trigger. If set and the corresponding bit in ACM_MEVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 3 Missed Status
		1	Event 3 Missed
2 (R/W1C)	EV2	Event 2 Missed. The ACM_MEVSTAT.EV2 bit indicates when an instance of event 2 has been missed since the last trigger. If set and the corresponding bit in ACM_MEVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 2 Missed Status
		1	Event 2 Missed
1 (R/W1C)	EV1	Event 1 Missed. The ACM_MEVSTAT.EV1 bit indicates when an instance of event 1 has been missed since the last trigger. If set and the corresponding bit in ACM_MEVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 1 Missed Status
		1	Event 1 Missed
0 (R/W1C)	EV0	Event 0 Missed. The ACM_MEVSTAT.EV0 bit indicates when an instance of event 0 has been missed since the last trigger. If set and the corresponding bit in ACM_MEVMSK is set (interrupt enabled), the condition generates an interrupt. This bit is W1C.	
		0	No Event 0 Missed Status
		1	Event 0 Missed

Missed Event Interrupt Mask Register

The ACM_MEVMSK register enables interrupts corresponding to status bits in the ACM_MEVSTAT register. When an ACM_MEVMSK bit is set (=1), an interrupt is generated when the corresponding event missed bit is set (bit in ACM_MEVSTAT is set).

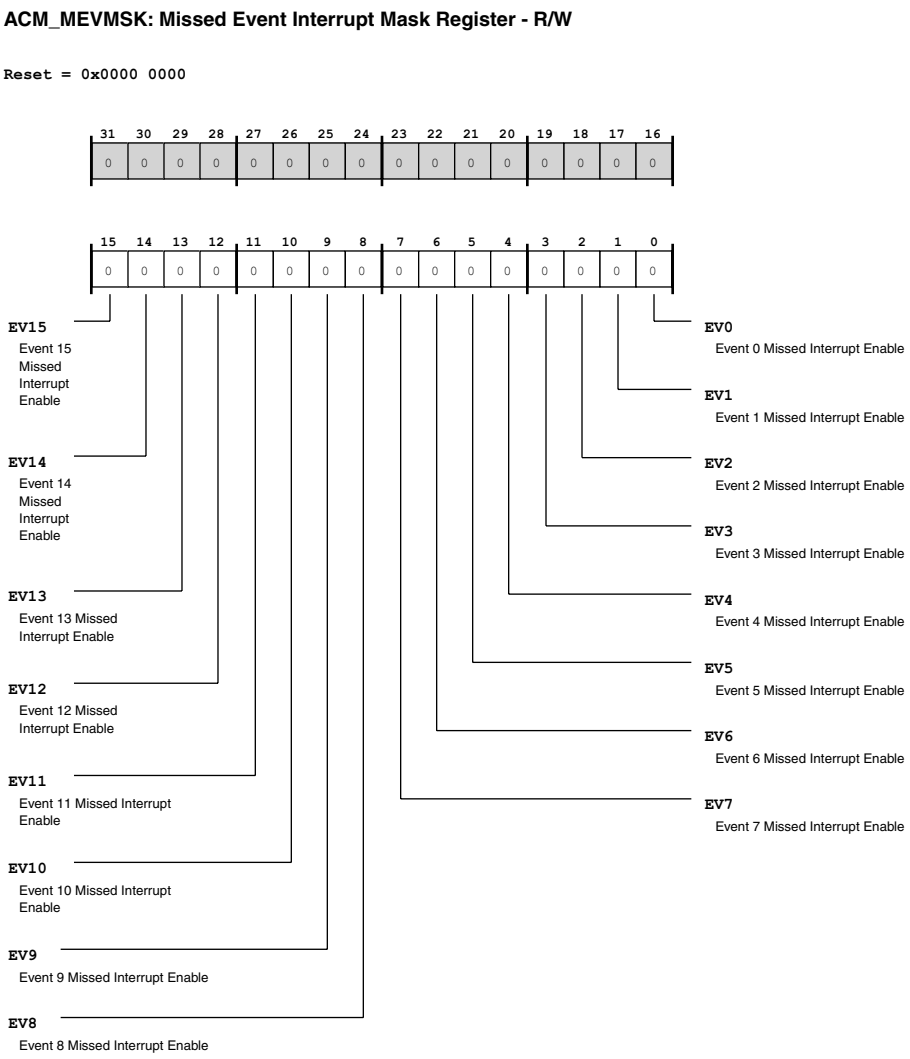


Figure 27-22: ACM_MEVMSK Register Diagram

Table 27-13: ACM_MEVMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	EV15	Event 15 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
14 (R/W)	EV14	Event 14 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Table 27-13: ACM_MEVMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	EV13	Event 13 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
12 (R/W)	EV12	Event 12 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
11 (R/W)	EV11	Event 11 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
10 (R/W)	EV10	Event 10 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
9 (R/W)	EV9	Event 9 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
8 (R/W)	EV8	Event 8 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
7 (R/W)	EV7	Event 7 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
6 (R/W)	EV6	Event 6 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
5 (R/W)	EV5	Event 5 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
4 (R/W)	EV4	Event 4 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Table 27-13: ACM_MEVMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	EV3	Event 3 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
2 (R/W)	EV2	Event 2 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
1 (R/W)	EV1	Event 1 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt
0 (R/W)	EV0	Event 0 Missed Interrupt Enable.	
		0	Disable (Mask) Interrupt
		1	Enable (Unmask) Interrupt

Event N Control Register

The ACM_EVCTLn registers each hold the ADC control value corresponding to the event related to the register. These registers each have an event enable bit, permitting selective enabling of a particular event.

Note that the ACM_EVCTLn register should not be programmed when an event is active. The register might give incorrect results. The user should program this register before giving trigger and should re-program the register after all the events are complete (ACM_STAT.ECOM1 or ACM_STAT.ECOM0 bit is set). Also note that even if none of the events are enabled in the ACM_EVCTLn register (for example, all ACM_EVCTLn.ENAEV =0), the ACM_STAT.ECOM0 or ACM_STAT.ECOM1 bit is set and an interrupt is raised (if unmasked) if a trigger is applied with the Timer enabled.

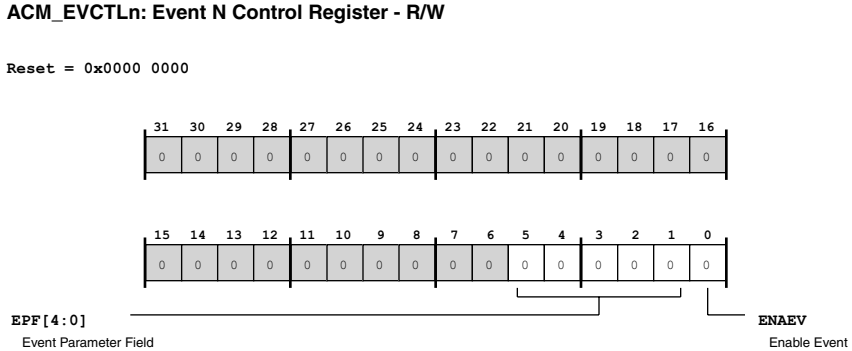


Figure 27-23: ACM_EVCTLn Register Diagram

Table 27-14: ACM_EVCTLn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5:1 (R/W)	EPF	Event Parameter Field. The ACM_EVCTLn.EPF bits select values for the ADC control pins (ACM_A0, ACM_A1, ACM_A2, and ACM_A3), which are output when the enabled event occurs. Selection of ACM_EVCTLn.EPF values are based on the type of ADC, usage mode, and other items. For more information, see the operating modes section. Note that all ACM_EVCTLn.EPF bits have the same external pin timing.	
0 (R/W)	ENAEV	Enable Event. The ACM_EVCTLn.ENAEV bit causes a sampling event to occur to the ADC with the ADC controls selected by the ACM_EVCTLn.EPF field when an event (time comparison match or other external trigger) occurs. If disabled, the corresponding event has no significance, and the control values is not used.	
		0	Disable Event
		1	Enable Event

Event N Time Register

The ACM_EVTIMEn registers each hold a 32-bit event time value. There are 16 event time registers: 8 are assigned to each ACM timer, if both timers are enabled. If only one timer is enabled, all 16 of the ACM_EVTIMEn registers are assigned to the enabled timer.

Note that the ACM_EVTIMEn register should not be programmed when an event is active. The register might give incorrect results. The user should program this register before giving trigger and should re-program the register after all the events are complete (ACM_STAT.ECOM1 or ACM_STAT.ECOM0 bit is set). Also note that even if none of the events are enabled in the ACM_EVTIMEn register (for example, all ACM_EVCTLn.ENAEV =0), the ACM_STAT.ECOM0 or ACM_STAT.ECOM1 bit is set and an interrupt is raised (if unmasked) if a trigger is applied with the Timer enabled.

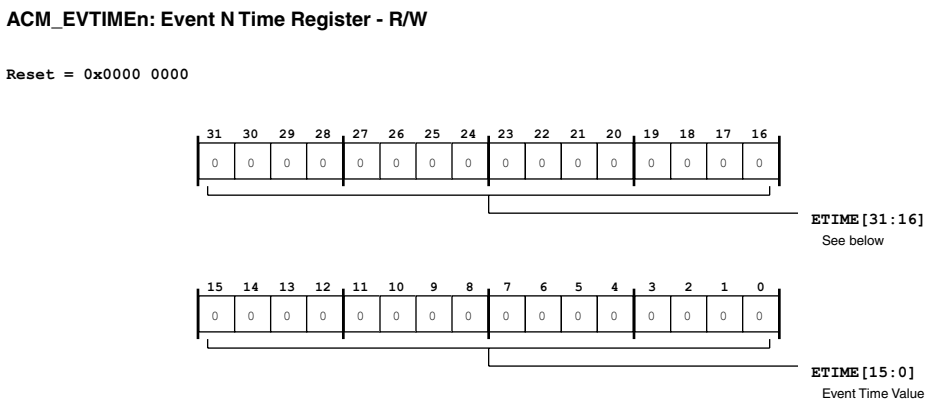


Figure 27-24: ACM_EVTIMEn Register Diagram

Table 27-15: ACM_EVTIMEn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ETIME	Event Time Value.

Event N Order Register

The ACM_EVORDn registers hold the order in which ADC data has been captured corresponding to the event. These registers can store the order of 256 data captures at a stretch. The ACM_EVORDn registers also have status bits indicating whether the particular event is missed/completed in the trigger cycle.

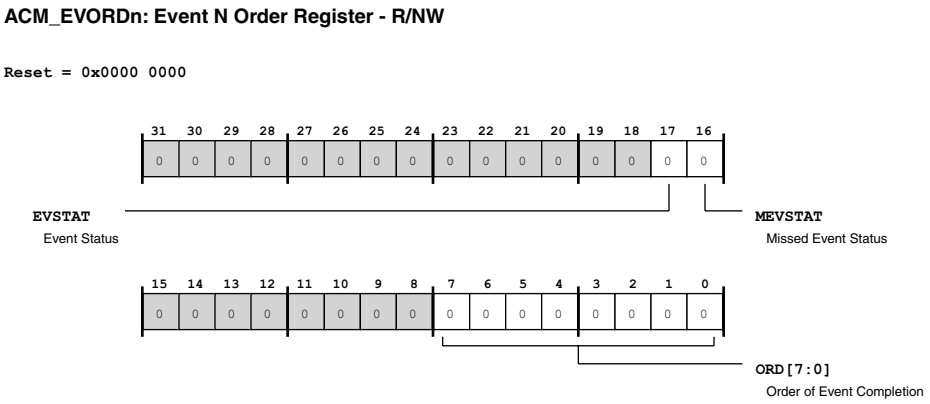


Figure 27-25: ACM_EVORDn Register Diagram

Table 27-16: ACM_EVORDn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/NW)	EVSTAT	Event Status. The ACM_EVORDn.EVSTAT bit reflects the state of the corresponding event's bit in the ACM_EVSTAT register.	
16 (R/NW)	MEVSTAT	Missed Event Status. The ACM_EVORDn.MEVSTAT bit reflects the state of the corresponding event's bit in the ACM_MEVSTAT register.	
7:0 (R/NW)	ORD	Order of Event Completion. The ACM_EVORDn.ORD bits indicate the order of event completion with 0 indicating the 1st event completed (after the ACM was enabled or after the ACM_CTL.ORST bit was set) and with 255 indicating the 256th event completed.	
		0	1st Event Completed
		1	2nd Event Completed
		255	256th Event Completed

Timer 0 Register

The ACM_TMR0 register contains the active count value for ACM timer 0. This read-only value may be accessed at any time.

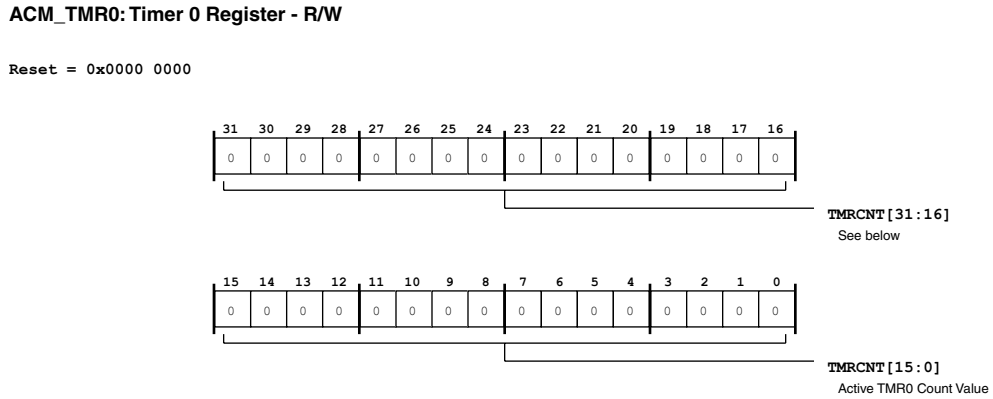


Figure 27-26: ACM_TMR0 Register Diagram

Table 27-17: ACM_TMR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	TMRCNT	Active TMR0 Count Value.

Timer 1 Register

The ACM_TMR1 register contains the active count value for ACM timer 1. This read-only value may be accessed at any time.

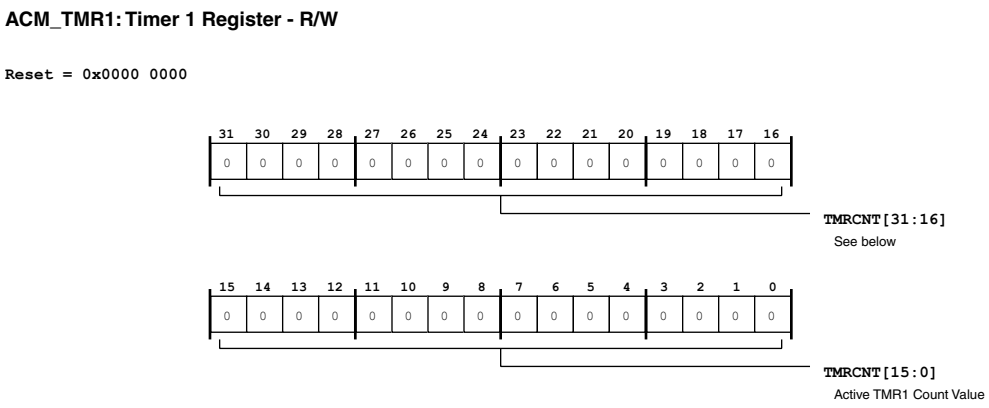


Figure 27-27: ACM_TMR1 Register Diagram

Table 27-18: ACM_TMR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	TMRCNT	Active TMR1 Count Value.

28 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 a variety of I/O peripheral interconnection schemes to I/O peripheral devices as well as co-processing and multiprocessing schemes.

The processor's link ports support 8-bit wide data transfers. The link port pins are multiplexed in the GPIO ports. For information on processor multiplexing, see the processor specific data sheet.

Link ports can operate independently and simultaneously, allowing glueless high-speed connectivity of up to four external processors.

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. This data can be directly read by the processor or transferred via 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-BF60x LP Register List

The LP are 8-bit wide ports, which can connect to another processor or peripheral LP. These ports allow a variety of interconnection schemes to I/O peripheral devices as well as co-processing and multiprocessing

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

Table 28-1: ADSP-BF60x LP Register List

Name	Description
LP_CTL	Control Register
LP_STAT	Status Register
LP_DIV	Clock Divider Value
LP_TX	Transmit Buffer
LP_RX	Receive Buffer
LP_TXIN_SHDW	Shadow Input Transmit Buffer
LP_TXOUT_SHDW	Shadow Output Transmit Buffer

ADSP-BF60x LP Interrupt List

Table 28-2: ADSP-BF60x LP Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
LP0 DMA Channel	72	13	LEVEL
LP0 Status	73		LEVEL
LP1 DMA Channel	74	14	LEVEL
LP1 Status	75		LEVEL
LP2 DMA Channel	76	15	LEVEL
LP2 Status	77		LEVEL
LP3 DMA Channel	78	16	LEVEL
LP3 Status	79		LEVEL

ADSP-BF60x LP Trigger List

Table 28-3: ADSP-BF60x LP Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
LP0 DMA Channel	35	PULSE/EDGE

Table 28-3: ADSP-BF60x LP Trigger List Trigger Masters (Continued)

Description	Trigger ID	Sensitivity
LP1 DMA Channel	36	PULSE/EDGE
LP2 DMA Channel	37	PULSE/EDGE
LP3 DMA Channel	38	PULSE/EDGE

Table 28-4: ADSP-BF60x LP Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
LP0 DMA Channel	35	
LP1 DMA Channel	36	
LP2 DMA Channel	37	
LP3 DMA Channel	38	

ADSP-BF60x LP DMA List

Table 28-5: ADSP-BF60x LP DMA List DMA Channel List

Description	DMA Channel
LP0 DMA Channel	DMA13
LP1 DMA Channel	DMA14
LP2 DMA Channel	DMA15
LP3 DMA Channel	DMA16

Block Diagram

The block diagram of a link port is shown in the following figure.

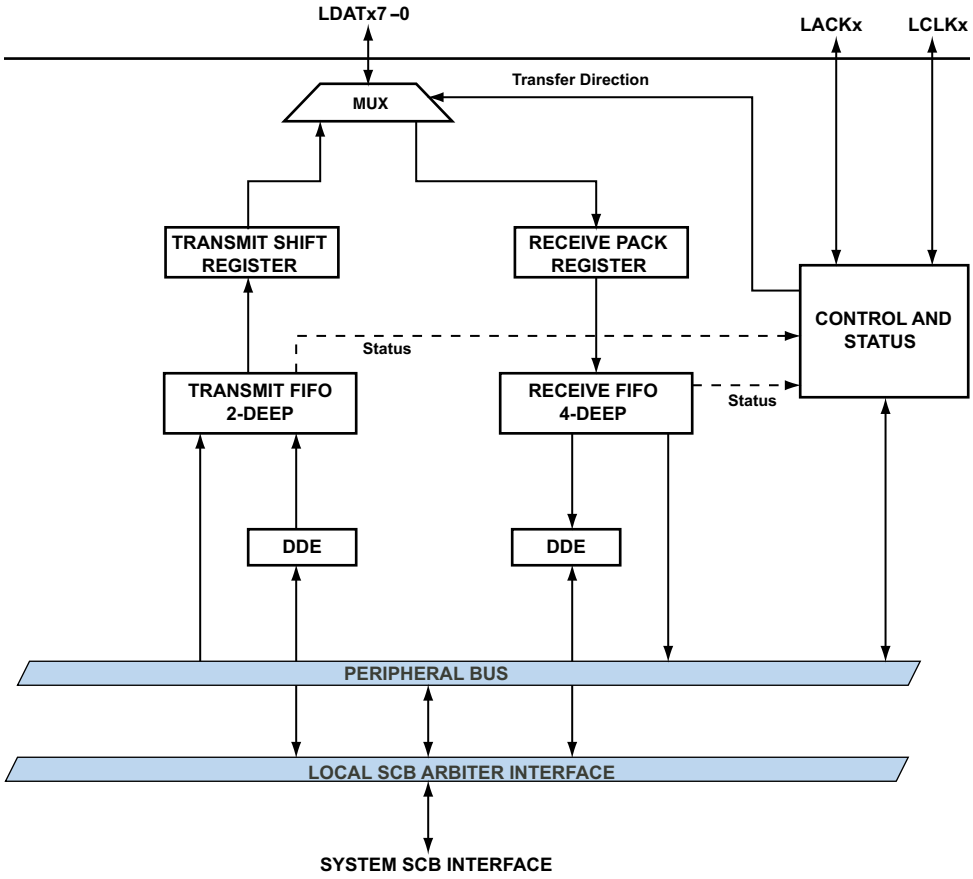


Figure 28-1: Link Port Block Diagram

External Connections

As shown in the following 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.

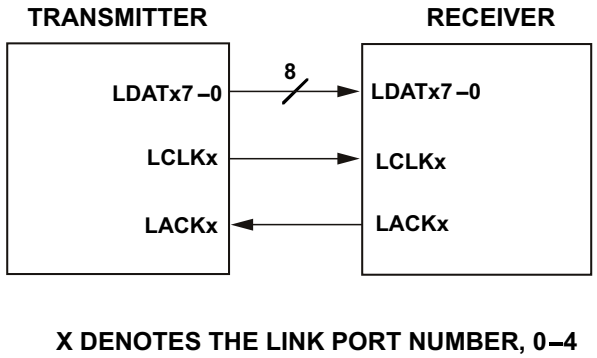


Figure 28-2: Link Port Pin Connections

External pull-downs should be used for the LP_CLK and LP_ACK pins so that the transmitter and receiver can be enabled, 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. These FIFOs can be accessed by the core MMR access bus as well as through the distributed DMA engines (DDE) through the system cross bar (SCB) interface. The LP_CTL.TRAN bit determines whether the module is enabled for transmit or receive operation.

Architectural Concepts

This section describes the following link port architectural concepts.

- *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 falling edge of LP_CLK driven by the transmitter is used by the receiver 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 de-asserted at that time, 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, LP_CLK is driven low by the transmitter and the transmission of the next word starts. If the transmit buffer is empty, LP_CLK remains low until the buffer is refilled, regardless of the state of LP_ACK.

The LP_ACK signal may de-assert when it anticipates that the buffer may fill. The LP_ACK signal is reasserted by the receiver 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 *SCLK*.
- 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 *SCLK*.
- When the link port is disabled the data, clock and acknowledge signals are three-stated.

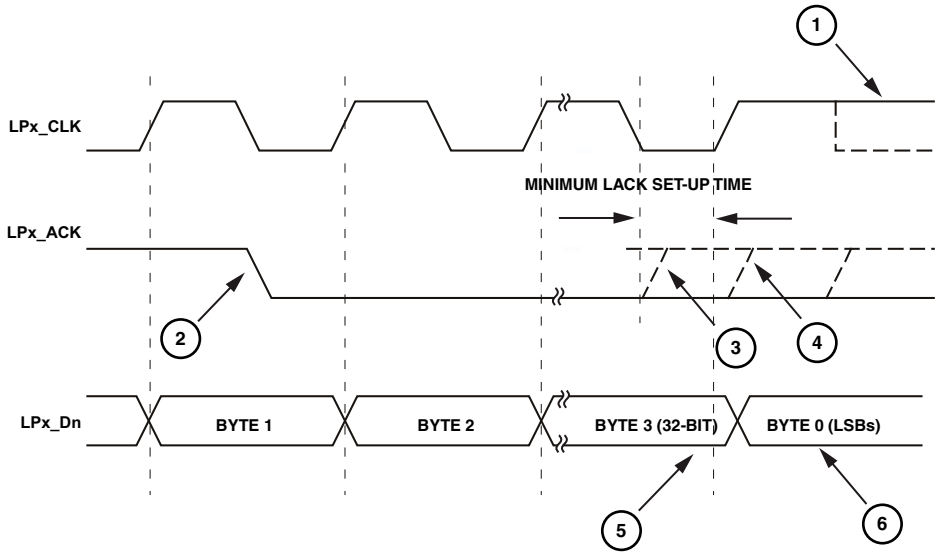


Figure 28-3: Link Port Communication and Handshake Waveform

The following list describes the stages shown in the figure above.

1. LP_CLK stays high at byte 0 if LP_ACK is sampled low on the previous LP_CLK rising edge. LP_CLK high indicates a stall.
2. The LP_ACK signal may de-assert 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 de-asserted. The transmitter does not send the following word.
6. Transmission of data for next word is held until LP_ACK is asserted.

The LP_ACK signal is sampled by the transmitter and if it 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 *SCLK*, is further

synchronized by one more *SCLK* stage and then this synchronized signal is given to the subsequent logic. The LP_CLK falling edge is aligned with *SCLK* falling edge in a 1:1 clock ratio mode and with the *SCLK* 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 *SCLK* cycles.

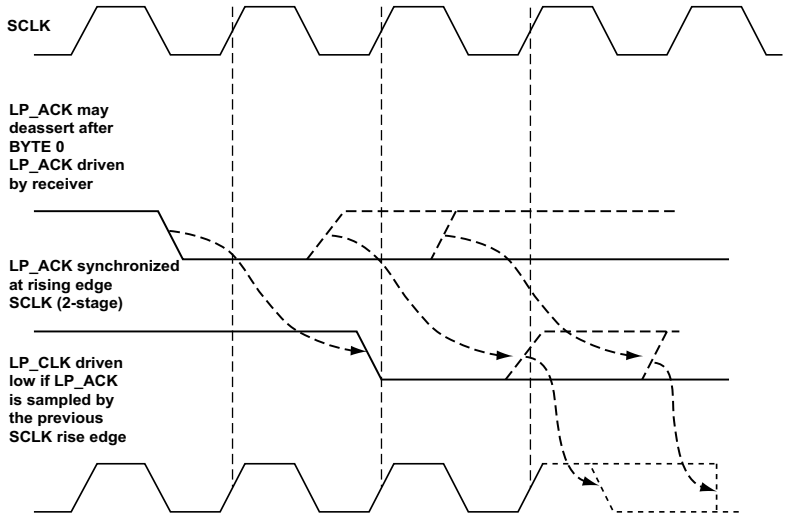


Figure 28-4: LP_ACK Synchronization for SCLK:LP_CLK=1:1

In the following figure synchronizing time is guaranteed to be 2 *SCLK* cycles.

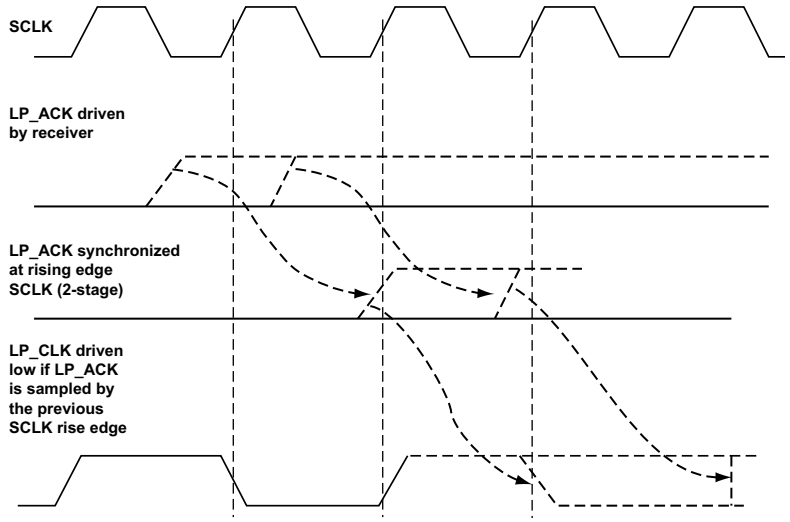


Figure 28-5: LP_ACK synchronization for SCLK: LP_CLK=1:2, 1:4 and Up

The frequency of the link port clock (LP_CLK) is determined by the value programmed in the LP_DIV register at the transmitter. 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 following figure shows this relationship.

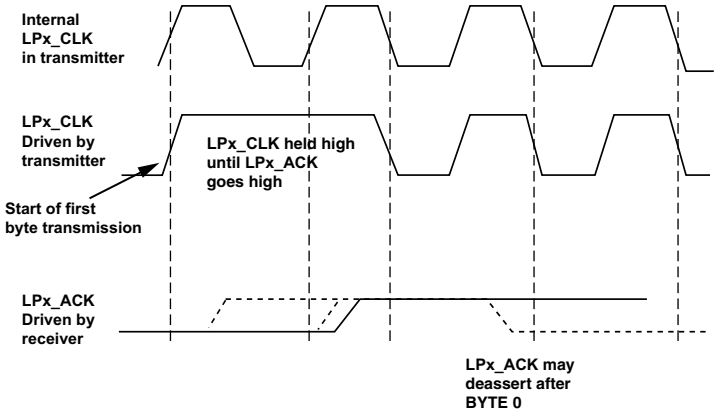


Figure 28-6: Relationship Between Internal Link Port Clock and Link Port Clock at the Pins

FIFO Buffers

When a link port is configured as transmitter, a 2-deep FIFO buffer is used. A shift register unpacks the single 32-bit word to four 8-bit data bytes. As the FIFO has space for more data, a new DMA request is made. If the FIFO becomes empty, the LP_CLK signal is de-asserted. The core can access FIFO through the LP_TX register.

Three writes (2 stage FIFO and 1 shift register) can be made to the transmit buffer by the core or DMA before it signals a full condition. The status of the FIFO is reflected by the LP_STAT.FFST bit field but the shift register full/empty condition is not provided. However, the program may poll the LP_STAT.LPBS bit to discover the if link port is driving data from the shift register to the pins or not. 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 overflows, any further write in to the link port buffer overwrites the input stage of the FIFO.

NOTE: The transmit FIFO can also be read out by the core via the data LP_TX register.

NOTE: If the transmitter is disabled while performing writes to the transmit FIFO, a FIFO full condition is signalled after two writes.

Shadow registers have been provided for the transmit buffer registers. Using these shadow registers, both stages of the 2-deep FIFO may be read without updating the status registers. The LP_TXIN_SHDW register corresponds to the input stage of the FIFO and the LP_TXOUT_SHDW register corresponds to the output stage of the FIFO as shown in the following figure.

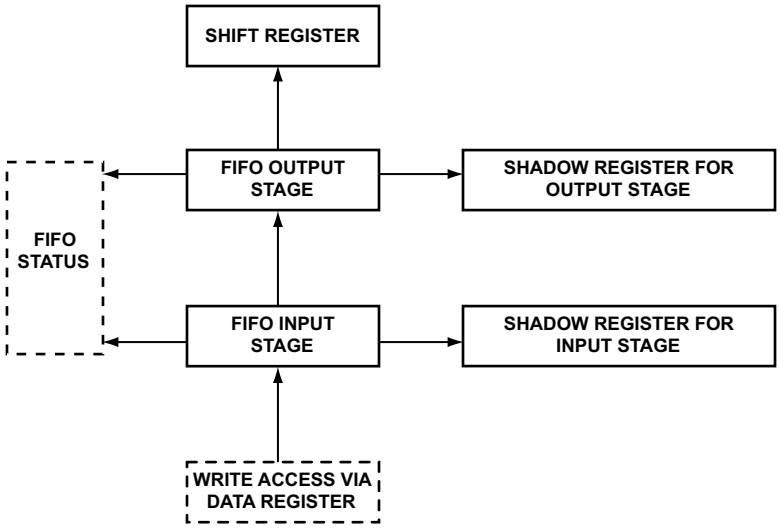


Figure 28-7: Transmit FIFO path

When a link port is configured as receiver, data is transferred to the core or DMA from the full 4-deep receive FIFO. An internal packing register performs the packing of data to 32 bits. Four reads can occur from the receive buffer by the core or DMA before it signals an empty condition. Status of the 4-deep read buffer FIFO is reflected by the LP_STAT.FFST bits. 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 and only the data retained in the receive FIFO can be retrieved further.

The LP_ACK output signal is driven low by the receiver once the first byte of data for the last but one empty slot (in the 4-deep FIFO) is received. This is to prevent data loss due to the transmitter starting transmission of the next word before the LP_ACK signal reaches the transmitter (due to the larger delay in synchronization). This guarantees that even after allowing for the extra synchronization cycle in the transmitter and receiver, there is no overflow in the receive FIFO. The following figure shows how FIFO slots influence the acknowledge signal generation. The greyed sections show received data and the white sections show empty locations where the decision to pull LP_ACK high is taken.

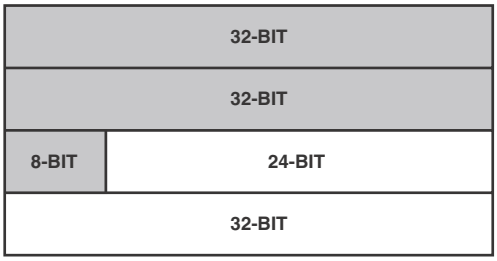


Figure 28-8: LACK Generation Based on Receive FIFO Status

NOTE: A 4-deep receive FIFO can be used only under a worst case situation as mentioned above. In all other cases, the FIFO must be thought of having only a 3-deep stage because LP_ACK 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 may read the FIFO which can cause unexpected problems in data transfers. This can only happen during an emulation event (typically hitting a breakpoint or single-stepping), when the emulator issues core reads via JTAG. To workaround for 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 may be enabled at different times. External pull-downs should be used for the LP_CLK and LP_ACK signals.

If the receiver is enabled before the transmitter, the LP_CLK signal of the transmitter is held low by the external pull-down and 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 it is enabled by the application.

If the transmitter is enabled before the receiver, LP_ACK signal of the receiver is held low by the external pull-down and transmission is held off as shown in the figure below. 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 receiver is enabled with LP_ACK driven high subsequently by hardware.

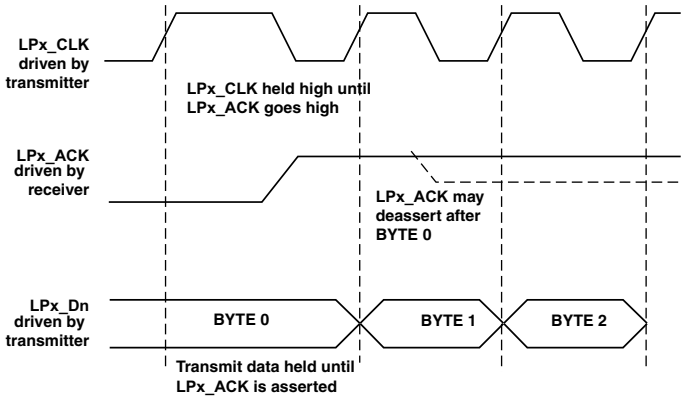


Figure 28-9: Enable the Transmitter Before the Receiver

NOTE: Service request interrupts/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 (*SCLK*). 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 relationship between the link port clock frequency, the *SCLK* frequency, and the LP_DIVvalue is expressed by the formula shown below.

- $f_{LP\_CLK} = f_{SCLK} <$ or $= f_{LP\_CLK\text{-}MAX}$ if DIV = 0
- $f_{LP\_CLK} = f_{SCLK}/(2 \times DIV)$ if DIV > 0

Where: f_{LP_CLK} = link clock frequency, $f_{LP\_CLK\ -MAX}$ = link clock maximum frequency, and f_{SCLK} = 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 example, if the *SCLK* frequency is 125 MHz and the limit for LP_CLK operation is 83 MHz, LP_DIV should be greater than or equal to 1, so that the LP_CLK frequency is less than or equal to 83 MHz. For supported frequencies, see the product specific data sheet.

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 figures show 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.

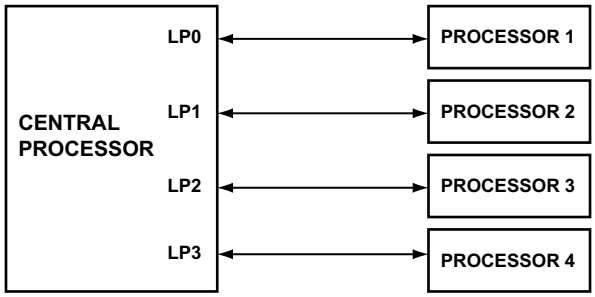


Figure 28-10: Central Processor Based Model

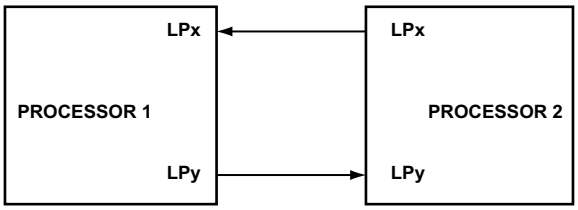


Figure 28-11: Link Port Full-duplex transfer Model

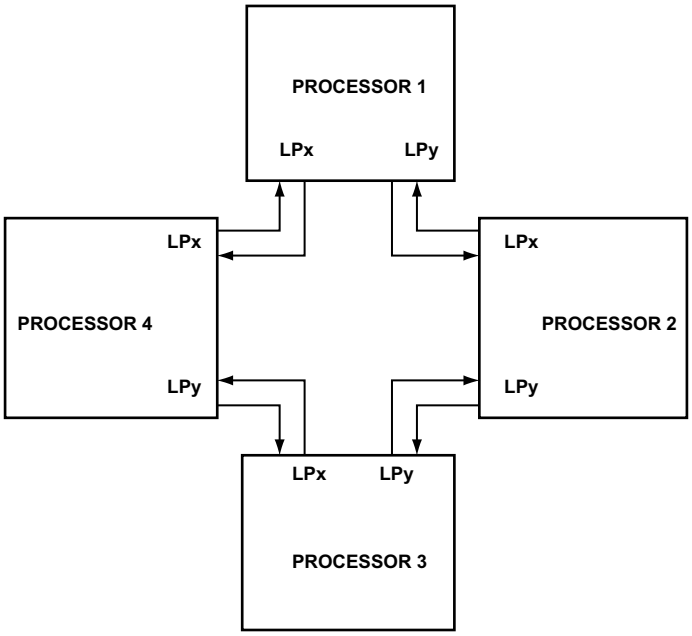


Figure 28-12: Link Port Ring Model

The link port protocol doesn't include built-in support for multiple masters, although there may be situations where multiple devices try to become the bus master at the same time. Multi-master conflicts may 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 may assert its LP_ACK signal to get the master's attention. The master knows, through the software protocol, whether it is supposed to respond with actual data or whether it is being asked for the token. If the master wishes to give up the token, it may 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 and because the transmitter and receiver are expecting a code word, this does not need to be exclusive of normal data transmission. If the master wishes to give up the token, it may 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 is being transferred. The application must assume the disabled state of the other end, and take appropriate action.

Similarly, in a multi-processing environment where the receiver did not read its full FIFO for an extended time (owing to internal bus arbitrations for example), the transmitter may require software or a peripheral timer based time out 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 data transfer modes, using the core and using DMA are explained in the following sections.

- *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, then the internal FIFO buffers may be written or read by the processor core as a memory-mapped register through the MMR access bus. In order to avoid FIFO overflow or underflow, the core should 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 link port data request interrupt. The link port 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 `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 below.

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 link port data pins. The FIFO detects this 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 link port data pins. 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 interrupts and status signals are used with the link ports.

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 with respect to FIFO conditions for data transfer and status interrupts are asserted when a service request status or an overflow status is set. Each of these interrupts are explained below.

- **Data Request Interrupt.** Asserted if the FIFO is not full in transmission mode and the FIFO is not empty in reception mode. This serves as a core triggered interrupt in non-DMA mode and as the DMA interrupt request in DMA mode. Generation of this interrupt is tied to the LP_STAT.FFST (link port buffer status bit).
- **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 needs to 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 in is generated, and the transmitter can enable itself for data transfer with the receiver. Note that a pull-down on LP_ACK is required for proper function of this feature.
- **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 needs to receive data through the connected link port. The transmitter does so by driving the first data out. When the LP_CLK of the disabled receiver link port is detected high, a LP_STAT.LRRQ interrupt in is generated, and the receiver can further enable itself for data transfer with the transmitter. Note that a pull-down on the LP_CLK signal is required for proper function of this feature.
- **Link Port Receive Overflow Interrupt (LPOVF)**. Generated when the receiver FIFO overflows and is enabled by setting the LP_CTL.ROVFMSK bit. This may happen if the transmitter continues to transmit data even though the receiver has de asserted 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.

Service interrupts and the overflow interrupt can be masked by setting the corresponding mask bits in LP_CTL register, as these 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 should read the LP_STAT register bits to determine the type of interrupt. Note that 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 LPSTAT register.

- **Transfer Status signals**. The link port bus status (LP_STAT.LPBS) gives the status of bus busy/idle condition, when the link port is configured as transmitter. The LP_STAT.LPBS is kept high if data is being driven by the link port in to the link port pins. Programs may poll this bit after polling the LP_STAT.FFST bit to safely disable the link port.

  The link buffer status (LP_STAT.FFST) field directly indicates the status of the FIFO (including empty/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 receive request status (LP_STAT.LRRQ) and link port transmit request status (LP_STAT.LTRQ) bits indicate that an external receiver wants to receive data (in case the link port is a disabled transmitter) or 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 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 may occur if the transmitter continues to transmit data even though the receiver has de asserted 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 describe 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. Install interrupt handlers for DMA and for transfer status (service request interrupt).
3. 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.
4. Program the link port clock divider by writing a value to the LP_DIV register.
5. If using DMA stop mode/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, Psize=1, Msize=4, interrupt generation and memory read).
6. Wait for the link port receiver (connected externally) to be enabled. The application can wait for the 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 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 a transfer completion interrupt.
11. 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/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, Psize=1, Msize=4, interrupt generation and memory write).
5. If using DMA array mode/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, Psize=1, 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, Psize=1, 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. Install interrupt handlers for data transfer and for transfer status (service request interrupt). The interrupt handlers for data transfer are the same source/ID as the DMA interrupt line in the SEC.
3. 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).
4. Program the link port clock divider by writing a value in to the LP_DIV register.
5. Wait for the link port receiver (connected externally) to be enabled. The application can wait for a transmit service request interrupt to assert.
6. Clear the transmit service request interrupt status by writing 1 to the LP_STAT.LTRQ bit.
7. Enable the link port by setting the LP_CTL.EN bit.
8. The data request interrupt is asserted whenever there is free space in the FIFO. The application can write to theLP_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/ID as the DMA interrupt line in the SEC).

3. Configure link port for reception (clear LP_CTL.TRAN bit) and enable receive request interrupt mask (set LP_CTL.RRQMSK bit).
4. Wait for the link port transmit (connected externally) to be enabled with subsequent transmission of data. 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-BF60x LP Register Descriptions

Link Port (LP) contains the following registers.

Table 28-6: ADSP-BF60x LP Register List

Name	Description
LP_CTL	Control Register
LP_STAT	Status Register
LP_DIV	Clock Divider Value
LP_TX	Transmit Buffer
LP_RX	Receive Buffer
LP_TXIN_SHDW	Shadow Input Transmit Buffer
LP_TXOUT_SHDW	Shadow Output Transmit Buffer

Control Register

The LP_CTL register provides LP interrupt masking, selection of transfer direction, and link port enable.

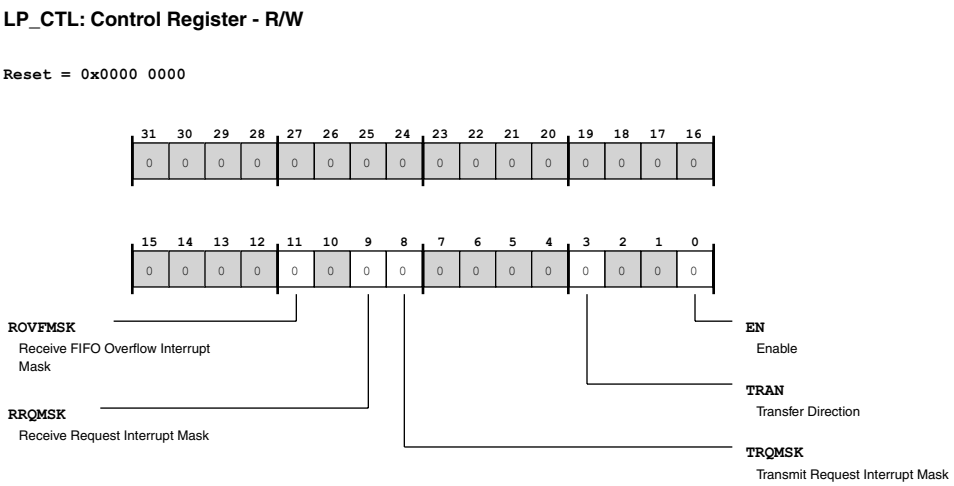


Figure 28-13: LP_CTL Register Diagram

Table 28-7: 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.

Table 28-7: LP_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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 Disable linkport
		1	Enable linkport Enable linkport

Status Register

The LP_STAT register provides status information on link port interrupts, FIFO, buses, and receive/transmit requests.

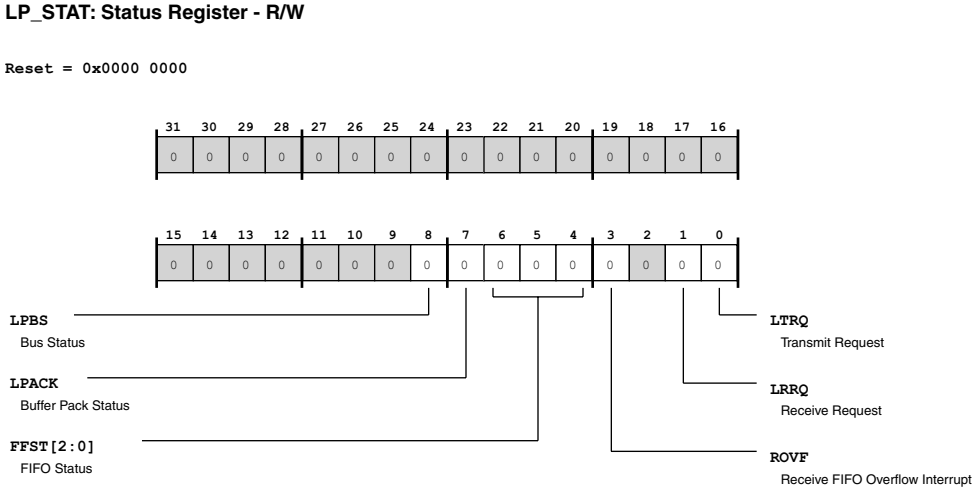


Figure 28-14: LP_STAT Register Diagram

Table 28-8: LP_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	LPBS	Bus Status. The LP_STAT.LPBS bit indicates the LPDAT bus status. LP_STAT.LPBS is kept high if data is being driven by the link port into the LP_Dn 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 LP_STAT.LPACK 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 LP_STAT.LPACK 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 LP_STAT.LPACK is high during unpacking.	
		0	Packing Complete Packing done
		1	Packing Incomplete Packing is in progress

Table 28-8: LP_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6:4 (R/NW)	FFST	FIFO Status. The LP_STAT.FFST 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
		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	TX - Reserved; RX - Reserved RX reserved.. TX reserved.
		6	TX - FIFO Full; RX - Reserved RX reserved.. TX reserved.
		7	TX - Reserved; RX - Reserved RX reserved.. TX 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).	

Clock Divider Value

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.

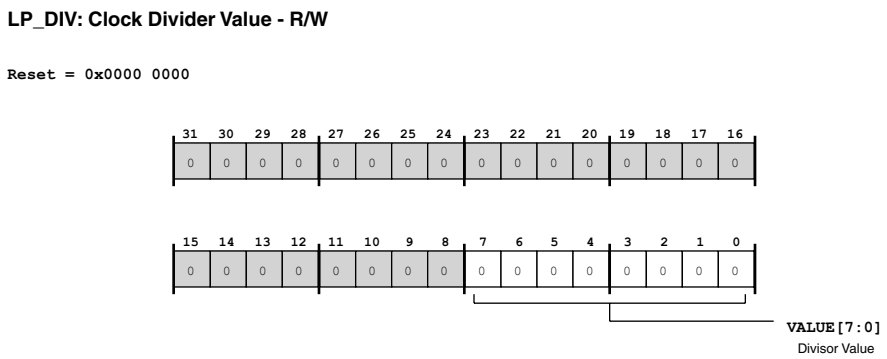


Figure 28-15: LP_DIV Register Diagram

Table 28-9: LP_DIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Divisor Value. The LP_DIV.VALUE bits select the clock divider (relating the LP' internally generated clock (LCLK) to the system clock (SCLK). The LP_DIV.VALUE should be programmed prior to LP enable. For LP_DIV.VALUE = 0, LCLK = SCLK For LP_DIV.VALUE = xxxxxxxx, LCLK = SCLK / (2 x DIV)

Transmit Buffer

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.

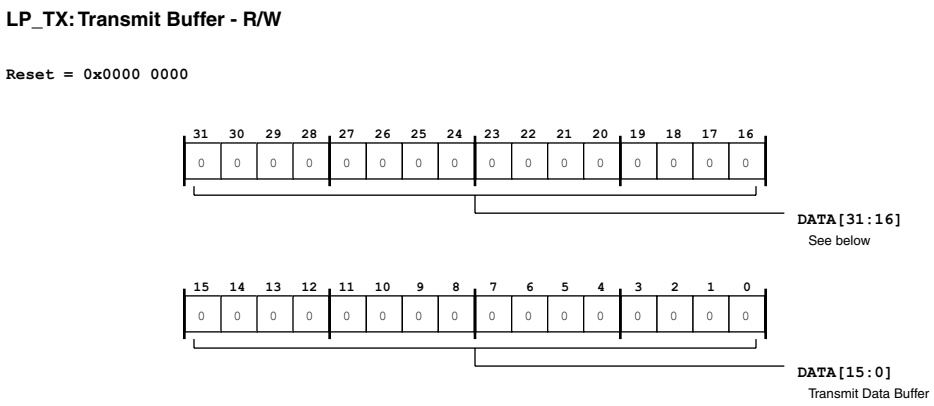


Figure 28-16: LP_TX Register Diagram

Table 28-10: LP_TX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Transmit Data Buffer.

Receive Buffer

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.

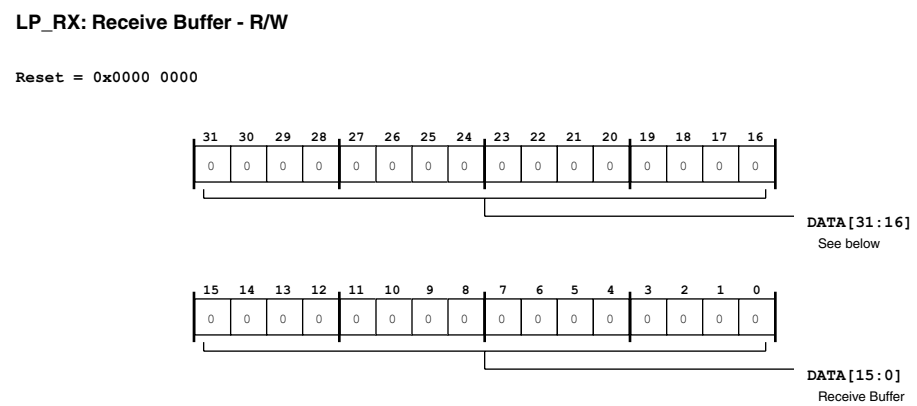


Figure 28-17: LP_RX Register Diagram

Table 28-11: LP_RX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Receive Buffer.

Shadow Input Transmit Buffer

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.

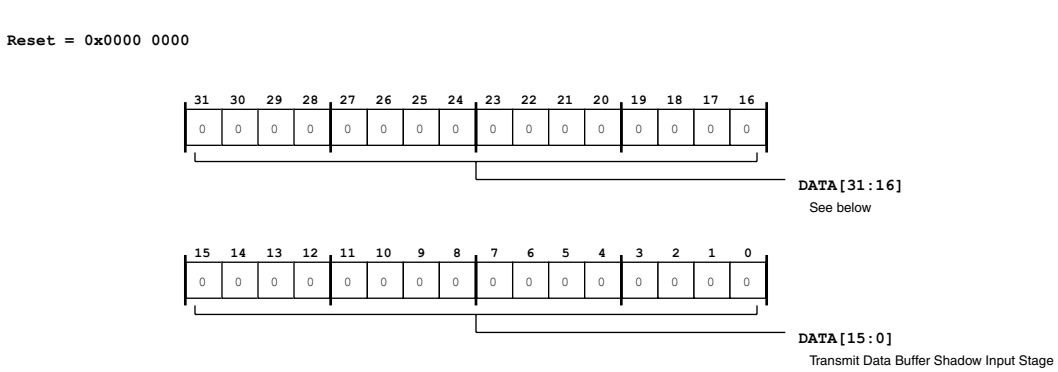


Figure 28-18: LP_TXIN_SHDW Register Diagram

Table 28-12: 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

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.

LP_TXOUT_SHDW: Shadow Output Transmit Buffer - R/NW

Reset = 0x0000 0000

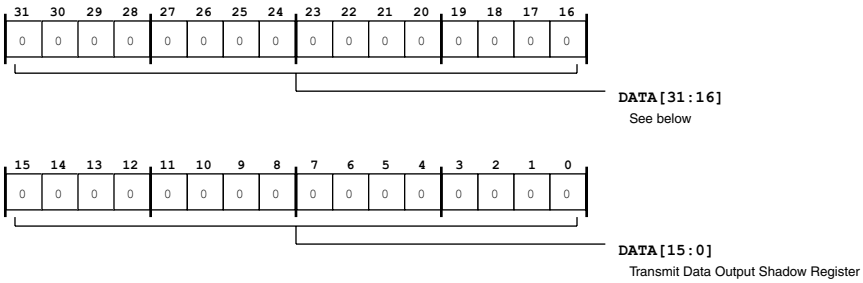


Figure 28-19: LP_TXOUT_SHDW Register Diagram

Table 28-13: LP_TXOUT_SHDW Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	DATA	Transmit Data Output Shadow Register.

29 Video Subsystem (VID)

The ADSP-BF60x processor features a sophisticated video subsystem (VID) consisting of the following functional blocks.

- Three video ports (PPI)
- The pixel compositor (PIXC)
- The pipelined vision processor (PVP)
- The video interconnect
- 18 DMA channels

The enhanced parallel peripheral interfaces (EPPIs) can operate in video input and video output modes, and also support several general-purpose modes of operation. The EPPIs are aware of video frame synchronization, blanking concepts and color formats. They can receive up to 16-bit video data directly from video sensors (cameras) and also directly control displays on the output.

The pixel compositor (PIXC) supports color space conversion and alpha blending for video overlays.

The pipelined vision processor (PVP) is a framework for various vision processing elements, which are targeting mainly edge and object detection strategies.

The video interconnect is a local, distributed bus system, which interconnects the EPPI ports, the PIXC and the PVP.

VID Features

The following is a brief list of the features of the video subsystem.

- Bandwidth-saving pre-processing and post-processing on video inputs and outputs
- Video analytics and color-space conversion during video reception
- Alpha blending during video transmission
- Concurrent memory-to memory co-processing

VID Functional Description

The following sections provide a functional description of the video subsystem.

ADSP-BF60x VID Register List

The VID selects connections for the PVP and PPI modules, configuring connections to use these peripherals for video operations. Also, the VID selects broadcast or non-broadcast mode for handling the receive Y channel data. Note that these connections are static and have to be configured before enabling the PPI, PIXC, and PVP blocks. A set of registers govern VID operations. For more information on VID functionality, see the VID register descriptions.

Table 29-1: ADSP-BF60x VID Register List

Name	Description
VID_CONN	Video Subsystem Connect Register

VID Block Diagram

The following figure shows an overview of the video subsystem including the PIXC, PVP, and EPPI.

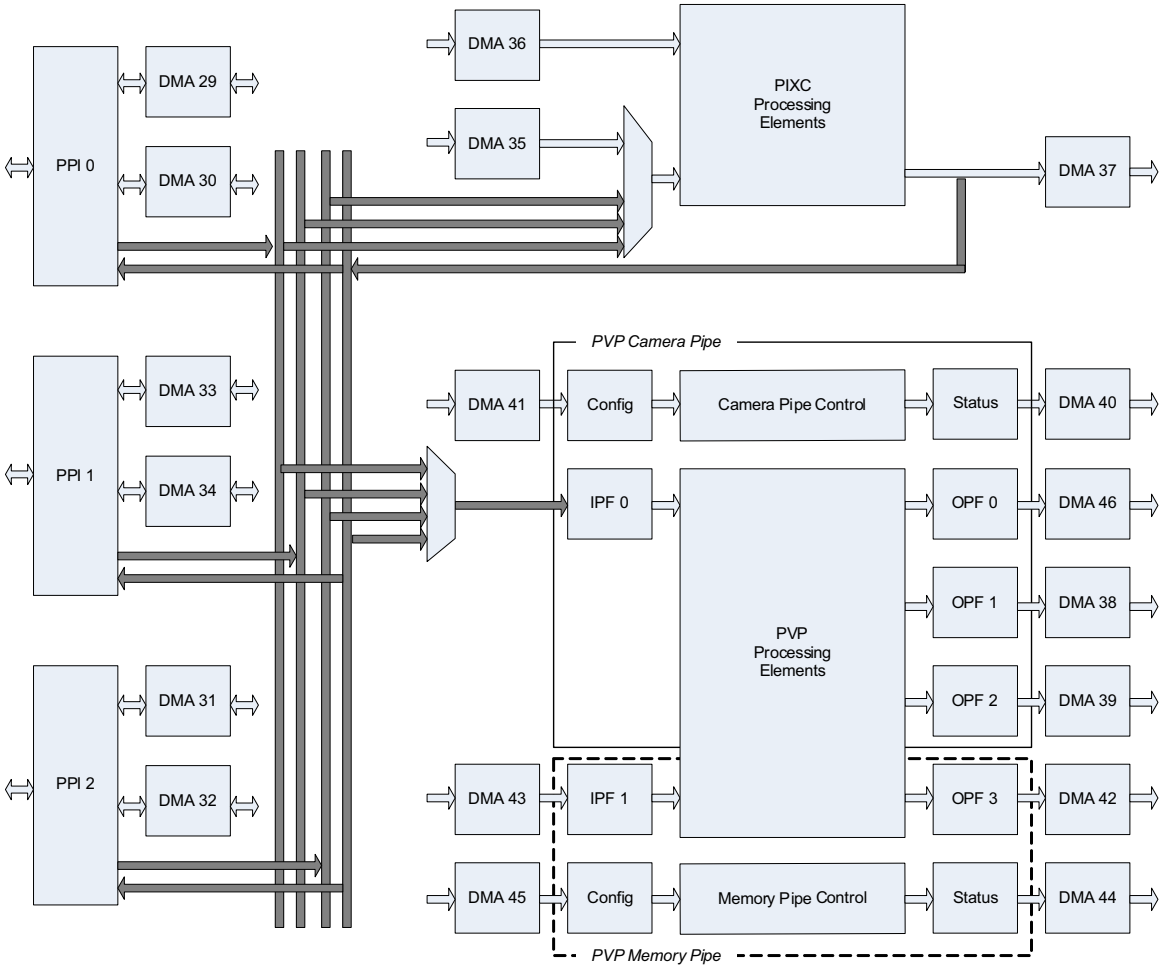


Figure 29-1: VID Block Diagram

VID Architectural Concepts

Traditionally, the three EPPIs operate in DMA mode and as such mainly transport data between off-chip and on-chip L1 and/or L2 SRAM or external L3 memories. Each EPPI interface contains two DMA channels. The primary channel is used for general data or for the luminance component only. The secondary channel can optionally be used for chrominance.

VID Status and Error Signals

The video subsystem does not have dedicated status or error interrupts. However, the VID signals errors to the associated EPPI peripheral when stall or overflow conditions occur. While the PVP has been designed to never stall on the video interconnect, the PIXC may do so. This is reported by the `EPPI_STAT`.

PXPERR flag. If the PIXC and the PVP are configured to receive data from the same EPPI input, the error flag only signals the stalling condition of the PIXC and the PVP continues to get new data.

VID Programming Model

The video subsystem is typically configured in static or pseudo-static fashion. There is only one MMR, the VID_CONN register, which needs to be written before the EPPI, PIXC and PVP can be enabled. When programs must alter the VID_CONN register at run time, it is important that all impacted video blocks are disabled.

The VID_CONN.PIXCOIN bit field controls whether the PIXC receives input from DMA or from a designated EPPI port. Similarly, the VID_CONN.PVPOIN bit field determines the data source of the PVP. When altering the VID_CONN.PIXCOIN bit field, the PIXC and both the former and the newly selected EPPIs need to be disabled. However, the other PPI and the PVP may continue operation without being disturbed.

For EPPIs operating in transmit mode, the three PPI transmit connectivity bit fields (VID_CONN.PPIOTX through VID_CONN.PPI2TX) control whether the respective PPI module receives data from DMA channel or from the PIXC. Only out of the PPIs or the PVP or the DMA can receive data from PIXC at a time.

For EPPIs operating in receive mode, the PPI broadcast bits (VID_CONN.PPIOBCAST through VID_CONN.PPI2BCAST) enable broadcasting of the main (luminance) input data. The bits only need to be set (=1) if received data is broadcast to both the video interconnect and DMA system. If not set (=0), the VID_CONN.PIXCOIN and VID_CONN.PVPOIN bits determine whether data goes to either the video interconnect or to the DMA system. Broadcasting only takes place if at least one of VID_CONN.PIXCOIN or VID_CONN.PVPOIN bit fields instruct the modules to consume data from the respective EPPI.

Care must be taken to ensure that the configuration of the EPPIs, PIXC and PVP works with the VID settings in a meaningful way. Also, if significant mode switching takes place on one module, the other modules are temporarily disabled.

VID Performance

The maximum pixel clock frequency is 83 MHz. Assuming each clock contains a valid pixel, the maximum pixel rate that can be processed by the PVP camer pipe is 83M Pixels/s. The same limitation also applies to the PVP's memory pipe. Unless the base clock (*SCLK*) is lower than 83 MHz, memory pipe throughput is *SCLK*/2. The PIXC can operate at full *SCLK* speed in memory-to-memory mode.

The bandwidth consumed by memory-to-memory operations of the PVP and the PIXC can be controlled via the bandwidth control functionality of the respective DMA channels. Similarly, the consumed bandwidth can be monitored by the bandwidth monitor functionality of the DMA channels.

ADSP-BF60x VID Register Descriptions

Video Subsystem Registers (VID) contains the following registers.

Table 29-2: ADSP-BF60x VID Register List

Name	Description
VID_CONN	Video Subsystem Connect Register

Video Subsystem Connect Register

The VID_CONN selects connections for the PVP and PPI modules, configuring connections to use these peripherals for video operations. Also, the VID_CONN selects broadcast or non-broadcast mode for handling the receive Y channel data. Note that these connections are static and have to be configured before enabling the PPI, PIXC, and PVP blocks.

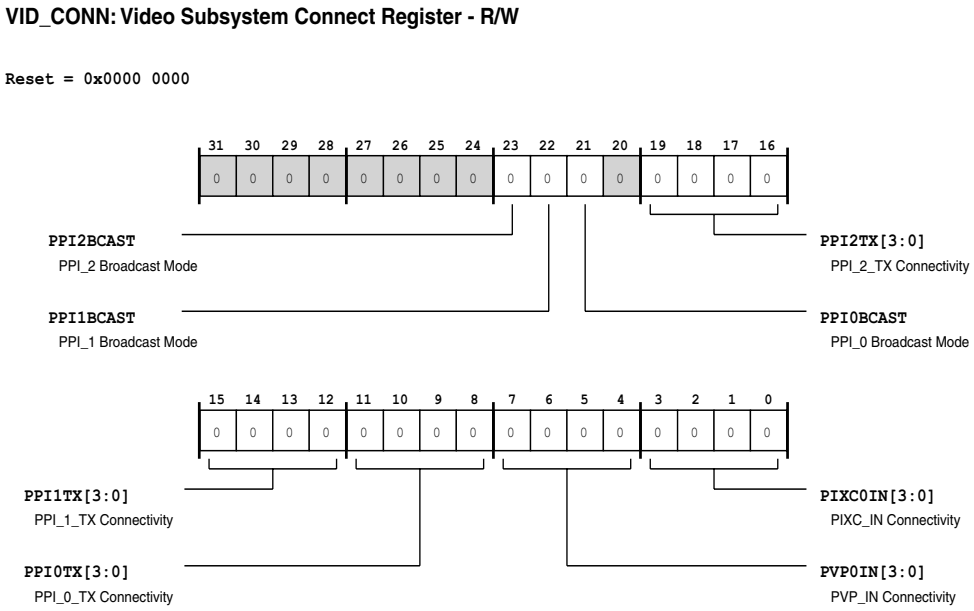


Figure 29-2: VID_CONN Register Diagram

Table 29-3: VID_CONN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23 (R/W)	PPI2BCAST	PPI_2 Broadcast Mode. The VID_CONN.PPI2BCAST selects broadcast or non-broadcast mode for receive Y channel data. For more information, see the VID_CONN.PPI0BCAST bit description.	
		0	Non-broadcast mode
		1	Broadcast mode

Table 29-3: VID_CONN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/W)	PPI1BCAST	PPI_1 Broadcast Mode. The VID_CONN.PPI1BCAST selects broadcast or non-broadcast mode for receive Y channel data. For more information, see the VID_CONN.PPI0BCAST bit description.	
		0	Non-broadcast mode
		1	Broadcast mode
21 (R/W)	PPI0BCAST	PPI_0 Broadcast Mode. The VID_CONN.PPI0BCAST selects broadcast or non-broadcast mode for receive Y channel data. This bit is valid only when PPI_0 is in receive mode, and either VID_CONN.PIXCOIN = 0x1 or VID_CONN.PVPOIN = 0x1. In broadcast mode, the receive Y channel data is sent to both the PxP interface and the PPI_0 DMA channel 0. In non-broadcast mode, the receive Y channel data is sent to either the PxP interface or the PPI_0 DMA channel based on the VID_CONN.PIXCOIN and VID_CONN.PVPOIN bits programming.	
		0	Non-broadcast mode
		1	Broadcast mode
19:16 (R/W)	PPI2TX	PPI_2_TX Connectivity. The VID_CONN.PPI2TX selects connection options for PPI_2 module output. These bits are ignored when PPI_2 is not in transmit mode.	
		0	PPI_2_TX to PPI_DMA PPI_2_TX is connected to the PPI DMA engine (PPI_0 DMA Channel 0)
		1	PPI_2_TX to PIXC PPI_2_TX is connected to PIXC output via PxP
15:12 (R/W)	PPI1TX	PPI_1_TX Connectivity. The VID_CONN.PPI1TX selects connection options for PPI_1 module output. These bits are ignored when PPI_1 is not in transmit mode.	
		0	PPI_1_TX to PPI_DMA PPI_1_TX is connected to the PPI DMA engine (PPI_1 DMA Channel 0)
		1	PPI_1_TX to PIXC PPI_1_TX is connected to PIXC output via PxP

Table 29-3: VID_CONN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11:8 (R/W)	PPI0TX	PPI_0_TX Connectivity. The VID_CONN.PPIOTX selects connection options for PPI_0 module output. These bits are ignored when PPI_0 is not in transmit mode.	
		0	PPI_0_TX to PPI_DMA PPI_0_TX is connected to the PPI DMA engine (PPI_0 DMA Channel 0)
		1	PPI_0_TX to PIXC PPI_0_TX is connected to PIXC output via PxP
7:4 (R/W)	PVP0IN	PVP_IN Connectivity. The VID_CONN.PVPOIN selects connection options for PVP module input.	
		0	PVP_IN to PVP_DMA PVP data input is connected to the PVP DMA engine
		1	PVP_IN to PPI_0_RX PVP input is connected to PPI_0_RX via PxP (valid only when PPI_0 is in receive mode)
		2	PVP_IN to PPI_1_RX PVP input is connected to PPI_1_RX via PxP (valid only when PPI_1 is in receive mode)
		3	PVP_IN to PPI_2_RX PVP input is connected to PPI_2_RX via PxP (valid only when PPI_2 is in receive mode)
		4	PVP_IN to PIXC_OUT PVP input is connected to PIXC output via PxP

Table 29-3: VID_CONN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/W)	PIXC0IN	PIXC_IN Connectivity. The VID_CONN.PIXCOIN selects connection options for PIXC module input.	
		0	PIXC_IN to PIXC_DMA PIXC image input is connected to the PIXC DMA IFIFO Channel (PIXC DMA channel 0)
		1	PIXC_IN to PPI_0_RX PIXC image input is connected to PPI_0_RX via PxP (valid only when PPI_0 is in receive mode)
		2	PIXC_IN to PPI_1_RX PIXC image input is connected to PPI_1_RX via PxP (valid only when PPI_1 is in receive mode)
		3	PIXC_IN to PPI_2_RX PIXC image input is connected to PPI_2_RX via PxP (valid only when PPI_2 is in receive mode)

30 Pipelined Vision Processor (PVP)

The Pipelined Vision Processor (PVP) provides a set of 12 high-performance signal processing blocks that can be flexibly combined to form streaming data processing pipes. These blocks are optimized for tasks typical of video and image processing, analytics (for example advanced driver assistance systems), robotics, and 2-dimensional vector applications. The PVP works in conjunction with the processor core(s). It is optimized for convolution and wavelet-based object detection, classification, tracking, and verification algorithms. The PVP bundles a set of processing blocks required for high-speed 2-dimensional digital signal processing.

Most blocks are optimized for 2-dimensional video analytics operations and can be re-purposed for general-purpose operations. Other blocks have general-purpose functionality and might be used in a variety of applications outside of video processing.

PVP Features

The PVP contains a number of highly configurable blocks that provide a broad set of pixel processing features:

- Variety of 1st-derivative and 2nd-derivative edge detection and classification methods
- Quad 5x5 convolution kernels with 8300 MMACS total
- Integral of pixels, variances, magnitudes and gradients
- Thresholds and histograms
- General-purpose 32-bit divider, multiplier, adder, and accumulator
- Up to 1280 x 960 pixel progressive video
- Up to 83M Pixels/second on each of up to four pipes
- Bandwidth-saving pre-processor architecture with data broadcasting to three “camera pipes”
- Memory-to-memory co-processing with autonomous sequencing of job lists
- Concurrent pre- and co-processing
- Autonomous update of coefficients at frame boundaries

Many of these features are influenced not just by individual PVP block features, but are also influenced by the configurable pipeline interconnections between blocks. For more information on PVP blocks and connections, see the *PVP Block Diagram* or see *Configuring Pipe Structure*.

PVP Functional Description

The PVP consists of signal processing blocks, input formatters, and output formatters. In combination and for simplicity, these elements are referred to as PVP blocks.

The following sections describe the PVP blocks within the PVP. The descriptions provide details on the PVP block operations and interactions with each other and the processor core(s).

ADSP-BF60x PVP Register List

The pipelined vision processor PVP implements the signal and image processing algorithms that are required for pre-processing of video frames in advanced driver assistance systems (ADAS) applications. A number of processing blocks contribute to PVP operations. These include:

- Convolution/Down-scaling engine (CNV)
- Pixel Magnitude and Angle computation unit (PMA)
- Threshold, Histogram and Compression engine (THC)
- Arithmetic Computation Unit (ACU)

- Pixel Edge Classifier (PEC)
- Integral Image computation (IIM)
- Up-Down Scaler (UDS)

In addition to these processing blocks, the PVP also has a number of blocks that permit interfacing to other modules (DMA, EPPI) and provide control for PVP operations:

- Input data formatter (IPF)
- Output data formatter (OPF)
- Image pipe controller (IPC)

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

Table 30-1: ADSP-BF60x PVP Register List

Name	Description
PVP_CTL	Control
PVP_IMSKn	Interrupt Mask n
PVP_STAT	Status
PVP_ILAT	Interrupt Latch Status n
PVP_IREQn	Interrupt Request n
PVP_OPFn_CFG	OPFn (Camera Pipe) Configuration
PVP_OPFn_CTL	OPFn (Camera Pipe) Control
PVP_OPF3_CFG	OPF3 (Memory Pipe) Configuration
PVP_OPF3_CTL	OPF3 (Memory Pipe) Control
PVP_PEC_CFG	PEC Configuration
PVP_PEC_CTL	PEC Control
PVP_PEC_D1TH0	PEC Lower Hysteresis Threshold
PVP_PEC_D1TH1	PEC Upper Hysteresis Threshold
PVP_PEC_D2TH0	PEC Weak Zero Crossing Threshold

Table 30-1: ADSP-BF60x PVP Register List (Continued)

Name	Description
PVP_PEC_D2TH1	PEC Strong Zero Crossing Threshold
PVP_IIMn_CFG	IIMn Configuration
PVP_IIMn_CTL	IIMn Control
PVP_IIMn_SCALE	IIMn Scaling Values
PVP_IIMn_SOVF_STAT	IIMn Signed Overflow Status
PVP_IIMn_UOVF_STAT	IIMn Unsigned Overflow Status
PVP_ACU_CFG	ACU Configuration
PVP_ACU_CTL	ACU Control
PVP_ACU_OFFSET	ACU SUM Constant
PVP_ACU_FACTOR	ACU PROD Constant
PVP_ACU_SHIFT	ACU Shift Constant
PVP_ACU_MIN	ACU Lower Sat Threshold Min
PVP_ACU_MAX	ACU Upper Sat Threshold Max
PVP_UDS_CFG	UDS Configuration
PVP_UDS_CTL	UDS Control
PVP_UDS_OHCNT	UDS Output HCNT
PVP_UDS_OVCNT	UDS Output VCNT
PVP_UDS_HAVG	UDS HAVG
PVP_UDS_VAVG	UDS VAVG
PVP_IPF0_CFG	IPF0 (Camera Pipe) Configuration
PVP_IPFn_PIPECTL	IPFn (Camera/Memory Pipe) Pipe Control
PVP_IPFn_CTL	IPFn (Camera/Memory Pipe) Control

Table 30-1: ADSP-BF60x PVP Register List (Continued)

Name	Description
PVP_IPFn_TAG	IPFn (Camera/Memory Pipe) TAG Value
PVP_IPFn_FCNT	IPFn (Camera/Memory Pipe) Frame Count
PVP_IPFn_HCNT	IPFn (Camera/Memory Pipe) Horizontal Count
PVP_IPFn_VCNT	IPFn (Camera/Memory Pipe) Vertical Count
PVP_IPF0_HPOS	IPF0 (Camera Pipe) Horizontal Position
PVP_IPF0_VPOS	IPF0 (Camera Pipe) Vertical Position
PVP_IPFn_TAG_STAT	IPFn (Camera/Memory Pipe) TAG Status
PVP_IPF1_CFG	IPF1 (Memory Pipe) Configuration
PVP_CNVn_CFG	CNVn Configuration
PVP_CNVn_CTL	CNVn Control
PVP_CNVn_C00C01	CNVn Coefficients 0,0 and 0,1
PVP_CNVn_C02C03	CNVn Coefficients 0,2 and 0,3
PVP_CNVn_C04	CNVn Coefficient 0,4
PVP_CNVn_C10C11	CNVn Coefficients 1,0 and 1,1
PVP_CNVn_C12C13	CNVn Coefficients 1,2 and 1,3
PVP_CNVn_C14	CNVn Coefficient 1,4
PVP_CNVn_C20C21	CNVn Coefficients 2,0 and 2,1
PVP_CNVn_C22C23	CNVn Coefficients 2,2 and 2,3
PVP_CNVn_C24	CNVn Coefficient 2,4
PVP_CNVn_C30C31	CNVn Coefficients 3,0 and 3,1
PVP_CNVn_C32C33	CNVn Coefficients 3,2 and 3,3
PVP_CNVn_C34	CNVn Coefficient 3,4

Table 30-1: ADSP-BF60x PVP Register List (Continued)

Name	Description
PVP_CNVn_C40C41	CNVn Coefficients 4,0 and 4,1
PVP_CNVn_C42C43	CNVn Coefficients 4,2 and 4,3
PVP_CNVn_C44	CNVn Coefficient 4,4
PVP_CNVn_SCALE	CNVn Scaling Factor
PVP_THCn_CFG	THCn Configuration
PVP_THCn_CTL	THCn Control
PVP_THCn_HFCNT	THCn Histogram Frame Count
PVP_THCn_RMAXREP	THCn Max RLE Reports
PVP_THCn_CMINVAL	THCn Min Clip Value
PVP_THCn_CMINTH	THCn Clip Min Threshold
PVP_THCn_CMAXTH	THCn Clip Max Threshold
PVP_THCn_CMAXVAL	THCn Max Clip Value
PVP_THCn_TH0	THCn Threshold Value 0
PVP_THCn_TH1	THCn Threshold Value 1
PVP_THCn_TH2	THCn Threshold Value 2
PVP_THCn_TH3	THCn Threshold Value 3
PVP_THCn_TH4	THCn Threshold Value 4
PVP_THCn_TH5	THCn Threshold Value 5
PVP_THCn_TH6	THCn Threshold Value 6
PVP_THCn_TH7	THCn Threshold Value 7
PVP_THCn_TH8	THCn Threshold Value 8
PVP_THCn_TH9	THCn Threshold Value 9

Table 30-1: ADSP-BF60x PVP Register List (Continued)

Name	Description
PVP_THCn_TH10	THCn Threshold Value 10
PVP_THCn_TH11	THCn Threshold Value 11
PVP_THCn_TH12	THCn Threshold Value 12
PVP_THCn_TH13	THCn Threshold Value 13
PVP_THCn_TH14	THCn Threshold Value 14
PVP_THCn_TH15	THCn Threshold Value 15
PVP_THCn_HHPOS	THCn Histogram Horizontal Position
PVP_THCn_HVPOS	THCn Histogram Vertical Position
PVP_THCn_HHCNT	THCn Histogram Horizontal Count
PVP_THCn_HVCNT	THCn Histogram Vertical Count
PVP_THCn_RHPOS	THCn RLE Horizontal Position
PVP_THCn_RVPOS	THCn RLE Vertical Position
PVP_THCn_RHCNT	THCn RLE Horizontal Count
PVP_THCn_RVCNT	THCn RLE Vertical Count
PVP_THCn_HFCNT_STAT	THCn Histogram Frame Count Status
PVP_THCn_HCNT0_STAT	THCn Histogram Counter Value 0
PVP_THCn_HCNT1_STAT	THCn Histogram Counter Value 1
PVP_THCn_HCNT2_STAT	THCn Histogram Counter Value 2
PVP_THCn_HCNT3_STAT	THCn Histogram Counter Value 3
PVP_THCn_HCNT4_STAT	THCn Histogram Counter Value 4
PVP_THCn_HCNT5_STAT	THCn Histogram Counter Value 5
PVP_THCn_HCNT6_STAT	THCn Histogram Counter Value 6

Table 30-1: ADSP-BF60x PVP Register List (Continued)

Name	Description
PVP_THCn_HCNT7_STAT	THCn Histogram Counter Value 7
PVP_THCn_HCNT8_STAT	THCn Histogram Counter Value 8
PVP_THCn_HCNT9_STAT	THCn Histogram Counter Value 9
PVP_THCn_HCNT10_STAT	THCn Histogram Counter Value 10
PVP_THCn_HCNT11_STAT	THCn Histogram Counter Value 11
PVP_THCn_HCNT12_STAT	THCn Histogram Counter Value 12
PVP_THCn_HCNT13_STAT	THCn Histogram Counter Value 13
PVP_THCn_HCNT14_STAT	THCn Histogram Counter Value 14
PVP_THCn_HCNT15_STAT	THCn Histogram Counter Value 15
PVP_THCn_RREP_STAT	THCn Number of RLE Reports
PVP_PMA_CFG	PMA Configuration

ADSP-BF60x PVP Interrupt List

Table 30-2: ADSP-BF60x PVP Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
PVP0 Camera Pipe Data Out B DMA Channel	111	38	LEVEL
PVP0 Camera Pipe Data Out C DMA Channel	112	39	LEVEL
PVP0 Camera Pipe Status Out DMA Channel	113	40	LEVEL
PVP0 Camera Pipe Control In DMA Channel	114	41	LEVEL
PVP0 Status 0	115		LEVEL
PVP0 Memory Pipe Data Out DMA Channel	116	42	LEVEL

Table 30-2: ADSP-BF60x PVP Interrupt List Interrupt List (Continued)

Description	Interrupt ID	DMA Channel	Sensitivity
PVP0 Memory Pipe Data In DMA Channel	117	43	LEVEL
PVP0 Memory Pipe Status Out DMA Channel	118	44	LEVEL
PVP0 Memory Pipe Control In DMA Channel	119	45	LEVEL
PVP0 Camera Pipe Data Out A DMA Channel	120	46	LEVEL
PVP0 Status 1	121		LEVEL

ADSP-BF60x PVP Trigger List

Table 30-3: ADSP-BF60x PVP Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
PVP0 Camera Pipe Data Out B DMA Channel	60	PULSE/EDGE
PVP0 Camera Pipe Data Out C DMA Channel	61	PULSE/EDGE
PVP0 Camera Pipe Status Out DMA Channel	62	PULSE/EDGE
PVP0 Camera Pipe Control In DMA Channel	63	PULSE/EDGE
PVP0 Memory Pipe Data Out DMA Channel	64	PULSE/EDGE
PVP0 Memory Pipe Data In DMA Channel	65	PULSE/EDGE
PVP0 Memory Pipe Status Out DMA Channel	66	PULSE/EDGE
PVP0 Memory Pipe Control In DMA Channel	67	PULSE/EDGE
PVP0 Camera Pipe Data Out A DMA Channel	68	PULSE/EDGE
PVP0 Status 0	78	LEVEL
PVP0 Status 1	79	LEVEL

Table 30-4: ADSP-BF60x PVP Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
PVP0 Camera Pipe Data Out B DMA Channel	60	
PVP0 Camera Pipe Data Out C DMA Channel	61	
PVP0 Camera Pipe Status Out DMA Channel	62	
PVP0 Camera Pipe Control In DMA Channel	63	
PVP0 Memory Pipe Data Out DMA Channel	64	
PVP0 Memory Pipe Data In DMA Channel	65	
PVP0 Memory Pipe Status Out DMA Channel	66	
PVP0 Memory Pipe Control In DMA Channel	67	
PVP0 Camera Pipe Data Out A DMA Channel	68	

ADSP-BF60x PVP DMA List

Table 30-5: ADSP-BF60x PVP DMA List DMA Channel List

Description	DMA Channel
PVP0 Camera Pipe Data Out B DMA Channel	DMA38
PVP0 Camera Pipe Data Out C DMA Channel	DMA39
PVP0 Camera Pipe Status Out DMA Channel	DMA40
PVP0 Camera Pipe Control In DMA Channel	DMA41
PVP0 Memory Pipe Data Out DMA Channel	DMA42
PVP0 Memory Pipe Data In DMA Channel	DMA43
PVP0 Memory Pipe Status Out DMA Channel	DMA44
PVP0 Memory Pipe Control In DMA Channel	DMA45
PVP0 Camera Pipe Data Out A DMA Channel	DMA46

PVP Block Diagram

The following figure shows the PVP block diagram. For more information on parts of this diagram, see the corresponding section of the *PVP Functional Description* or see *PVP Architectural Concepts*.

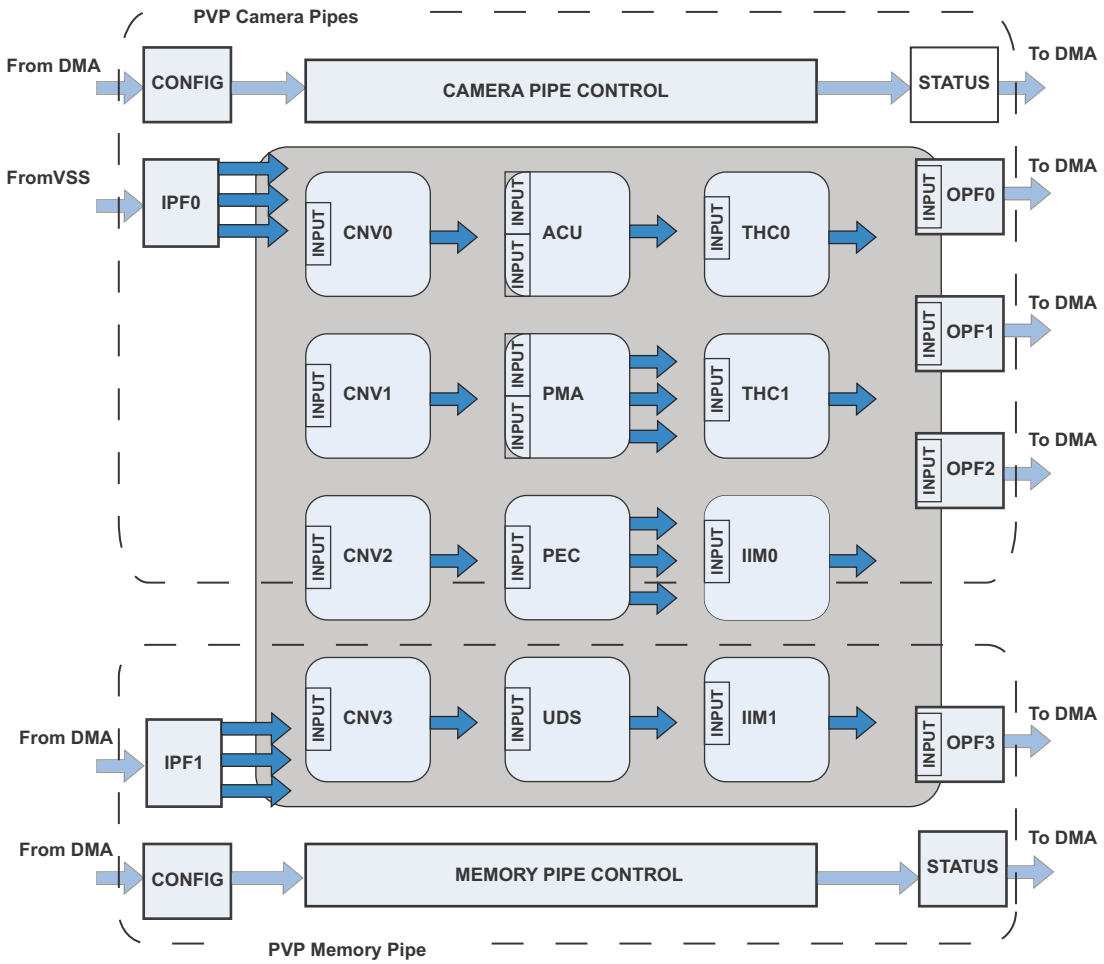


Figure 30-1: PVP High-Level Block Diagram

PVP Definitions

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

Cartesian to Polar Magnitude and Polar Angle Conversion

This conversion processes two 16-bit signed inputs as coordinates in the Cartesian form (x, y) and converts them into Polar form (Magnitude, Angle φ).

Convolution

Convolution is a mathematical operation on two signals producing a third signal which is typically viewed as a modified version of one of the original signals.

Integral Image Processing

The integral image blocks calculate a 2-dimensional integral over the input frame and outputs the summed area table (SAT). Alternatively, these blocks can generate a rotated SAT (RSAT) or can integrate in horizontal dimension only (integral row mode).

Pixel Edge Classification Block

The pixel edge classifier operates in either 1st derivative mode (PEC-1) or 2nd derivative mode (PEC-2). In PEC-1 mode, the block expects the magnitude on the lower 16-bits and the angle information on the upper 16-bits. In PEC-2 mode, the block generates 8-connected chain codes for contour tracing, using output from second derivative edge detection methods such as Difference of Gaussian (DoG) and Laplacian of Gaussian (LoG).

Threshold-Histogram-Compression Block

The threshold-histogram-compression blocks implement a collection of statistical and range reduction signal processing functions. Using the PVP's pipeline interconnection options, output from a variety of other PVP blocks may serve as input to the threshold-histogram-compression blocks.

Up-Down Scaling Block

The up-down-scaling block expects 16-bit or 32-bit unsigned input data and drives 32-bit unsigned output data. When an anti-aliasing or an averaging filter is enabled, the input must be 16 bits and correspondingly the output is 16 bits presented in the lower 16 bits of the 32-bit output.

Input Formatters (IPFn)

The PVP features two input formatters. Input formatter 0 (IPF0) serves the camera pipe. Data for IPF0 comes to the PVP from the video subsystem. Input formatter 1 (IPF1) serves the memory pipe. Data for IPF1 comes to the PVP through DMA. The input formatters are the masters of the data flow for the entire pipeline that they serve. Data received by IPF0 can be broadcast and is sent to the DMA system through the camera pipe output formatters (OPF0, OPF1, and OPF2). Data received by IPF1 is sent to the DMA system through the memory pipe output formatter (OPF3). The input formatters are responsible for the following tasks:

- Formatting input data
- Extracting color or luminance components

- Windowing
- Separating odd and even pixels
- Separating Bayer and RCCC color components
- Counting frames
- Reporting/controlling status

The following tables provide details for input formatter (IPF0 and IPF1) inputs and outputs. The IPF0 data source is fixed to video subsystem, and the IPF1 data source is fixed to DMA. The data format is listed as s16 for 16-bit signed data, u16 for 16-bit unsigned data, s32 for 32-bit signed data, and u32 for 32-bit unsigned data. For a graphical overview of all PVP block interconnections, see *Configuring Pipe Structure*.

Table 30-6: IPF0 Block (Camera Pipe) Connectivity

IPF0 Block I/O	Data Format	PVP Block Connect Selections	Port Connect
Input Port 0	s16, u16, s32, u32	Video Sub-System	0
Input Port 1	n/a	n/a	n/a
Output Port 0	s32 (Full Frame)	CNV0, CNV1, CNV2, CNV3, PMA, ACU, PEC, THC0, THC1, IIM0, IIM1. OPF0, OPF1, OPF2	n/a
Output Port 1	s32 (Odd/Window)		
Output Port 2	s32 (Even)		

Table 30-7: IPF1 (Memory Pipe) Block Connectivity

IPF1 Block I/O	Data Format	PVP Block Connect	Port Connect
Input Port 0	s16, u16, s32, u32	DMA	0
Input Port 1	n/a	n/a	n/a
Output Port 0	s32 (Full Frame)	CNV0, CNV1, CNV2, CNV3, PMA, ACU, PEC, THC0, THC1, IIM0, IIM1, OPF3, UDS	n/a
Output Port 1	s32 (Odd)		
Output Port 2	s32 (Even)		

Both input formatters feature three 32-bit outputs, which can be individually enabled by the respective `PVP_IPFn_CTL.OPORTOEN`, `PVP_IPFn_CTL.OPORT1EN`, and `PVP_IPFn_CTL.OPORT2EN` bits. All IPFn outputs can operate in parallel. If enabled, IPFn output port 0 outputs all data words.

Output Ports that do not source a complete pipeline must be kept disabled. A pipeline is considered as complete if it is terminated by an OPFn block or by a THCn block with histogram output.

Input Formatters with Odd/Even Outputs

IPFn output ports 1 and 2 can be set to partner such that all odd-indexed data words output from port 1 and all even-indexed data words output from port 2. In this mode, the even-indexed data words are delayed by one clock cycle where a pair of even/odd words leaves the IPFn simultaneously. The following figure shows an example memory pipe operation in which IPF1 uses this feature.

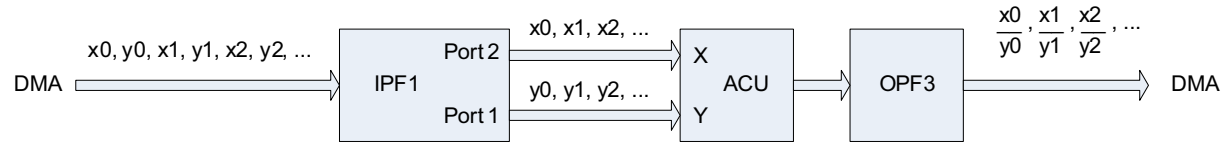


Figure 30-2: Separation of Odd and Even Data Words

In this memory pipe example, the arithmetic control unit (ACU) is configured to perform division operations in memory pipe mode. The ACU expects the numerator on its X input and expects the denominator on its Y input. The memory pipe receives all data in a single DMA channel, which may operate in 2-dimensional mode. The DMA may provide a series of interleaved numerator/denominator pairs. Using the odd/even feature, IPF1 splits the data words as required. The resulting quotients are sent to DMA through the memory pipe output formatter (OPF3).

Input Formatters with Windowing

IPF0 output port 1 (serving the camera pipe) can be set to output only selected pixels, which reside within window coordinates as specified by the `PVP_IPF0_HPOS`, `PVP_IPF0_VPOS`, `PVP_IPFn_HCNT`, and `PVP_IPFn_VCNT` registers. The figure shows how these coordinates work together to identify the pixel window with a frame.

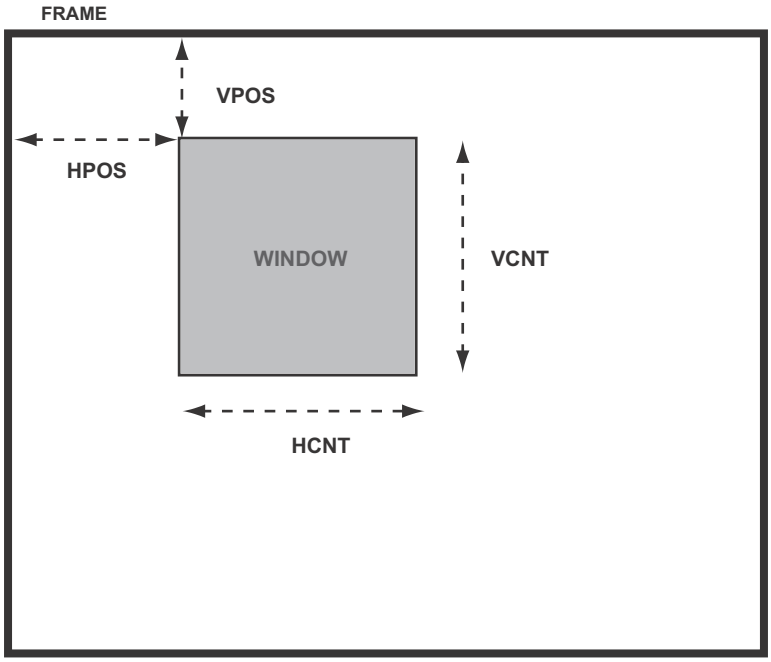


Figure 30-3: Definition of Port 1 Window Coordinates

IPF1 (serving the memory pipe) uses a different technique to support window. For windowing on IPF1, the PVP_IPFn_HCNT and PVP_IPFn_VCNT registers are not used. Rather, the window is described by DMA registers. The PVP_IPFn_HCNT and PVP_IPFn_VCNT registers are required to match the XCNT, YCNT, and MSIZE settings of the input data DMA. The PVP applies the window coordinates after color extraction. So, be sure to consider the color format selected by the PVP_IPFn_CTL.CFRMT bit field when using IPF1 for windowing operations.

Input Formatters Receiving Packed Data

The PVP has 32-bit wide data input. IPF0 receives data from the video subsystem (PPIs and PIXC), and IPF1 receives 32-bit data words from the DMA. This capacity for input data is wider than typical video data, because video pixels are typically represented by 16-bit data entities. To operate the PVP most efficiently, the data source can pack pairs of 16-bit data words into 32-bit data input to the PVP. The DMA controller enables 32-bit packing by setting the DMA_CFG.PSIZE(PPIs) and the pixel compositor (PIXC) send packed color formats to the PVP. Or, the PPIs enable packing by setting the EPPI_CTL.PACKENentities when the PVP_IPFn_CTL.UNPACK

The number of significant bits varies with the type of data. Monochrome data can have 8, 16, 24, or 32 significant bits. Note that, if the PPI receives 10-, 12- or 14-bit data it performs zero/sign-extension on the data to 16 bits. In special color formats, the IPFn blocks also extend data from 5 or 6 significant bits.

Dimensions of a frame supplied to the IPFn blocks generally must be a multiple of four. Further restrictions may apply by specific PVP blocks, especially if the odd/even ports are used or complex color space format is selected.

Input Formatters Receiving Unsigned Data

Typically, the IPFn blocks receive video data that is 8-, 10-, 12-, or 14-bits wide, which is zero extended to 16-bit values. This typical pixel data can always be positive values, whether the data is processed by signed- or unsigned-computation engines. If the IPFn blocks receive data that is 16 bits wide (without extension) or that is 32 bits wide, correct operation requires that the PVP blocks process the signed- or unsigned-data input with operations that are appropriate for the data type.

Data processing related conflicts can occur when:

- The IPFn receives data is unsigned 16- or 32-bit data, *AND*
- The IPFn forwards the unsigned data to PVP blocks that can only operate on signed data.

The affected PVP blocks are:

- CNVn blocks and PMA block, which require signed 16-bit data
- ACU block, which requires signed 32-bit data

To avoid these conflicts, use the `PVP_IPFn_CTL.QFRMT` bit to instruct the IPFn to right shift the incoming data by one bit position and to zero fill the most significant bit. This operation directs the PVP to interpret the input as a positive value for the signed computation engines. The output formatter (OPFn) blocks provide the counterpart functionality, which shifts the result back to the left by one bit position. This approach avoids conflicts due to signed bits (at the cost of losing one bit of input accuracy).

Input Formatters with Color Extraction

The PVP blocks process data in monochrome format. Often, the IPFn blocks' data inputs have to probe color data streams. To support this data, the IPFn blocks include features for extracting luminance or chromatic values from composite colored data streams. These features include:

- Extraction of Y (luma) values out of YCrCb streams
- Extraction of G (green) values out of RGB streams
- Extraction of G (green) values out of Bayer streams
- Extraction of R (red) values out of Bayer streams

The `PVP_IPFn_CTL.CFRMT` bit field defines the color format being used by the incoming data. The monochrome and color video formats table provides the overview of transfers operations, where each cell shows two consecutive 32-bit words that follow the conventions shown below. This table refers to data sent by the EPPI or PIXC in case of IPF0 and data stored in memory (and fetched by DMA) for IPF1.

1st word MS byte	1st word byte 2	1st word byte 1	1st word LS byte
2nd word MS byte	2nd word byte 2	2nd word byte 1	2nd word LS byte

NOTE: The video formats table uses YCrCb terminology for color formats, rather than the YUV terminology used in the PPI and PIXC peripheral descriptions. Items marked with an asterisk (*) indicate that multiple options are commonly used in industry standard, but there is no difference for the PVP as the Cr components and the Cb components are discarded.

The IPFn blocks process all data words for the monochrome input format. For YCrCb formats, the IPFn blocks discard the Cr and Cb color components, and only process the luminance values Y. With RGB formats, the PVP_IPFn_CTL.EXTRED bit is used. If this bit is cleared, the IPFn blocks extract the green components. If this bit is set, the IPFn blocks extract the red components.

Table 30-8: Supported Monochrome and Color Video Formats

PVP_IPFn_CTL.CFRMT		PVP_IPFn_CTL.UNPACK = 0	PVP_IPFn_CTL.UNPACK = 1
0x1B	32-bit mono	Y0 Y1	N/A
0x1A	24-bit mono	0 Y0 0 Y1	N/A
0x19	16-bit mono	0 Y0 0 Y1	Y1 Y0 Y3 Y2
0x18	8-bit mono	Y0 Y1	Y3 Y2 Y1 Y0 Y7 Y6 Y5 Y4

Table 30-8: Supported Monochrome and Color Video Formats (Continued)

PVP_IPFn_CTL.CFRMT		PVP_IPFn_CTL.UNPACK = 0	PVP_IPFn_CTL.UNPACK = 1
0x14	16-bit YCrCb 422	0 Cb0 0 Y1	Y0 Cb0 Y1 Cr0
0x15	16-bit YCrCb 422	N/A	Cr0 Cb0 Y2 Y0
0x16	16-bit YCrCb 422	N/A	Y1 Y0 Cr0/Cb1* Cb0
0x10 0x13	8-bit YCrCb 422	0 0 0 Cb0 0 0 0 Y0	Y1 Cr0 Y0 Cb0 Y3 Cr1 Y2 Cb1
0x11	8-bit YCrCb 422	N/A	Cr1 Cb1 Cr0 Cb0 Y3 Y2 Y1 Y0
0x12	8-bit YCrCb 422	N/A	Y3 Y2 Y1 Y0 Cr1/ Cb1* Cb1/ Cr1* Cr0/ Cb0* Cb0/ Cr0*

Table 30-8: Supported Monochrome and Color Video Formats (Continued)

PVP_IPFn_CTL.CFRMT		PVP_IPFn_CTL.UNPACK = 0	PVP_IPFn_CTL.UNPACK = 1
0x04	16-bit RGB	0 R0 0 G0	G0 R0 R1 B0
0x00	8-bit RGB	0 0 0 R0 0 0 0 G0	R1 B0 G0 R0 G2 R2 B1 G1
0x01	8-bit RGB	B0 G0 R0 B1 G1 R1	R1 B0 G0 R0 G2 R2 B1 G1
0x02	16-bit RGB 565	0 RGB0 0 RGB1	RGB1 RGB0 RGB3 RGB2
0x03	24-bit RGB 666	RGB0 RGB1	RGB1 RGB0 RGB2 RGB1

In addition to the streaming formats shown in the monochrome and color video formats table, the IPFn blocks can also extract color components out of Bayer pattern matrices as shown in the following table.

Table 30-9: Supported Bayer Patterns

PVP_IPFn_CTL.CFRMT		Bayer Pattern Color Matrices
0x05	Bayer Type 1	B G B G B G B G G R G R G R G R B G B G B G B G G R G R G R G R B G B G B G B G G R G R G R G R B G B G B G B G G R G R G R G R
0x06	Bayer Type 2	G R G R G R G R B G B G B G B G G R G R G R G R B G B G B G B G G R G R G R G R B G B G B G B G G R G R G R G R B G B G B G B G

If the PVP_IPFn_CTL.EXTRED bit =0, the IPFn blocks extract the green components of the Bayer matrix. The output frame has half the horizontal resolution of the input frame. If the PVP_IPFn_CTL.EXTRED bit =1, the IPFn blocks extract the red component, dividing horizontal and vertical resolution.

In Bayer extraction mode, the input values can be 10, 12, or 16 bits wide. An 8-bit format is not supported. The PPI's EPPI_CTL.SWAPEN bit must be cleared in this mode.

The Bayer concepts can also apply to red-clear-clear-clear (RCCC) data. This is similar to Bayer format, except that the green and blue pixels have monochrome (clear filter) values. For extraction on RCCC data, programs have the following options:

- Extract the red pixels as in Bayer extraction mode.
- Extract one clear pixel by swapping type 1 versus type 2 and setting the PVP_IPFn_CTL.EXTRED bit.
- Extract the diagonal clear pixels that correspond to the green pixels in Bayer format.
- Use the odd/even mechanism and only process the clear columns.
- Ignore the fact that red pixels are special and low-pass them using convolution blocks.

Input Formatters with Color Separation

Unlike color extraction (which is performed at the IPFn input ports), the PVP performs color separation as an operation of the IPF0 output ports. Also, color separation only is available for IPF0 (camera pipe operation) and is *not*supported for IPF1 (memory pipe operation).

NOTE: Color separation operations require that the IPF0 input does not manipulate color components. Also, color seperation only is supported for monochrome `PVP_IPFn_CTL.CFRMT` options (0x18, 0x19, 0x1A, 0x1B).

Color separation grants concurrent access to multiple color components. It is enabled with the `PVP_IPFn_CTL.BFRMT0` bit. Color separation mode supports two Bayer types, as controlled with the `PVP_IPFn_CTL.BFRMT1` bit. PVP hardware does not differentiate between Bayer and red-clear-clear-clear (RCCC) formats. Data received from RCCC sources needs to be interpreted accordingly. (See the **Color Separation for Type 1 and 2** diagram.)

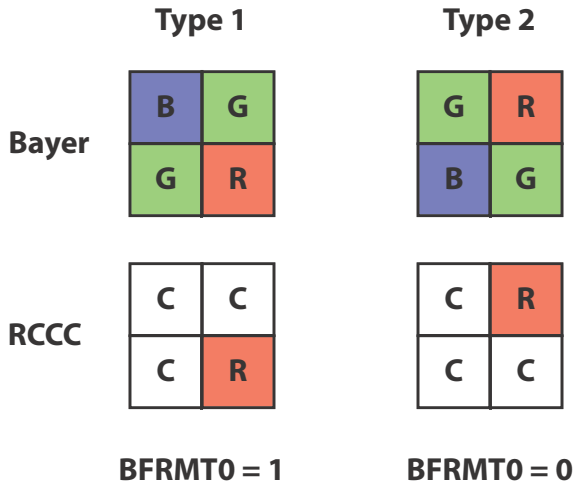


Figure 30-4: Color Separation for Type 1 and 2

In color separation mode, output port 2 outputs the red pixels, and output port 1 provides the green ones. The green pixel next to the red one always is used. This pixel is delayed by one pixel clock, so it aligns with the red pixel. The PVP processing elements with two inputs are allowed to do math operation on both color components. For example, the ACU can output the sum of the red pixel plus the green pixel. The other green pixel (the one next to the blue pixel) is suppressed.

Output port 0 may be configured to forward all input pixels (`PVP_IPFn_CTL.BFRMT1` =0) or may be configured to output only the blue pixels (`PVP_IPFn_CTL.BFRMT1` =1). Neither configuration aligns this output with green pixel output (port 1) and red pixel output (port 2), so the blue pixels and cannot be processed along with the red/green pixels by ACU operations.

As shown is the **Color Separation and Ports** diagram, Color Separation leads to reduced frame resolution. The resulting pixel rate is one fourth of the input data rate.

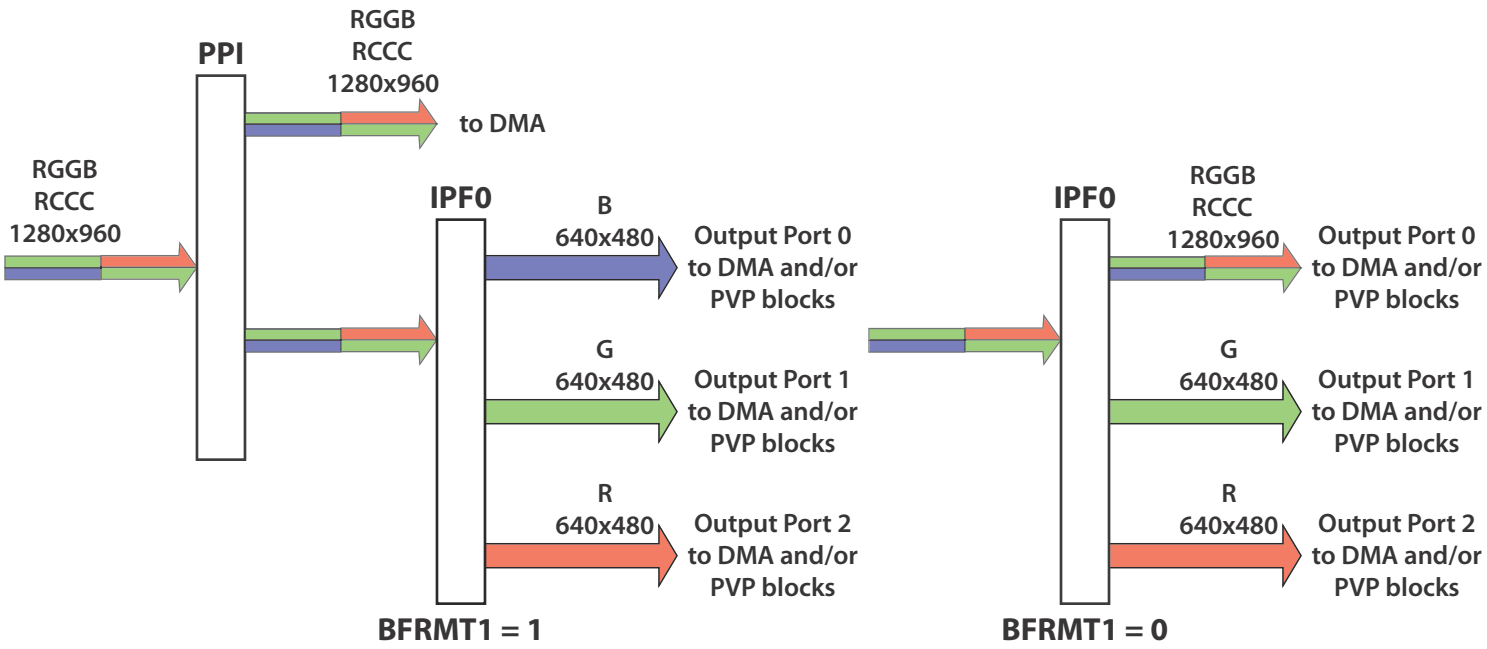


Figure 30-5: Color Separation and Ports

Input Formatters Using PPI and PVP

If IPF0 is receiving data from any of the PPIs in camera pipe mode, the settings of the two modules need to partner for reception of reasonable data formats. The following tables list the supported combinations of settings for the PPI and PVP.

NOTE: An "X" entry for a field value in the table indicates that the control bit field is not used for the described transfer.

Table 30-10: PPI and PVP Settings for RGB 8-Bit (YCbCr 4:4:4) with 8 Bits per PIXCLK

PPI/PVP	Control Bit Fields	Field Value (Description)	
PPI settings	EPPI_CTL.XFRTYPE	0x3 (Non-ITU656 Mode, GP Mode)	
	EPPI_CTL.DLEN	0x0 (8-bit data length)	
	EPPI_CTL.SPLTWRD	0x0 (PPI_DATA has DLEN-1 bits of Y or Cr or Cb)	
	EPPI_CTL.SPLTEO	0x0 (Do Not Split Samples)	
	EPPI_CTL.PACKEN	0x0 (Disable)	0x1 (Enable)
	EPPI_CTL.DMACFG	X	
	EPPI_CTL.SUBSPLTODD	X	

Table 30-10: PPI and PVP Settings for RGB 8-Bit (YCbCr 4:4:4) with 8 Bits per PIXCLK (Continued)

PPI/PVP	Control Bit Fields	Field Value (Description)	
PVP Settings	PVP_IPFn_CTL.CFRMT	0x00 (RGB 8-Bit)	
	PVP_IPFn_CTL.UNPACK	0x0 (No Unpacking)	0x1 (Unpack Data)
	Data Placement	RGBRGB..., not packed on PxP 0 0 0 R0 0 0 0 G0	RGBRGB..., packed on PxP R1 B0 G0 R0 G2 R2 B1 G1

Table 30-11: RGB 888 (YCbCr 4:4:4) with 24 Bits per PIXCLK

PPI/PVP	Setting	Value/Information	
PPI settings	EPPI_CTL.XFRTYPE	0x3 (Non-ITU656 Mode, GP Mode)	
	EPPI_CTL.DLEN	0x7 (24-bit data length)	
	EPPI_CTL.SPLTWRD	0x0 (PPI_DATA has DLEN-1 bits of Y or Cr or Cb)	
	EPPI_CTL.SPLTEO	0x0 (Do Not Split Samples)	
	EPPI_CTL.PACKEN	0x0 (Disable)	0x1 (Enable)
	EPPI_CTL.DMACFG	X	
	EPPI_CTL.SUBSPLTODD	X	
PVP Settings	PVP_IPFn_CTL.CFRMT	0x01 (RGB 888)	
	PVP_IPFn_CTL.UNPACK	0x0 (No Unpacking)	0x1 (Unpack Data)
	Data Placement	RGB…RGB, not packed on PxP B0 G0 R0 B1 G1 R1	RGB…RGB, packed on PxP R1 B0 G0 R0 G2 R2 B1 G1

Table 30-12: RGB 16-Bit (YCbCr 4:4:4) with 10, 12, or 16 Bits per PIXCLK

PPI/PVP	Setting	Value/Information	
PPI settings	EPPI_CTL.XFRTYPE	0x3 (Non-ITU656 Mode, GP Mode)	
	EPPI_CTL.DLEN	0x1 (10-bit data length) 0x2 (12-bit data length 0x4 (16-bit data length)	
	EPPI_CTL.SPLTWRD	0x0 (PPI_DATA has DLEN-1 bits of Y or Cr or Cb)	
	EPPI_CTL.SPLTEO	0x0 (Do Not Split Samples)	
	EPPI_CTL.PACKEN	0x0 (Disable)	0x1 (Enable)
	EPPI_CTL.DMACFG	X	
	EPPI_CTL.SUBSPLTODD	X	
PVP Settings	PVP_IPFn_CTL.CFRMT	0x04 (RGB 16-Bit)	
	PVP_IPFn_CTL.UNPACK	0x0 (No Unpacking)	0x1 (Unpack Data)
	Data Placement	RGBRGB..., not packed on PxP 0 R0 0 G0	RGBRGB..., packed on PxP G0 R0 R1 B0

Table 30-13: RGB 565 (YCbCr 4:4:4) with 16 Bits per PIXCLK

PPI/PVP	Setting	Value/Information	
PPI settings	EPPI_CTL.XFRTYPE	0x3 (Non-ITU656 Mode, GP Mode)	
	EPPI_CTL.DLEN	0x4 (16-bit data length)	
	EPPI_CTL.SPLTWRD	0x0 (PPI_DATA has DLEN-1 bits of Y or Cr or Cb)	
	EPPI_CTL.SPLTEO	0x0 (Do Not Split Samples)	
	EPPI_CTL.PACKEN	0x0 (Disable)	0x1 (Enable)
	EPPI_CTL.DMACFG	X	
	EPPI_CTL.SUBSPLTODD	X	

Table 30-13: RGB 565 (YCbCr 4:4:4) with 16 Bits per PIXCLK (Continued)

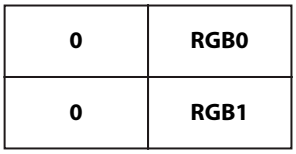
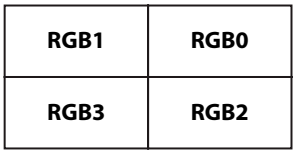
PPI/PVP	Setting	Value/Information	
PVP Settings	PVP_IPFn_CTL.CFRMT	0x02 (RGB 565)	
	PVP_IPFn_CTL.UNPACK	0x0 (No Unpacking)	0x1 (Unpack Data)
	Data Placement	RGB…RGB, not packed on PxP 	RGB…RGB, packed on PxP 

Table 30-14: RGB 666 (YCbCr 4:4:4) with 18 Bits per PIXCLK

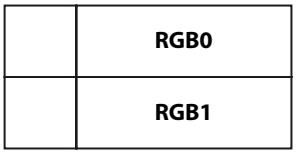
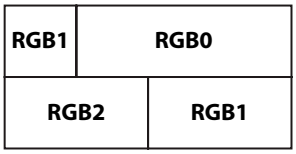
PPI/PVP	Setting	Value/Information	
PPI settings	EPPI_CTL.XFRTYPE	0x3 (Non-ITU656 Mode, GP Mode)	
	EPPI_CTL.DLEN	0x5 (18-bit data length)	
	EPPI_CTL.SPLTWRD	0x0 (PPI_DATA has DLEN-1 bits of Y or Cr or Cb)	
	EPPI_CTL.SPLTEO	0x0 (Do Not Split Samples)	
	EPPI_CTL.PACKEN	0x0 (Disable)	0x1 (Enable)
	EPPI_CTL.DMACFG	X	
	EPPI_CTL.SUBSPLTODD	X	
PVP Settings	PVP_IPFn_CTL.CFRMT	0x03 (RGB 666)	
	PVP_IPFn_CTL.UNPACK	0x0 (No Unpacking)	0x1 (Unpack Data)
	Data Placement	RGB…RGB, not packed on PxP 	RGB…RGB, packed on PxP 

Table 30-15: Bayer Format Type-1 and Bayer Format Type-2

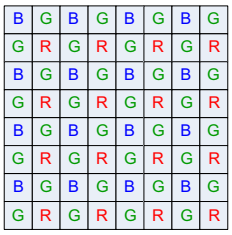
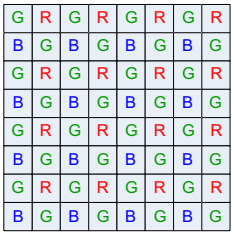
PPI/PVP	Setting	Value/Information	
PPI settings	EPPI_CTL.XFRTYPE	0x3 (Non-ITU656 Mode, GP Mode)	
	EPPI_CTL.DLEN	0x1 (10-bit data length), 0x2 (12-bit data length), or 0x4 (16-bit data length)	
	EPPI_CTL.SPLTWRD	0x0 (PPI_DATA has DLEN-1 bits of Y or Cr or Cb)	
	EPPI_CTL.SPLTEO	0x0 (Do Not Split Samples)	
	EPPI_CTL.PACKEN	0x1 (Enable)	0x1 (Enable)
	EPPI_CTL.DMACFG		
	EPPI_CTL.SUBSPLTODD	X	
PVP Settings	PVP_IPFn_CTL.CFRMT	0x05 (Bayer Type-1)	0x06 (Bayer Type-2)
	PVP_IPFn_CTL.UNPACK	0x1 (Unpack Data)	0x1 (Unpack Data)
	Data Placement		

Table 30-16: YCbCr 4:2:2 8-Bit Type 1 (CrYCbY) with 8 Bits per PIXCLK

PPI/PVP	Setting	Value/Information	
PPI settings	EPPI_CTL.XFRTYPE	0x0, 0x1, or 0x2 (ITU656 Modes); or 0x3 (Non-ITU656 Mode, GP Mode)	
	EPPI_CTL.DLEN	0x0 (8-bit data length)	
	EPPI_CTL.SPLTWRD	0x0 (PPI_DATA has DLEN-1 bits of Y or Cr or Cb)	
	EPPI_CTL.SPLTEO	0x0 (Do Not Split Samples)	
	EPPI_CTL.PACKEN	0x0 (Disable)	0x1 (Enable)
	EPPI_CTL.DMACFG	X	
	EPPI_CTL.SUBSPLTODD	X	

Table 30-16: YCbCr 4:2:2 8-Bit Type 1 (CrYCbY) with 8 Bits per PIXCLK (Continued)

PPI/PVP	Setting	Value/Information	
PVP Settings	PVP_IPFn_CTL.CFRMT	0x10 (YCbCr 4:2:2 8-Bit Type 1)	
	PVP_IPFn_CTL.UNPACK	0x0 (No Unpacking)	0x1 (Unpack Data)
	Data Placement	CrYCbY..., not packed on PxP 0 0 0 Cb0 0 0 0 Y0	CrYCbY…, packed on PxP Y1 Cr0 Y0 Cb0 Y3 Cr1 Y2 Cb1

Table 30-17: YCbCr 4:2:2 8-Bit Pair 16-bit (CrY...CbY) with 16 Bits per PIXCLK

PPI/PVP	Setting	Value/Information
PPI settings	EPPI_CTL.XFRTYPE	0x3 (Non-ITU656 Mode, GP Mode)
	EPPI_CTL.DLEN	0x4 (16-bit data length)
	EPPI_CTL.SPLTWRD	0x0 (PPI_DATA has DLEN-1 bits of Y or Cr or Cb)
	EPPI_CTL.SPLTEO	0x0 (Do Not Split Samples)
	EPPI_CTL.PACKEN	0x1 (Enable)
	EPPI_CTL.DMACFG	0x0 (PPI uses one DMA Channel)
	EPPI_CTL.SUBSPLTODD	0x0 (Disable)
PVP Settings	PVP_IPFn_CTL.CFRMT	0x19 (YCbCr 4:2:2 8-Bit Pair 16-Bit)
	PVP_IPFn_CTL.UNPACK	0x1 (Unpack Data)
	Data Placement	CrYCbY…, packed on PxP Y1 Y0 Y3 Y2

Table 30-18: Y Alone 8-Bit (YYYY) with 8 Bits per PIXCLK

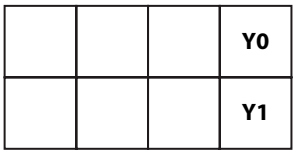
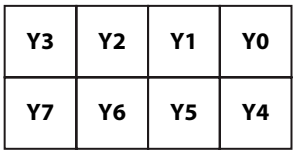
PPI/PVP	Setting	Value/Information	
PPI settings	EPPI_CTL.XFRTYPE	0x0, 0x1, or 0x2 (ITU656 Modes); or 0x3 (Non-ITU656 Mode, GP Mode)	
	EPPI_CTL.DLEN	0x0 (8-bit data length)	
	EPPI_CTL.SPLTWRD	0x0 (PPI_DATA has DLEN-1 bits of Y or Cr or Cb)	
	EPPI_CTL.SPLTEO	0x0 (Do Not Split Samples)	
	EPPI_CTL.PACKEN	0x0 (Disable)	0x1 (Enable)
	EPPI_CTL.DMACFG	X	
	EPPI_CTL.SUBSPLTODD	X	
PVP Settings	PVP_IPFn_CTL.CFRMT	0x18 (Y Alone 8-Bit)	
	PVP_IPFn_CTL.UNPACK	0x0 (No Unpacking)	0x1 (Unpack Data)
	Data Placement	YYYY..., not packed, All Ys on PxP 	YYYY..., packed on PxP, All Ys on PxP 

Table 30-19: YCbCr 4:2:2 8-Bit Type 1/2/3 (CbYCrY) with 8 Bits per PIXCLK

PPI/ PVP	Setting	Value/Information		
PPI settings	EPPI_CTL.XFRTYPE	0x0, 0x1, or 0x2 (ITU656 Modes); or 0x3 (Non-ITU656 Mode, GP Mode)		
	EPPI_CTL.DLEN	0x0 (8-bit data length)		
	EPPI_CTL.SPLTWRD	0x0 (PPI_DATA has DLEN-1 bits of Y or Cr or Cb)		
	EPPI_CTL.SPLTEO	0x1 (Split Even/Odd Samples)		
	EPPI_CTL.PACKEN	X		
	EPPI_CTL.DMACFG	0x1 (PPI uses two DMA Channels)	0x0 (PPI uses one DMA Channel)	0x0 (PPI uses one DMA Channel)
	EPPI_CTL. SUBSPLTODD	X	0x0 (Disable)	0x1 (Enable)

Table 30-19: YCbCr 4:2:2 8-Bit Type 1/2/3 (CbYCrY) with 8 Bits per PIXCLK (Continued)

PPI/ PVP	Setting	Value/Information		
PVP Settings	PVP_IPFn_CTL.CFRMT	0x10 (YCbCr 4:2:2 8-bit type1)	0x11 (YCbCr 4:2:2 8-bit type2)	0x12 (YCbCr 4:2:2 8-bit type3)
	PVP_IPFn_CTL.UNPACK	0x1 (Unpack Data)		
	Data Placement	CrYCbY..., split, and DMA separated, All Y on PxP Y1 Cr0 Y0 Cb0 Y3 Cr1 Y2 Cb1	CrYCbY..., split, CbCr comes first, followed by Y on PxP Cr1 Cb1 Cr0 Cb0 Y3 Y2 Y1 Y0	CrYCbY..., split/ subsplit, Y comes first, then CrCb on PxP Y3 Y2 Y1 Y0 Cr1/ Cb1* Cb1/ Cr1* Cr0/ Cb0* Cb0/ Cr0*

Table 30-20: SMPTE YCbCr 4:2:2 8-Bit Type 3 (CrY...CbY) with 16 Bits per PIXCLK

PPI/PVP	Setting	Value/Information	
PPI settings	EPPI_CTL.XFRTYPE	0x0, 0x1, or 0x2 (ITU656 Modes); or 0x3 (Non-ITU656 Mode, GP Mode)	
	EPPI_CTL.DLEN	0x4 (16-bit data length)	
	EPPI_CTL.SPLTWRD	0x1 (PPI_DATA contains 2 elements per word)	
	EPPI_CTL.SPLTEO	0x1 (Split Even/Odd Samples)	
	EPPI_CTL.PACKEN	X	
	EPPI_CTL.DMACFG	0x0 (PPI uses one DMA Channel)	
	EPPI_CTL.SUBSPLTODD	0x1 (Enable)	0x0 (Disable)
PVP Settings	PVP_IPFn_CTL.CFRMT	0x12 (YCbCr 4:2:2 8-bit type3)	
	PVP_IPFn_CTL.UNPACK	0x1 (Unpack Data)	
	Data Placement	CrY...CbY, split & subsplit, Y comes first, followed by CrCb on PxP Y3 Y2 Y1 Y0 Cr1/ Cb1* Cb1/ Cr1* Cr0/ Cb0* Cb0/ Cr0*	CrY...CbY, split, Y comes first, followed by CrCb on PxP Y3 Y2 Y1 Y0 Cr1/ Cb1* Cb1/ Cr1* Cr0/ Cb0* Cb0/ Cr0*

Table 30-21: UCbCr 4:2:2 16-Bit Type 1 (CrYCbY) with 10 to 16 Bits per PIXCLK

PPI/PVP	Setting	Value/Information	
PPI settings	EPPI_CTL.XFRTYPE	0x0, 0x1, or 0x2 (ITU656 Modes); or 0x3 (Non-ITU656 Mode, GP Mode)	
	EPPI_CTL.DLEN	0x1 (10-bit data length), 0x2 (12-bit data length), 0x3 (14-bit data length), or 0x4 (16-bit data length)	
	EPPI_CTL.SPLTWRD	0x0 (PPI_DATA has DLEN-1 bits of Y or Cr or Cb)	
	EPPI_CTL.SPLTEO	0x0 (Do Not Split Samples)	
	EPPI_CTL.PACKEN	0x0 (Disable)	0x1 (Enable)
	EPPI_CTL.DMACFG	X	
	EPPI_CTL.SUBSPLTODD	X	
PVP Settings	PVP_IPFn_CTL.CFRMT	0x14 (YCbCr 4:2:2 16-Bit Type 1)	
	PVP_IPFn_CTL.UNPACK	0x0 (No Unpacking)	0x1 (Unpack Data)
	Data Placement	CrYCbY… sign/zero extended to 16 bits, not packed on PxP **0** | **Cb0** **0** | **Y1**	CrYCbY… sign/zero extended to 16 bits, packed on PxP **Y0** | **Cb0** **Y1** | **Cr0**

Table 30-22: YCbCr 4:2:2 16-Bit Type 2/3, sub-split (CrYCbY) with 10 to 16 Bits per PIXCLK

PPI/PVP	Setting	Value/Information	
PPI settings	EPPI_CTL.XFRTYPE	0x0, 0x1, or 0x2 (ITU656 Modes); or 0x3 (Non-ITU656 Mode, GP Mode)	
	EPPI_CTL.DLEN	0x1 (10-bit data length), 0x2 (12-bit data length), 0x3 (14-bit data length), or 0x4 (16-bit data length)	
	EPPI_CTL.SPLTWRD	0x0 (PPI_DATA has DLEN-1 bits of Y or Cr or Cb)	
	EPPI_CTL.SPLTEO	0x1 (Split Even/Odd Samples)	
	EPPI_CTL.PACKEN	X	
	EPPI_CTL.DMACFG	0x0 (PPI uses one DMA Channel)	
	EPPI_CTL.SUBSPLTODD	0x0 (Disable)	0x1 (Enable)

Table 30-22: YCbCr 4:2:2 16-Bit Type 2/3, sub-split (CrYCbY) with 10 to 16 Bits per PIXCLK (Continued)

PPI/PVP	Setting	Value/Information	
PVP Settings	PVP_IPFn_CTL.CFRMT	0x15 (YCbCr 4:2:2 16-Bit Type 2)	0x16 (YCbCr 4:2:2 16-Bit Type 3)
	PVP_IPFn_CTL.UNPACK	0x1 (Unpack Data)	
	Data Placement	CrYCbY..., sign/zero extended to 16 bits, split not sub-split, CrCb comes first, followed by Y on PxP Cr0 Cb0 Y2 Y0	CrY...CbY, sign/zero extended to 16 bits, split and sub-split, Y comes first, followed by CrCb on PxP Cr0 Cb0 Y2 Y0

Table 30-23: SMPTE YCbCr 4:2:2 16-Bit Type 3, split (CrYCbY) with 20 to 24 Bits per PIXCLK

PPI/PVP	Setting	Value/Information	
PPI settings	EPPI_CTL.XFRTYPE	0x0, 0x1, or 0x2 (ITU656 Modes); or 0x3 (Non-ITU656 Mode, GP Mode)	
	EPPI_CTL.DLEN	0x6 (20-bit data length) or 0x7 (24-bit data length)	
	EPPI_CTL.SPLTWRD	0x1 (PPI_DATA contains 2 elements per word)	
	EPPI_CTL.SPLTEO	0x1 (Split Even/Odd Samples)	
	EPPI_CTL.PACKEN	X	
	EPPI_CTL.DMACFG	0x0 (PPI uses one DMA Channel)	
	EPPI_CTL.SUBSPLTODD	0x1 (Enable)	0x0 (Disable)
PVP Settings	PVP_IPFn_CTL.CFRMT	0x16 (YCbCr 4:2:2 16-Bit Type 3)	
	PVP_IPFn_CTL.UNPACK	0x1 (Unpack Data)	
	Data Placement	CrY...CbY, sign/zero extended to 16 bits, split and sub-split, Y comes first, followed by CrCb on PxP Y1 Y0 Cr0/Cb1* Cb0	CrY...CbY, sign/zero extended to 16 bits, split not sub-split, Y comes first, followed by CrCb on PxP Y1 Y0 Cr0/Cb1* Cb0

Table 30-24: Y Alone 16-Bit (YYYY) with 10 to 16 Bits per PIXCLK

PPI/PVP	Setting	Value/Information	
PPI settings	EPPI_CTL.XFRTYPE	0x0, 0x1, or 0x2 (ITU656 Modes); or 0x3 (Non-ITU656 Mode, GP Mode)	
	EPPI_CTL.DLEN	0x1 (10-bit data length), 0x2 (12-bit data length), 0x3 (14-bit data length), or 0x4 (16-bit data length)	
	EPPI_CTL.SPLTWRD	0x0 (PPI_DATA has DLEN-1 bits of Y or Cr or Cb)	
	EPPI_CTL.SPLTEO	0x0 (Do Not Split Samples)	
	EPPI_CTL.PACKEN	0x0 (Disable)	0x1 (Enable)
	EPPI_CTL.DMACFG	X	
	EPPI_CTL.SUBSPLTODD	X	
PVP Settings	PVP_IPFn_CTL.CFRMT	0x19 (Y Alone 16-Bit)	
	PVP_IPFn_CTL.UNPACK	0x0 (No Unpacking)	0x1 (Unpack Data)
	Data Placement	YYYY..., sign/zero extended to 16 bits, not packed, All Y on PxP 0 Y0 0 Y1	YYYY..., sign/zero extended to 16 bits, packed, All Y on PxP Y1 Y0 Y3 Y2

Table 30-25: SMPTE Y Alone 16-Bit, split, sub-split (CrYCbY) with 20 to 24 Bits per PIXCLK

PPI/PVP	Setting	Value/Information
PPI settings	EPPI_CTL.XFRTYPE	0x0, 0x1, or 0x2 (ITU656 Modes); or 0x3 (Non-ITU656 Mode, GP Mode)
	EPPI_CTL.DLEN	0x6 (20-bit data length) or 0x7 (24-bit data length)
	EPPI_CTL.SPLTWRD	0x1 (PPI_DATA contains 2 elements per word)
	EPPI_CTL.SPLTEO	0x1 (Split Even/Odd Samples)
	EPPI_CTL.PACKEN	X
	EPPI_CTL.DMACFG	0x1 (PPI uses two DMA Channels)
	EPPI_CTL.SUBSPLTODD	X

Table 30-25: SMPTE Y Alone 16-Bit, split, sub-split (CrYCbY) with 20 to 24 Bits per PIXCLK (Continued)

PPI/PVP	Setting	Value/Information
PVP Settings	PVP_IPFn_CTL.CFRMT	0x19 (Y Alone 16-Bit)
	PVP_IPFn_CTL.UNPACK	0x1 (Unpack Data)
	Data Placement	CrY...CbY..., sign/zero extended to 16 bits, split and DMA separated, All Y on PxP Y1 Y0 Y3 Y2

Table 30-26: Y Alone 24-Bit (YYYY) with 24 Bits per PIXCLK

PPI/PVP	Setting	Value/Information
PPI settings	EPPI_CTL.XFRTYPE	0x3 (Non-ITU656 Mode, GP Mode)
	EPPI_CTL.DLEN	0x7 (24-bit data length)
	EPPI_CTL.SPLTWRD	0x0 (PPI_DATA has DLEN-1 bits of Y or Cr or Cb)
	EPPI_CTL.SPLTEO	0x0 (Do Not Split Samples)
	EPPI_CTL.PACKEN	0x0 (Disable)
	EPPI_CTL.DMACFG	X
	EPPI_CTL.SUBSPLTODD	X
PVP Settings	PVP_IPFn_CTL.CFRMT	0x1A (Y Alone 24-Bit)
	PVP_IPFn_CTL.UNPACK	0x0 (No Unpacking)
	Data Placement	YYYY..., not packed, All Y on PxP 0 Y0 0 Y1

Input Formatters and Pipe Mastering

The IPFn blocks provide master control of the camera and memory pipelines. The IPFn blocks do not accept any data on their data input until the pipe is fully configured and enabled. After the camera pipe has been enabled (with the PVP_CTL.CPEN bit) or memory pipe has been enabled (with the PVP_CTL.MPEN bit), the corresponding input formatter immediately requests a block control structure list (BCL) fetch from the configuration DMA channel. If the configuration DMA does not grant the request because it is either not ready or not enabled, the pipe engine stalls either until the pipe is configured by memory mapped register (MMR) writes or until the DMA starts granting.

When the configuration DMA is granted, the input formatters fetch BCL words until the PVP_IPF0_CFG.START or PVP_IPF1_CFG.START bit is set. The PVP assumes that the BCL describes a valid pipe configuration and writes a 1 to the PVP_xxx_CFG.START bits of all involved blocks. The START bit can be seen as a self-clearing block enable bit. The self-clearing nature of this bit ensures that software does not need to perform garbage collection at or after pipe re-configuration. Blocks that are no longer used are automatically disabled.

The PVP_IPF0_CFG.START and PVP_IPF1_CFG.START bits of the input formatters have additional purpose --- to enable the entire camera or memory pipe. While the order of BCSs inside a BCL does not matter, only the very last BCS writes into the PVP_IPF1_CFG register.

After the PVP_IPF0_CFG.START or PVP_IPF1_CFG.START bit has been written, the IPF0 starts accepting data from the video sub-system (PPIs or PIXC), and the camera pipe starts processing. Or, IPF1 starts requesting data from data input DMA, and the memory pipe starts processing.

For a more detailed description of pipeline operations and functionality, see the *Programming Model*.

Output Formatters (OPFn)

The output formatters collect the data results of PVP processing blocks, apply final formatting, and forward the results to the DMA channels. The OPF0, OPF1, and OPF2 formatters serve the camera pipes. The OPF3 formatter serves the memory pipe. Each OPF is associated with a specific DMA channel. The input to each output formatter is selectable from the output of the PVP blocks, as shown in the following tables. For a graphical overview of PVP block connectivity, see *Configuring Pipe Structure*.

Table 30-27: OPF0, OPF1, and OPF2 (Camera Pipe) Block Connectivity

OPF0/1/2 Block I/O	Data Format	PVP Block Connect Selections	Port Connect
Input 0	s16, u16, s32, u32	IPF0 CNV0 CNV1 CNV2 CNV3 PMA ACU PEC THC0 THC1 IIM0 IIM1	0, 1, 2 0 0 0 0 0, 1, 2 0 0 0 0 0 0
Input 1	n/a	n/a	n/a
Output 0	s16, u16, s32, u32	DMA	n/a
Output 1	n/a	n/a	n/a

Table 30-27: OPF0, OPF1, and OPF2 (Camera Pipe) Block Connectivity (Continued)

OPF0/1/2 Block I/O	Data Format	PVP Block Connect Selections	Port Connect
Output 2	n/a	n/a	n/a

Table 30-28: OPF3 (Memory Pipe) Block Connectivity

OPF3 Block I/O	Data Format	PVP Block Connect Selections	Port Connect
Input 0	s16, u16, s32, u32	IPF1 CNV0 CNV1 CNV2 CNV3 PMA ACU PEC THC0 THC1 IIM0 IIM1 UDS	0, 1, 2 0 0 0 0 0, 1, 2 0 0 0 0 0 0 0
Input 1	n/a	n/a	n/a
Output 0	s16, u16, s32, u32	DMA	n/a
Output 1	n/a	n/a	n/a
Output 2	n/a	n/a	n/a

OPFn Data Packing

To ensure efficient memory bandwidth use, the OPFn blocks support data packing. Ideally, the PVP results are packed into 32-bit entities and forwarded to the DMA channel, which is also programmed to 32-bit mode by setting `PVP_OPFn_CTL.OSIZE` =0 and by setting the `DMA_CFG.PSIZE` =2 for each respective DMA channel.

The `PVP_OPFn_CTL.OSIZE` =0 setting instructs the OPFn block to accept a full 32-bit word from the data source. In this case, packing is not possible, and `PVP_OPFn_CTL.OSIZE` =0 and `DMA_CFG.PSIZE` =2 are mandatory settings.

The `PVP_OPFn_CTL.ISIZE` =1 setting instructs the OPFn block to accept 16-bit words from the data source. In this case, there are multiple data packing options. To use data packing, set `PVP_OPFn_CTL.OSIZE` =0 and set `DMA_CFG.PSIZE` =2. For no data packing, set `PVP_OPFn_CTL.OSIZE` =1 and set `DMA_CFG.PSIZE` =1 for 16-bit transfers between the OPFn block and the corresponding DMA channel. The `PVP_`

OPFn_CTL.IUP16 bit determines whether the OPFn blocks latch the lower 16 bits or the upper 16 bits from their 32-bit input.

The PVP_OPFn_CTL.ISIZE =2 setting instructs the OPFn block to accepts 8-bit bytes from the data source. To pack values to a 32-bit output, set PVP_OPFn_CTL.OSIZE =0 and set DMA_CFG.PSIZE =2. To only pack two values into a 16-bit output, set PVP_OPFn_CTL.OSIZE =1 and set DMA_CFG.PSIZE =1. To disable packing, set PVP_OPFn_CTL.OSIZE =2 and set DMA_CFG.PSIZE = 0. The PVP_OPFn_CTL.IUP16 bit selects between bits [7:0] and bits [23:16] on the inputs.

The PVP_OPFn_CTL.ISIZE =3 setting instructs the OPF to accept 4-bit nibbles from the data source. Nibbles must always be packed to 8-, 16-, or 32-bit entities as described in the OPFn data packing options table. This packing results in dual, quad or octal nibble groups per DMA transfer. The PVP_OPFn_CTL.IUP16 bit selects between bits [3:0] and bits [19:16] on the inputs.

NOTE: Data packing has requirements for the horizontal size of data frames, which must be a multiple of the packing ratio.

The following table summarizes the OPFn data packing options.

Table 30-29: OPFn Data Packing Options

PVP_OPFn_CTL.ISIZE	PVP_OPFn_CTL.OSIZE	DMA_CFG.PSIZE	PVP_OPFn_CTL.IUP16	32-bit DMA word
0	0	2	0	D0[31:0]
1	1	1	0	0, D0[15:0]
			1	0, D0[31:16]
	0	2	0	D1[15:0], D0[15:0]
			1	D1[31:16], D0[31:16]
2	2	0	0	0, 0, 0, D0[7:0]
			1	0, 0, 0, D0[23:16]
	1	1	0	0, 0, D1[7:0], D0[7:0]
			1	0, 0, D1[23:16], D0[23:16]
	0	2	0	D3[7:0], D2[7:0], D1[7:0], D0[7:0]
			1	D3[23:16], D2[23:16], D1[23:16], D0[23:16]

Table 30-29: OPFn Data Packing Options (Continued)

PVP_OPFn_CTL.ISIZE	PVP_OPFn_CTL.OSIZE	DMA_CFG.PSIZE	PVP_OPFn_CTL.IUP16	32-bit DMA word
3	2	0	0	0, 0, 0, 0, 0, 0, D1[3:0], D0[3:0]
			1	0, 0, 0, 0, 0, 0, D1[19:16], D0[19:16]
	1	1	0	0, 0, 0, 0, D3[3:0], D2[3:0], D1[3:0], D0[3:0]
			1	0, 0, 0, 0, D3[19:16], D2[19:16], D1[19:16], D0[19:16]
	0	2	0	D7[3:0], D6[3:0], D5[3:0], D4[3:0], D3[3:0], D2[3:0], D1[3:0], D0[3:0]
			1	D7[19:16], D6[19:16], D5[19:16], D4[19:16], D3[19:16], D2[19:16], D1[19:16], D0[19:16]

If the OPFn block is receiving unsigned data and its PVP_IPFn_CTL.QFRMT bit is set, it is helpful for the OPFn block to correct the shift position back before transmitting the results to DMA. The PVP_IPFn_CTL.QFRMT bit right shifts the input by one bit to keep data unsigned. The OPF corrects the right shift by the IPF. For this purpose, the PVP_OPFn_CTL.QFRMT bit may be used to shift the results to the left by one bit position.

OPFn Output FIFOs

To avoid data loss during cases of temporary peak loads on system buses, the OPFn blocks have local FIFOs to supplement the DMA FIFOs. The width of the OPFn blocks' FIFO vary with the PVP_OPFn_CTL.OSIZE bit field settings. The individual OPFn blocks have different FIFO depths:

- OPF0: 380 entries x 32 bits or 960 x 16 bits or 1920 x 8 bits
- OPF1: 1024 entries x 32 bits or 2048 x 16 bits or 4196 x 8 bits
- OPF2: 512 entries x 32 bits or 1024 x16 bits or 2048 x 8 bits
- OPF3: 32 entries x 32 bits or 64 x 16 bits or 128 x 8 bits

It is important to note the capacities of the OPFn block FIFOs and use this information when configuring camera and memory pipes. *Be sure to assign OPF1 to the camera pipeline with the highest output data rate.* For example, it is best to route 32-bit results that come out of the CNVn blocks, the ACU block, or the IIMn blocks (running at full input frame resolution) to OPF1. *The OPF0 block should be used for the camera pipe with the lowest output data rate.* For example, it is best to route items that are subject to data reduction (windowing, sub sampling, run-length encoding) or items that have small data widths (threshold index, edge classes) to OPF1. *The OPF2 block should be used for normal data rate throughput.* Unexpected overflow status is reported by the PVP_STAT.OPF0OVF, PVP_STAT.OPF1OVF, and PVP_STAT.OPF2OVF bits, and latched interrupts for overflow are indicated by the PVP_ILAT.OPF0OVF, PVP_ILAT.OPF1OVF, and PVP_ILAT.OPF2OVF bits.

NOTE: The OPF3 block has a relatively small FIFO, because the memory pipe has been designed to never overflow.

The OPFn blocks support finish signaling to the DMA. For more information about this feature, see *Finish Commands*.

Threshold-Histogram-Compression (THCn)

The PVP features two threshold-histogram-compression blocks. These blocks implement a collection of statistical and range reduction signal processing functions. The input to the THC blocks can be one of the following described in the following table. For a graphical overview of all PVP block interconnections, see *Configuring Pipe Structure*.

Table 30-30: THCn Block Connectivity

THC0/1 Block I/O	Data Format	PVP Block Connect Selections	Port Connect
Input 0	s32	IPF0 IPF1 PMA CNV0 CNV1 CNV2 CNV3 ACU PEC	0,1, 2 0, 1, 2 1, 2 0 0 0 0 0 2
Input 1	n/a	n/a	n/a
Output 0	s32 (result)	IIM0, IIM1, OPF0, OPF1, OPF2, OPF3	n/a
Output 1	n/a	n/a	n/a
Output 2	n/a	n/a	n/a

The figure gives the detailed overview of a THCn block.

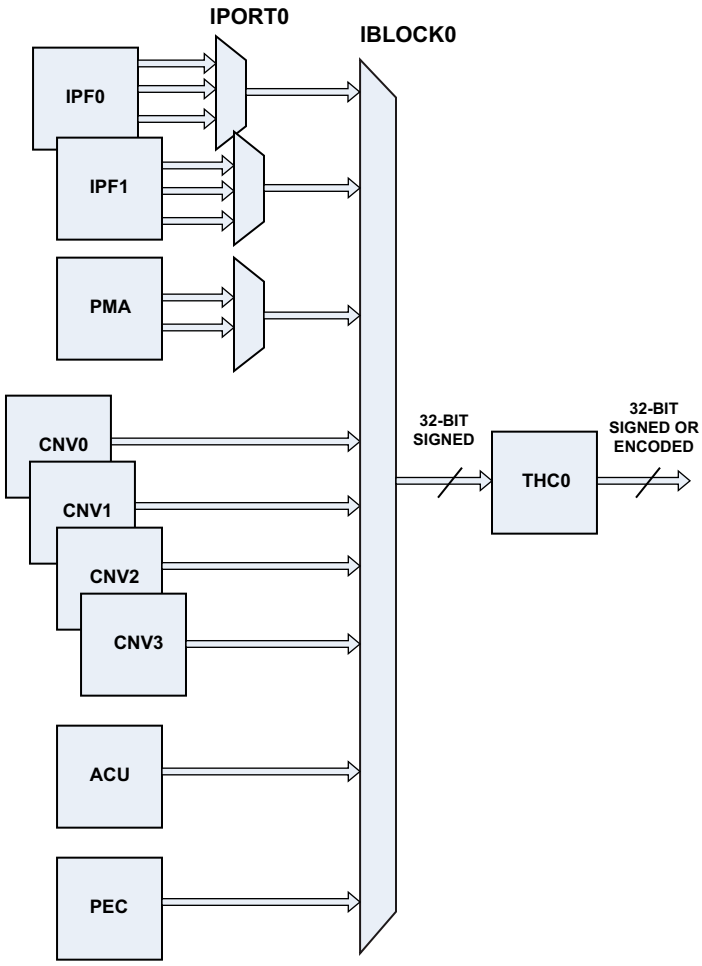


Figure 30-6: Threshold Histogram Compression Block Diagram

The PVP_THCn_CTL.ZEXT bit determines how the input data is handled. When set to 1, the lower 16-bits of input data are zero extended to 32-bits and are used for further processing. The upper 16-bits are used as an angle input. When the bit is cleared (= 0), the entire 32-bits of input data are used for further processing. The angle input is set to 0.

The stages of THCn block operation are as follows.

- Threshold Unit
- Histogram Unit
- Compression Unit

The THCn overview figure shows how these stages contribute to THC operations.The details of each unit are described in the sections that follow.

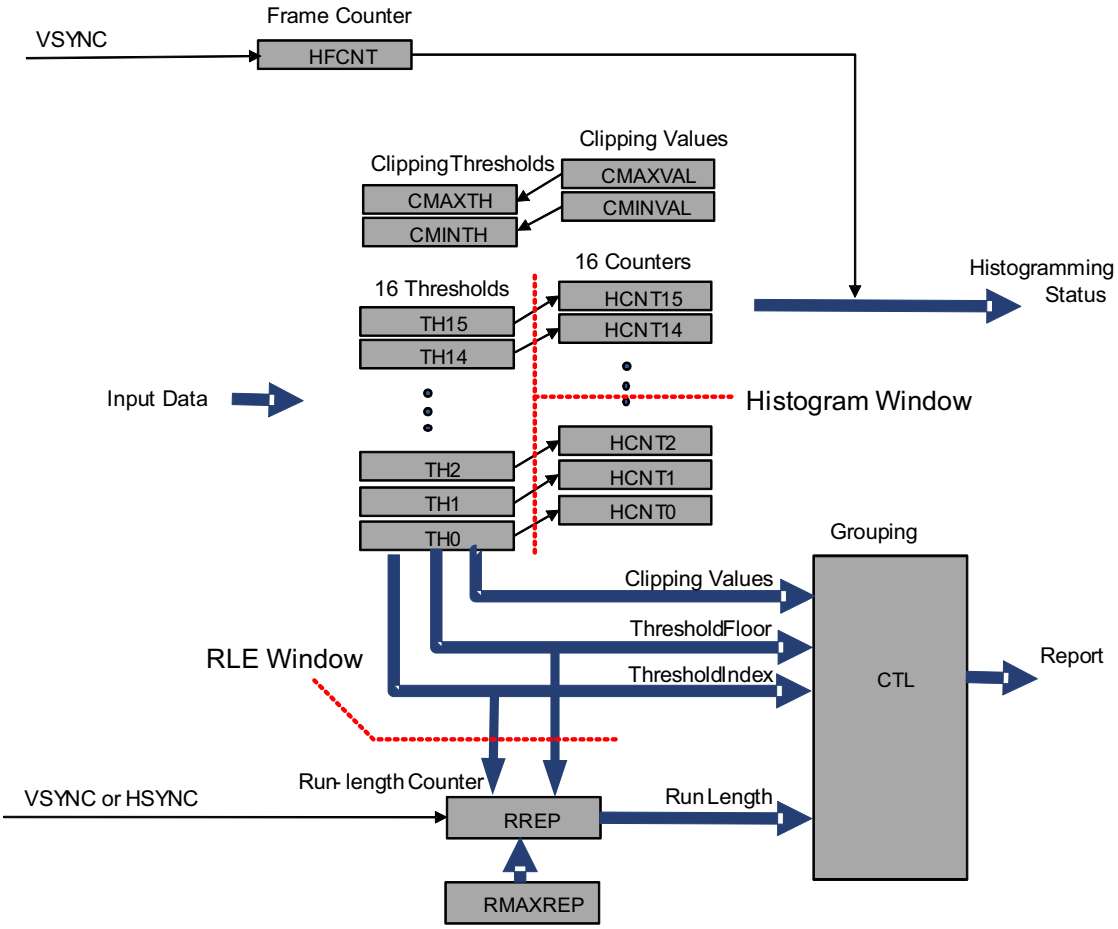


Figure 30-7: THCn Overview

THCn Threshold Unit

The THCn threshold unit operates on the incoming pixels for the THCn block. The output of the threshold unit is a 32-bit pixel out value and a 4-bit index value. This unit supports three modes of operation based on the `PVP_THCn_CTL.MODE` bits.

- *Clipping Mode* — In this mode, the input pixel is classified in one of the three ranges. The input data is first compared to the value in the `PVP_THCn_CMINTH` register. If this value less than the value programmed in the register, the pixel out value is the `PVP_THCn_CMINVAL` register value, and the index output is 0x0. If the input data is greater than the value programmed in the `PVP_THCn_CMAXTH` register, the pixel out value is the `PVP_THCn_CMAXVAL` value, and the index output is 0x2. If the input data lies between the two values the pixel out value is equal to the input data, and the index output is 0x1.
- *Quantization Mode* — In this mode, the input pixels are compared to the 16 threshold value registers (`PVP_THCn_TH0-PVP_THCn_TH15`) and are classified as belonging to one of sixteen bins. The registers must be programmed in ascending order from `PVP_THCn_TH0` to `PVP_THCn_TH15`. The pixel out takes one of these 16 threshold values based on floor logic. If the input is less than `PVP_THCn_TH0`, the pixel

out is the highest threshold value among the 16 threshold values. The index output is the number of the threshold register that the pixel out takes.

- *Hysteresis Mode* — In this mode, the 16 threshold value registers (PVP_THCn_TH0-PVP_THCn_TH15) are interpreted as eight pairs of threshold values, starting from PVP_THCn_TH0–PVP_THCn_TH1, then PVP_THCn_TH2–PVP_THCn_TH3, and so on. The register pairs form a range. If the input pixel value falls in this range, the pixel out depends on the previous output. The pixel out is the higher register in one of the eight pairs of threshold registers. The index output is the number of the threshold register pair that the pixel out takes.

The hysteresis example figure demonstrates hysteresis functionality. The rounded numbers are index output when the input pixel falls in that region.

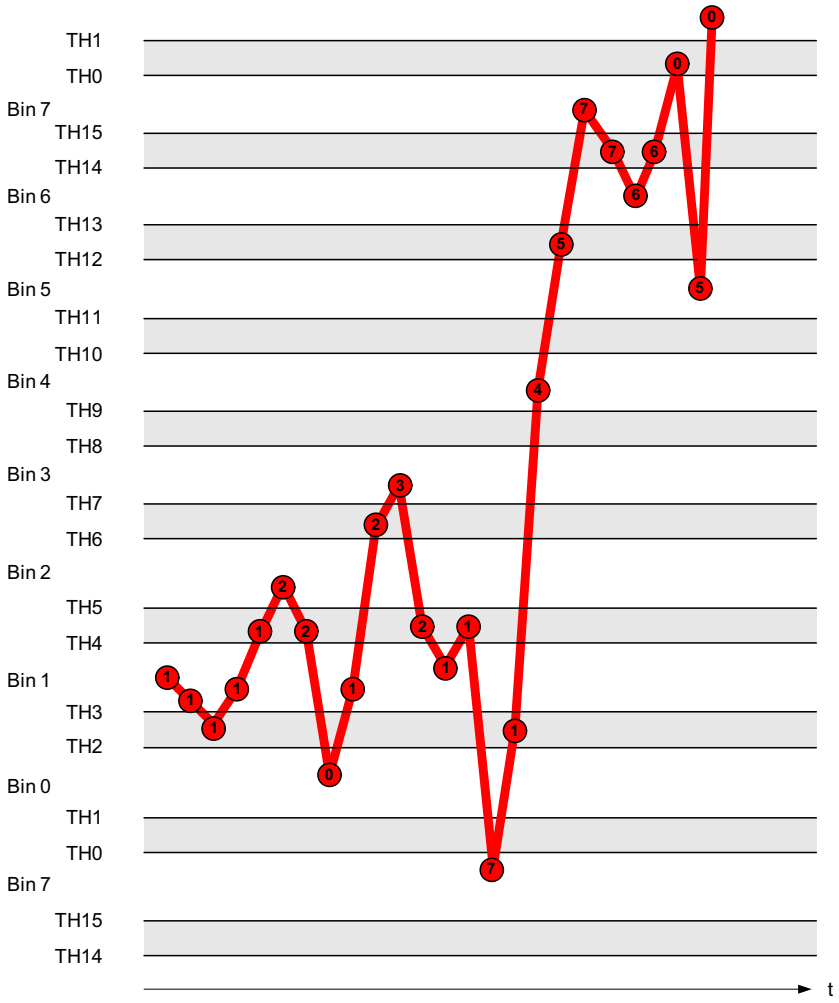


Figure 30-8: Hysteresis Example

If the pixel value falls in the gray region, the output index is computed based on the pseudo code below:

```
if (previous_bin == upper_bin)
index = upper_bin
else if (previous_bin = lower_bin)
```

```
index = lower_bin
else if (previous_bin > upper_bin)
index = upper_bin
else
index = lower_bin
```

THCn Histogram Unit

The THCn histogram unit has 16 histogram counters, with each counter corresponding to one index. The counters run for the number of frames programmed in the PVP_THCn_HFCNT register. After the programmed number of frames, the histogram counters are cleared and are restarted. The histogram counters are moved to the PVP_THCn_HFCNT_STAT registers at every frame. The value of the current frame count also is available in the PVP_THCn_HFCNT_STAT register.

When the threshold unit is operating in hysteresis mode, the index output ranges from 0 to 7, so only the PVP_THCn_HCNT0_STAT to PVP_THCn_HCNT7_STAT registers contain valid data. When the threshold unit is operating in clipping mode, the index output ranges from 0 to 2, and only the PVP_THCn_HCNT0_STAT to PVP_THCn_HCNT2_STAT registers have valid data. Each of the histogram counters is 32 bits wide. If the counter reaches the highest value of 0xFFFFFFFF before the expiration of the value in the PVP_THCn_HFCNT register, the counter does not overflow. Rather, the counter saturates at that value.

The THCn histogram unit can be enabled or disabled using the PVP_THCn_CTL.HISTEN bit.

THCn Compression Unit

The THCn compression unit uses the run-length-encoding (RLE) technique to compress data. In most cases, the output from the threshold unit has a lot of repetition. Converting the sequence of data values into a report containing value and run-length count reduces throughput at the output of the THCn block.

The value and run-length are grouped into a single word based on the PVP_THCn_CTL.OFRMT bits. The run-length indicates the actual number of pixels and not repetitions, so the minimum run-length value is 1.

If the PVP_THCn_CTL.RLEFRAME bit is 0, a report is generated at the end of a row and the run-length counter is reset. If = 1, compression works across rows in the frame. Due to the nature of compression, a variable number of reports are generated per row or per frame. The PVP_THCn_RREP_STAT register contains the actual number of reports generated per frame. A value can be set in the PVP_THCn_RMAXREP register to limit the number of reports generated per row or per frame. When the number of reports reaches the limit, the last report's run-length field is cleared.

Note that some formats (selectable with the PVP_THCn_CTL.OFRMT bits) disable compression. Also, some formats use angle information to generate the final report.

A typical usage of compression unit is to set the maximum number of reports in PVP_THCn_RMAXREP register. Program the output DMA to transfer the maximum number of reports. After all the reports are generated for the frame, the compression unit sends a *Finish Command* to DMA. This command ensures that the work unit moves to the next work unit despite the variable data per work unit. The actual number of reports generated can be read from PVP_THCn_RREP_STAT status register.

THCn Windowing

The THCn histogram and THCn compression units support windowing to filter incoming pixels.

The THCn histogram window supports filtering pixels either inside or outside the window. The THCn window mode is selected with the PVP_THCn_CTL.HISTWM bits. The window coordinates are specified by the PVP_THCn_HHPOS, PVP_THCn_HVPOS, PVP_THCn_HHCNT, and PVP_THCn_HVCNT registers.

The THCn compression window supports filtering pixels only inside the window. The THCn window mode is selected with the PVP_THCn_CTL.RLEWM bits. The window coordinates are specified by the PVP_THCn_RHPOS, PVP_THCn_RVPOS, PVP_THCn_RHCNT, and PVP_THCn_RVCNT registers. The minimum supported window size is 16x16.

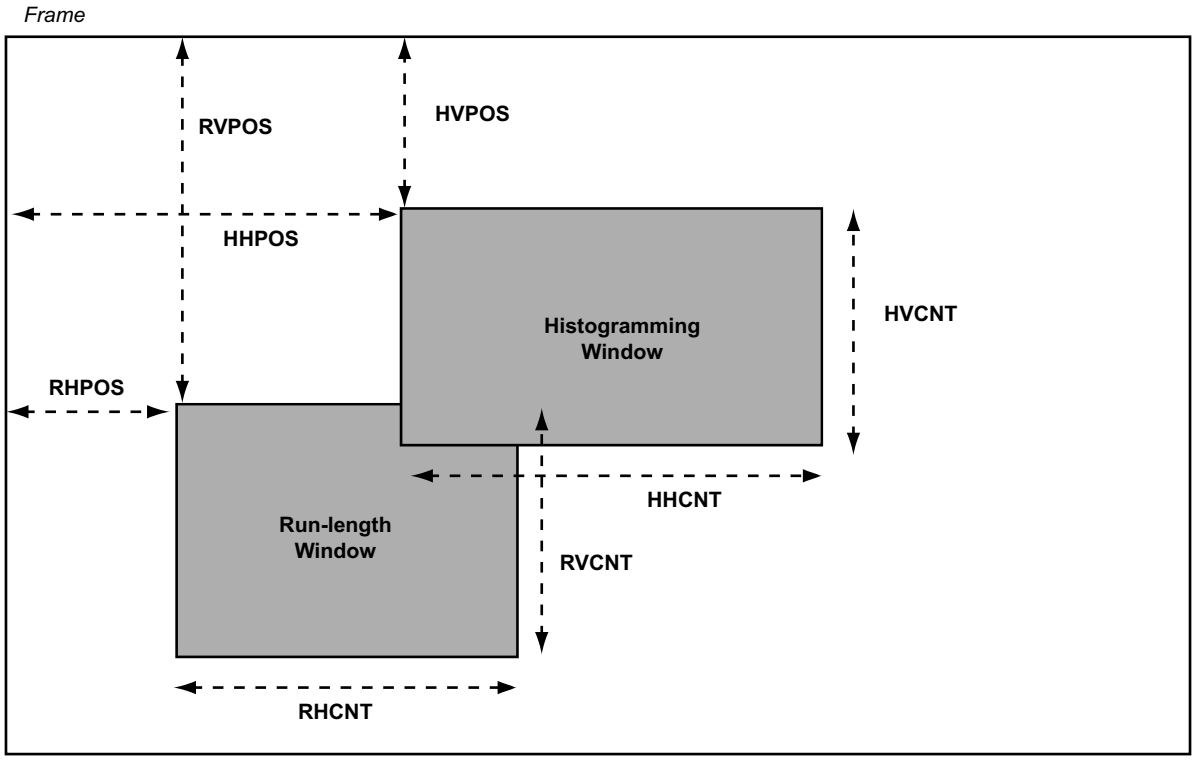


Figure 30-9: Definition of RLE and Histogram Windows

Convolution (CNVn)

Convolution is a mathematical operation on two signals producing a third signal which is typically viewed as a modified version of one of the original signals. The PVP has four convolution blocks (CNVn).

The convolution block implements a 2D convolution of the input pixels with a 5x5 coefficient matrix. The actual mathematical operation is described in the equation below.

$$Y_{i,j}\Big|_{i=0-(M-1),\,j=0-(N-1)} = \sum_{m=0-4,\,n=0-4} X_{i+m-2,\,j+n-2} * C_{m,n}$$

where:

- N = number of columns in the input data
- M = number of rows in the input data
- Y = MxN output data
- X = MxN input data
- C = 5x5 coefficient data

Note that the strict mathematical formula for convolution mirrors the coefficients in both directions before the MAC operation. For CNV blocks, the coefficients are multiplied in place to simplify the operation (similar to Correlation). Programs can always put the 5x5 coefficients in different order to realize either convolution or correlation.

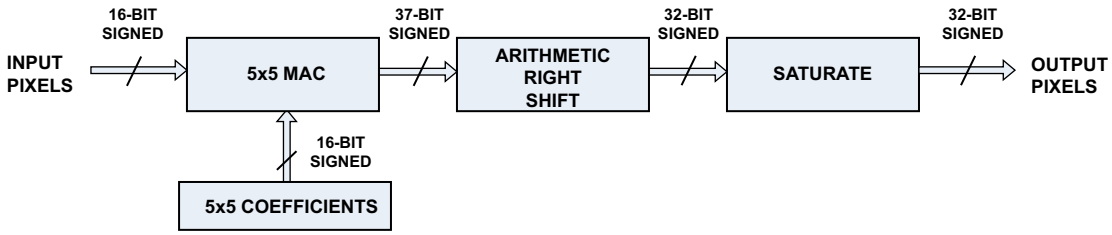


Figure 30-10: CNVn Overview

The following figures and tables describe the data flow within the CNV block. For a graphical overview of all PVP block interconnections, see *Configuring Pipe Structure*.

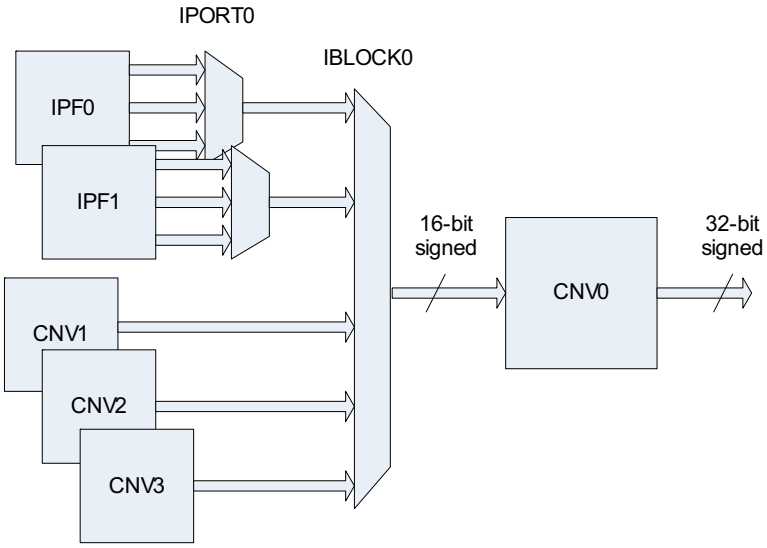


Figure 30-11: Convolution Block Connectivity

Table 30-31: CNV0 Block Connectivity

CNV0 Block I/O	Data Format	PVP Block Connect Selections	Port Connect
Input 0	s16	IPF0 IPF1 CNV1 CNV2 CNV3	0, 1, 2 0, 1, 2 0 0 0
Input 1	n/a	n/a	n/a
Output 0	s32 (Result)	CNV1, CNV2, CNV3, PMA, ACU, PEC, THC0, THC1, IIM0, IIM1. OPF0, OPF1, OPF2, OPF3	n/a
Output 1	n/a	n/a	n/a
Output 2	n/a	n/a	n/a

Table 30-32: CNV1 Block Connectivity

CNV1 Block I/O	Data Format	PVP Block Connect Selections	Port Connect
Input 0	s16	IPF0 IPF1 CNV0 CNV2 CNV3	0, 1, 2 0, 1, 2 0 0 0
Input 1	n/a	n/a	n/a
Output 0	s32 (Result)	CNV0, CNV2, CNV3, PMA, ACU, PEC, THC0, THC1, IIM0, IIM1. OPF0, OPF1, OPF2, OPF3	n/a
Output 1	n/a	n/a	n/a
Output 2	n/a	n/a	n/a

Table 30-33: CNV2 Block Connectivity

CNV2 Block I/O	Data Format	PVP Block Connect Selections	Port Connect
Input 0	s16	IPF0 IPF1 CNV0 CNV1 CNV3	0, 1, 2 0, 1, 2 0 0 0
Input 1	n/a	n/a	n/a
Output 0	s32 (Result)	CNV0, CNV1, CNV3, PMA, ACU, PEC, THC0, THC1, IIM0, IIM1. OPF0, OPF1, OPF2, OPF3	n/a
Output 1	n/a	n/a	n/a
Output 2	n/a	n/a	n/a

Table 30-34: CNV3 Block Connectivity

CNV3 Block I/O	Data Format	PVP Block Connect Selections	Port Connect
Input 0	s16	IPF0 IPF1 CNV0 CNV1 CNV2	0, 1, 2 0, 1, 2 0 0 0
Input 1	n/a	n/a	n/a
Output 0	s32 (Result)	CNV0, CNV1, CNV2, PMA, ACU, PEC, THC0, THC1, IIM0, IIM1. OPF0, OPF1, OPF2, OPF3	n/a
Output 1	n/a	n/a	n/a
Output 2	n/a	n/a	n/a

The CNVn block expects 16-bit signed input data and drives 32-bit signed output data. The key features of the CNVn block are:

- When the PVP_CNVn_CTL.SAT32 bit =0, the output is saturated to a 16-bit value and is sign extended to drive the 32-bit output. When the bit =1, the output is saturated to a 32-bit value.
- The boundary pixels at the output require input pixels beyond the edges for computation. When the PVP_CNVn_CTL.ZEROFILL bit =1, the input pixels outside the edges are considered to be 0. When the bit =0, the edges are duplicated beyond the boundary of the input pixels.
- The PVP_CNVn_CTL.SHIFT bit controls output arithmetic right shift from 0 to 31 bits. This shift allows the coefficients to be in fractional format with full control of the placement of the binary point.
- The CNVn block supports down scaling by dropping output pixels in both directions. The PVP_CNVn_SCALE bits are used to set the horizontal downscale factor, and the PVP_CNVn_SCALE.VSCL bits are used to set the vertical downscale factor. The CNVn block supports decimation in powers of 2. The maximum horizontal downscale factor is 1024, and the vertical factor is 512. The value programmed in these registers is one less than the actual scale factor. So, a scale factor of 0 disables down scaling.

  The PVP_CNVn_SCALE.HSCL and PVP_CNVn_SCALE.VSCL field settings must not change on the fly if the output of the down scaling CNVn block supplies another CNVn block. If so, the PVP_IPFn_PIPECTL.DRAIN bit must be set for the new configuration.
- The CNVn block supports 5x5 convolution kernels. The block automatically supports any smaller 2D kernels like 3x3 by centering the kernel in the 5x5 matrix and filing the surrounding values with zeros. The coefficients for the y^{th} row are set in the PVP_CNVn_C00C01, PVP_CNVn_C02C03, and PVP_CNVn_C04 registers.
- The accumulator in the CNVn block is 37-bits wide—no overflow is possible during MAC operations.

NOTE: A down scaled CNVn must not dynamically change the scaling ratio when supplying another CNVn block. A CVNn block may only drive another CNVx as long as it does not downscale or the scaling ratio does not alter while the pipe is enabled.

Red Pixel Substitution

For the benefit of red-clear-clear-clear (RCCC) data processing, Convolution Block CNV1 provides a unique feature called red pixel substitution. (This feature is not supported by the CNV0, CNV2 and CNV3 blocks). If paired with IPF0's Color Separation mode, red pixel substitution can substitute the red pixels by a mean value of the surrounding clear pixels.

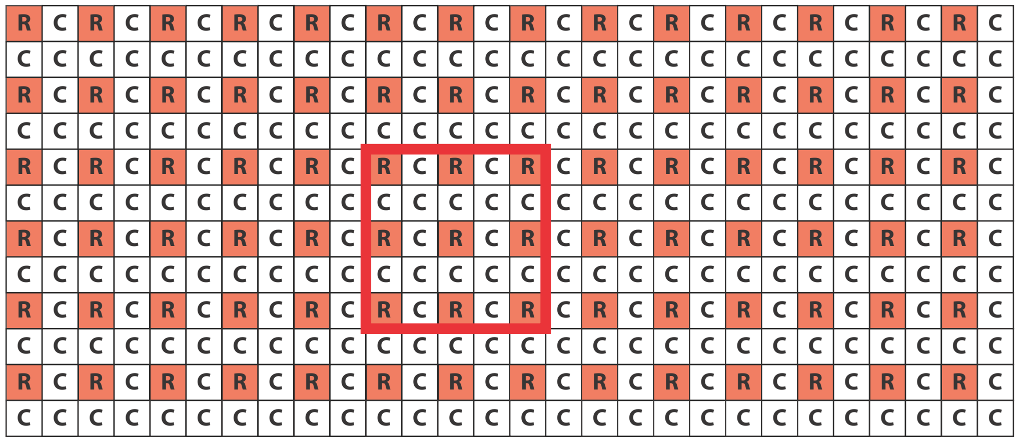


Figure 30-12: CNV RCCC Received From Sensor

When CNV1 receives a Bayer or a RCCC data stream, red pixel substitution performs convolution/correlation, shift and saturate operation only on the red pixel. The blue and green pixels (the clear pixels) are passed to the output without modification. In this mode, the `PVP_CNVn_CTL.SAT32` control bit must be set to zero. Only then can accumulation results saturate to 16 bits and match the data range of the unmodified 16-bit clear pixels. All pixels are sign-extended to 32 bits on the output.

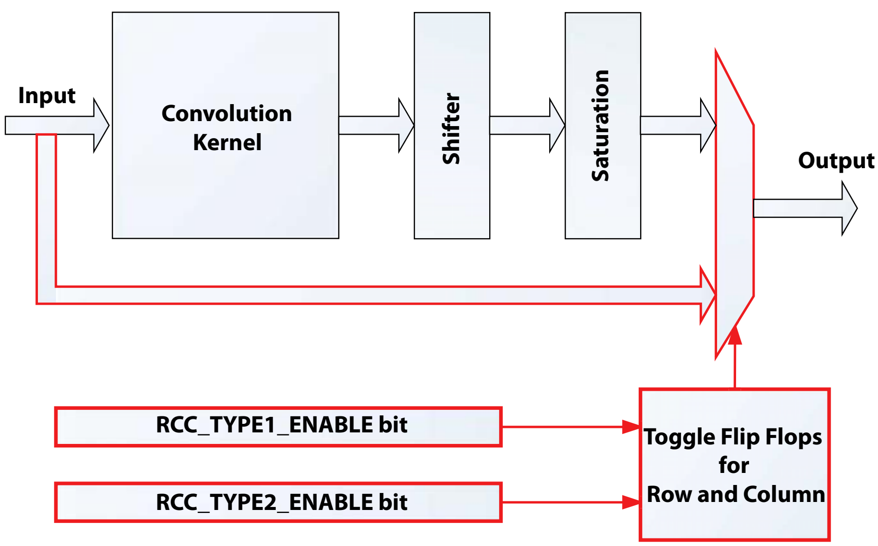


Figure 30-13: CNV RCCC Convolution Kernel Flow

The red pixel substitution mode is enabled by the `PVP_CNVn_CTL.RFRMT0` bit. The `PVP_CNVn_CTL.RFRMT1` bit distinguishes between Bayer Type 1 or Type 2 configuration.

There are multiple strategies for substituting the red pixel in an RCCC data stream. The CNV1 block allows for many types of mean value generation that interpolate the missing clear value (in place of the red pixel). The following figure provides an overview of the most prominent convolution kernels.

Type 0:

0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	0	1	0	0
0	0	0	0	0

Shift by 2

Type 1:

0	0	0	0	0
0	1	1	1	0
0	1	0	1	0
0	1	1	1	0
0	0	0	0	0

Shift by 3

Type 2:

0	0	0	0	0
0	1	3	1	0
0	3	0	3	0
0	1	3	1	0
0	0	0	0	0

Shift by 4

Type 3:

0	0	0	0	0
0	1	2	1	0
0	2	4	2	0
0	1	2	1	0
0	0	0	0	0

Shift by 4

Type 4:

0	1	0	1	0
1	1	1	1	1
0	1	0	1	0
1	1	1	1	1
0	1	0	1	0

Shift by 4

Type 5:

0	1	0	1	0
1	3	3	3	1
0	3	0	3	0
1	3	3	3	1
0	1	0	1	0

Shift by 5

Type 6:

0	1	1	1	0
1	2	2	2	1
1	2	4	2	1
1	2	2	2	1
0	1	1	1	0

Shift by 5

Type 7:

0	1	0	1	0
1	2	3	2	1
0	3	4	3	0
1	2	3	2	1
0	1	0	1	0

Shift by 5

Figure 30-14: CNV RCCC Filter Candidates

Polar Magnitude and Angle Block (PMA)

The PVP features one polar magnitude and angle block (PMA). The block takes two 16-bit signed inputs as coordinates in the Cartesian form (x, y) and converts them into Polar form (Magnitude, Angle φ).

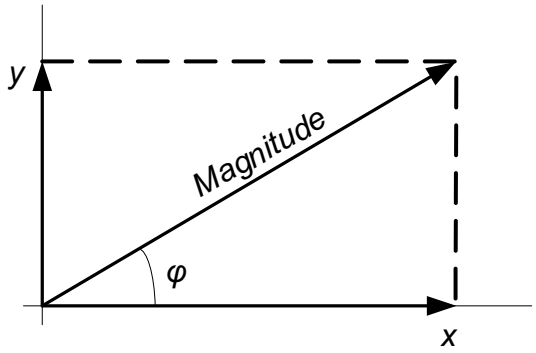


Figure 30-15: Cartesian versus Polar form

The PMA block applies the Pythagorean mathematical formulas to obtain magnitude and angle, as shown.

$$Magnitude = \sqrt{|x|^2 + |y|^2} \qquad \varphi = \arctan(\frac{y}{x})$$

Figure 30-16: PMA Equations

The magnitude is output as a 16-bit unsigned value, ranging from 0 to the square root of 2. For x = y = 0x7FFF, the PMA block outputs 0xB503. For the special case of x = y = 0x8000, the PMA block outputs 0xB504.

The angle φ is calculated to five bits of resolution (11.25°) and an accuracy of ±0.25°. Mathematically, the angle is a signed value. However, since the angle is always zero-extended when output to 16-bit or 32-bit buses, it can also be interpret as an unsigned binning value.

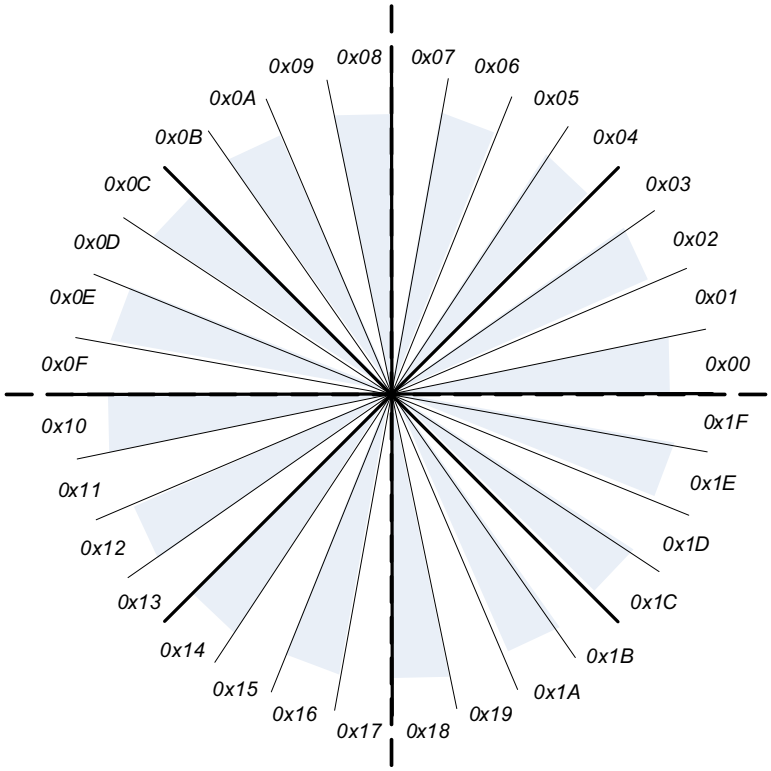


Figure 30-17: PMA Angle Binning

The PMA sets the angle to 0x08 (+90°) if the x input value is 0 and the y input is a positive value and to 0x18 (-90°) if the y input is negative. If both, x and y are zero, the PMA outputs 0x1F by convention.

Other than data flow configuration, the PMA block does not have any control or status registers. Both of its inputs can be individually configured to receive data from any of the convolution blocks or from either input formatter.

NOTE: Care is required in that both inputs are timed consistently and are not subject to non-matching latency in up front pipeline configuration.

The PMA block has three output ports. If PMA is enabled, all three 32-bit ports are always active as follows.

- Port 0 drives the 16-bit unsigned magnitude. The upper 16 bits are always driven as zeros.
- Port 1 drives the 5-bit angle value. The upper 27 bits are always driven as zeros.
- Port 2 drives a combined format. The lower 16 bits contain the magnitude, bits 16 to 20 drive the angle, and the upper ones are always zero. The Port 2 signal is not only good for being streamed to system memory. The PEC block and THCn blocks have special functionality to deal with this format.

The table and figure show the PMA block data flow and PVP block connections. For a graphical overview of all PVP block interconnections, see *Configuring Pipe Structure*.

Table 30-35: PMA Block Connectivity

PMA Block I/O	Data Format	PVP Block Connect Selections	Port Connect
Input 0	s16	IPF0 IPF1 CNV0 CNV1 CNV2 CNV3	0, 1, 2 0, 1, 2 0 0 0 0
Input 1	s16	IPF0 IPF1 CNV0 CNV1 CNV2 CNV3	0, 1, 2 0, 1, 2 0 0 0 0
Output 0	u32	PEC, THC0, THC1, IIM0, IIM1, OPF0, OPF1, OPF2, OPF3	0 (magnitude only) 1 (angle only) 2 (magnitude+angle)
Output 1	n/a	n/a	n/a
Output 2	n/a	n/a	n/a

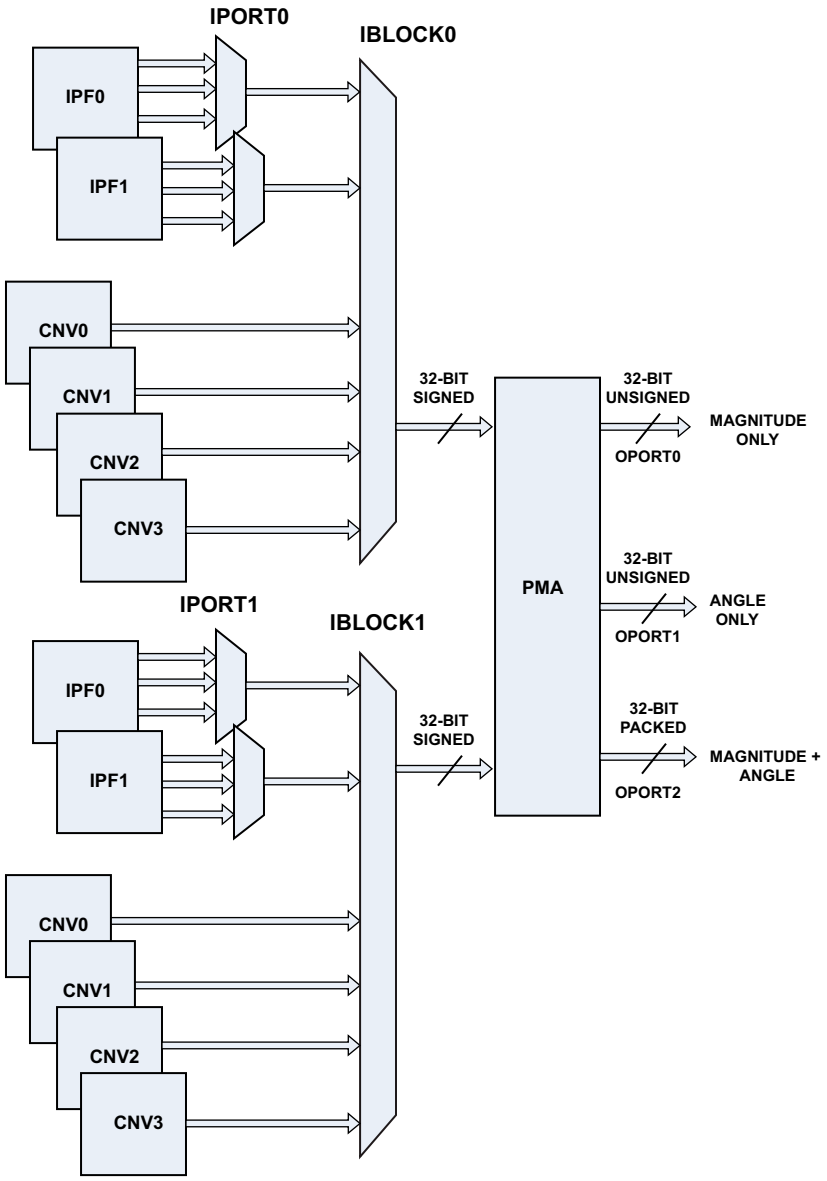


Figure 30-18: PMA Connectivity

Arithmetic Control Unit (ACU)

The arithmetic control unit (ACU) provides basic 32-bit addition, subtraction, multiplication, and division. This block also includes a 48-bit accumulator, which can normalize and saturate the output. While the ACU is important building block in many video applications, this block also adds significant general-purpose value to the processor.

The ACU block has two 32-bit inputs (x, y) and one 32-bit output (result). It features a number of registers that hold constant operands. The *offset* value can be used for addition and subtraction. The *factor* value

can be used for multiplication and division. The *shift* value is used for normalization and the *min* and *max* pair of registers control the saturation.

As shown in the ACU overview figure, the individual operation can also be combined. Multiple computes can be performed in a single pass. For example, ((x+y) x 3)>>2 can be performed by setting the PVP_ACU_FACTOR register to =3 and by setting the PVP_ACU_SHIFT register to =2.

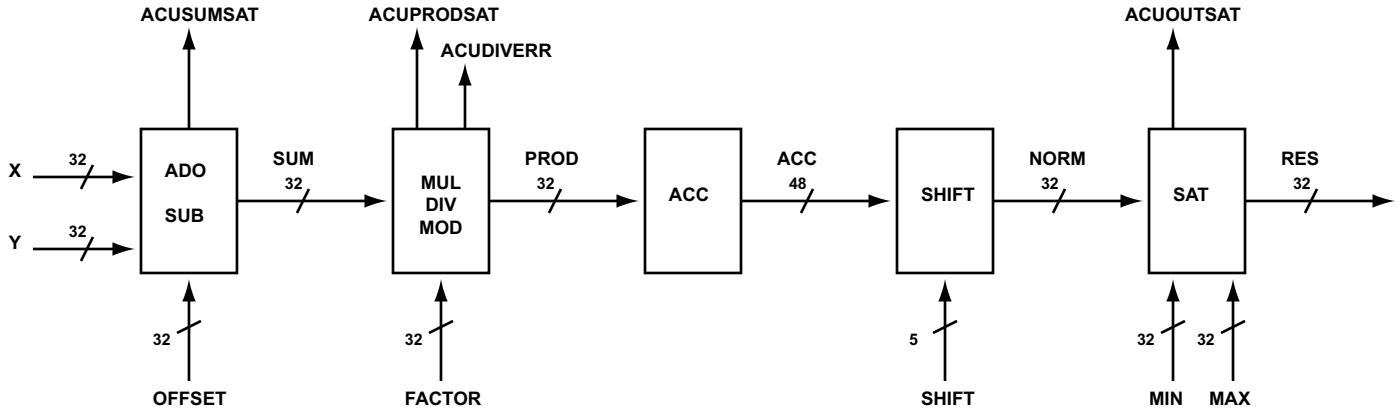


Figure 30-19: ACU Overview

For combinations of computes, the individual operations need to be in the order shown above. For example ((x>>2) x 3) +y) cannot be performed in a single pass. Similarly, not every operation can be completed on every operand. The ACU block operations and operands table lists the options available by individual math sub blocks.

Table 30-36: ACU Block Operations and Operands

RESULT	OPERATIONS	OPERANDS
SUM =	+ (32-bit signed add) - (32-bit signed subtract)	X, Y Y, X X, OFFSET OFFSET, X Y, OFFSET OFFSET, Y
PROD =	x (32-bit signed multiply) / (32-bit signed divide) % (32-bit signed modulo)	X, Y Y, X X, FACTOR FACTOR, X Y, FACTOR FACTOR, Y SUM, FACTOR FACTOR, SUM
ACC =	48-bit accumulate over row 48-bit accumulate over frame	X SUM PROD

Table 30-36: ACU Block Operations and Operands (Continued)

RESULT	OPERATIONS	OPERANDS
NORM =	48-bit arithmetic right shift	X, SHIFT SUM, SHIFT PROD, SHIFT ACC, SHIFT
RES =	32-bit signed saturation	NORM, MIN, MAX

For divide-by-zero operations, the quotient is set to the enumerator's value. The remainder is set to zero. The event is reported by the PVP_STAT.ACUDIVERR status bit. The operation itself progresses normally.

The intermediate results SUM and PROD are 32-bit values. If the result exceeds 32-bit range, the data is implicitly saturated to 0x7FFFFFFF for positive results and to 0x80000000 for negative results. The PVP_STAT.ACUSUMSAT, and PVP_STAT.ACUPRODSAT status bits report the saturation events.

If accumulation is active (PVP_ACU_CTL.SFTINP =3), the ACU outputs only one value every video row (PVP_ACU_CTL.ACCFRAME =0) or every frame (PVP_ACU_CTL.ACCFRAME =1). With every new row or frame, the accumulator is cleared automatically. The 32-bit input values or the intermediate 32-bit SUM or PROD results are summed up into the 48-bit accumulator. If intermediate accumulation results exceed 48 bits before the end of row/frame the accumulator silently overflows. It neither saturates nor generates a status event.

The SHIFT and the SAT operations are not optional. Their functionality can be bypassed by leaving the default zero value in the PVP_ACU_SHIFT register (no shift) and by setting the clipping values in the PVP_ACU_MIN register to 0x80000000 and in the PVP_ACU_MAX register to 0x7FFFFFFF. Saturation events due to MAX/MIN register method are reported by the PVP_ILAT.ACUOUTSAT status bit.

The ACU can receive data from either input formatter or any convolution block. Both inputs can be individually configured. Care is required to properly time input data so that both inputs match. The ACU block connectivity table and figure show the input and output options for this PVP block. For a graphical overview of all PVP block interconnections, see *Configuring Pipe Structure*.

Table 30-37: ACU Block Connectivity

ACU Block I/O	Data Format	PVP Block Connect Selections	Port Connect
Input 0	s32	IPF0 IPF1 CNV0 CNV1 CNV2 CNV3	0, 1, 2 0, 1, 2 0 0 0 0

Table 30-37: ACU Block Connectivity (Continued)

ACU Block I/O	Data Format	PVP Block Connect Selections	Port Connect
Input 1	s32	IPF0 IPF1 CNV0 CNV1 CNV2 CNV3	0, 1, 2 0, 1, 2 0 0 0 0
Output 0	s32	PEC, THC0, THC1, IIM0, IIM1, OPF0, OPF1, OPF2, OPF3	n/a
Output 1	n/a	n/a	n/a
Output 2	n/a	n/a	n/a

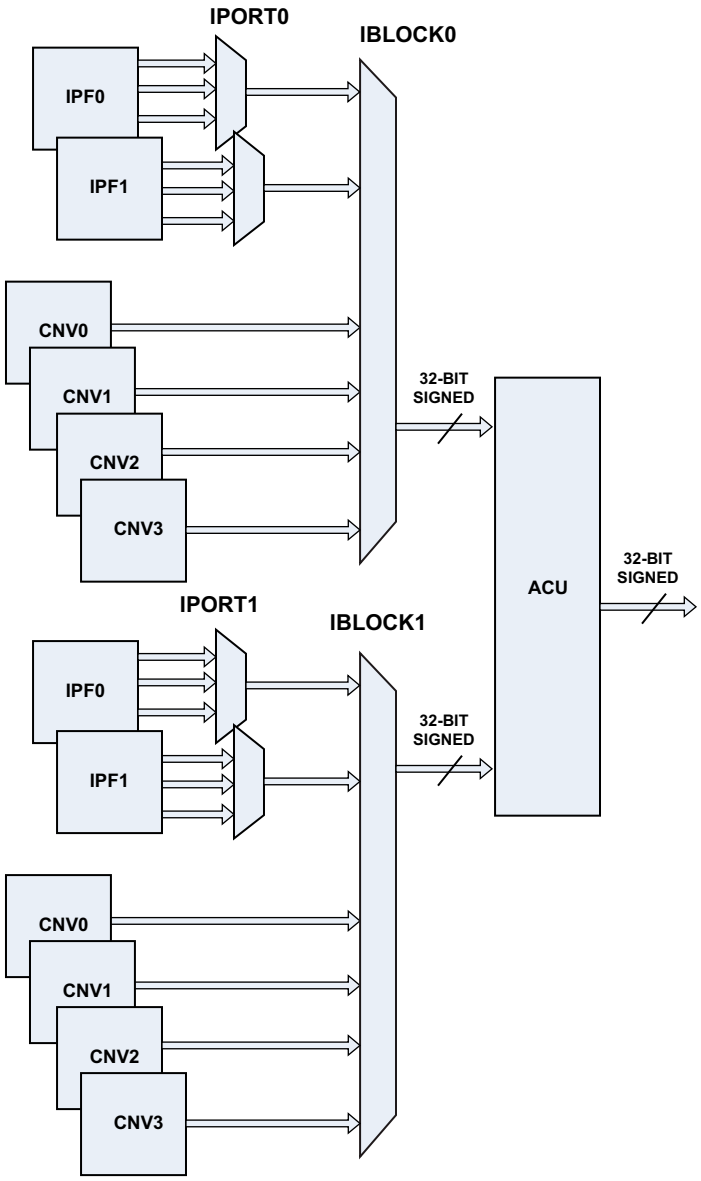


Figure 30-20: ACU Connectivity

If the X input does not influence the ACU output based on the PVP_ACU_CTL settings, the PVP_ACU_CFG.IBLOCK0 bit must be set =0. Similarly, if Y is not used in the equation, the PVP_ACU_CFG.IBLOCK1 bit must be =0.

Input values can be doubled using the adder, if X and Y inputs sense that same data. Similarly, input values can be squared. The intermediate SUM result, cannot be squared.

The ACU block is useful in memory pipe mode when the supported operation needs to apply to all elements in an array. If one operand is a constant value (as in $res_i = x_i \%c$), the constant *c* value is best written into the constant register, (the PVP_ACU_FACTOR register in the example). Input 0 (x) is typically connected to port 0 of IPF1. If, the operation is applied to elements of two arrays (as in $res_i = x_i / y_i$), input

0 (x) typically connects to port 1 of IPF1 and input 1 (y) connects to port 2. The IPF expects the input data in interleaved manner, (x0, y0, x1, y1, x2, y2, and so on) If data is not already stored this way in memory, the 2-dimensional feature of the DMA can perform the interleaving.

Pixel Edge Classifier (PEC)

The PEC block processes the edge information, enhancing vision processing. The PEC supports two modes of operation—1st derivative mode (PEC-1) and 2nd derivative mode (PEC-2). These modes are mutually exclusive, and the PEC block can operate in only one mode at a time. The PEC block takes 32-bit input data and drives up to three streams of output data.

The PEC block supports back-to-back frame size change in output from CNV block. The PEC only supports even sized frames and supports incoming frames with a decimation ratio of up to 256 (1, 2, 4, 8, 16, 32, 64, 128 and 256). The smallest frame size that the PEC can support is 16x16. Both PEC-1 mode and PEC-2 mode output zeros for pixels on the frame boundaries. The PEC mode cannot change from frame to frame. The block has to be disabled and enabled to change mode.

PEC-1 mode supports the subsequent stages after the Sobel filters for the first derivative Canny edge detection algorithm. The primary features of PEC-1 are:

- Edge thinning feature using non-maximum suppression
- Streaking elimination using hysteresis; 3-level threshold applied on the magnitude of the gradient

In PEC-1 mode, the PEC block expects the polar magnitude data on the lower 16 bits and expects the polar angle information on the upper 16 bits. So, the PEC block either can receive data from PMA output port-2 (which has the packed polar magnitude and polar angle) or can receive data from the IPFn ports (assuming the IPFn port drives the packed magnitude and angle data).

In PEC-2 mode, the PEC block can generate 8-connected chain codes for contour tracing using output from second derivative edge detection methods, such as Difference of Gaussian (DoG) and Laplacian of Gaussian (LoG). The primary features of PEC-2 mode are:

- Detects zero-crossing points in the second derivative filter output
- Classifies edges based on the edge direction
- Computes location of the edge with half pixel resolution (sub-pixel interpolation)

In PEC-2 mode, the block expects 16-bit signed input data. Therefore, the PEC block can interface with either the IPF ports, the ACU block (32-bit signed, but saturated to 16-bits) or any of the CNV blocks (32-bit signed, but saturated to 16-bits). In this mode the PEC only operates on the lower 16 bits of the 32-bit signed input. The PEC block connectivity table and figure show the high level connectivity of the PEC block. For a graphical overview of all PVP block interconnections, see *Configuring Pipe Structure*.

Table 30-38: PEC Block Connectivity

PEC Block I/O	Data Format	PVP Block Connect Selections	Port Connect
Input 0	s32	IPF0 IPF1 PMA CNV0 CNV1 CNV2 CNV3 ACU	0, 1, 2 0, 1, 2 2 0 0 0 0 0
Input 1	n/a	n/a	n/a
Output 0	u8 (encoded result)	THC0, THC1, IIM0, IIM1. OPF0, OPF1, OPF2	n/a
Output 1	u32 (decoded result)		
Output 2	u32 (angle only result)		

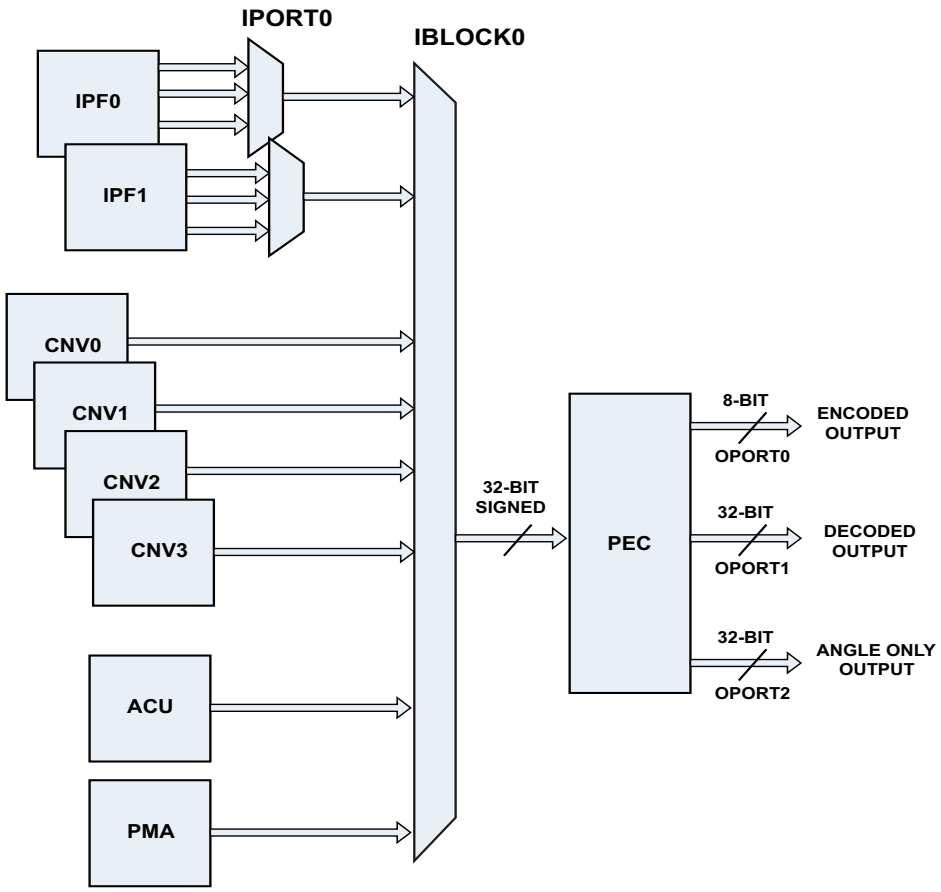


Figure 30-21: Pixel Edge Classifier Block Connectivity

PEC 1st Derivative Mode (PEC-1)

This mode is selected by setting the PVP_PEC_CTL.MODE bit to 0. The PEC-1 mode works on 16-bit gradient magnitude and 5-bit gradient angle from the PMA block. The functions in this mode are Angle Mapping, Non-Maximum Suppression, Hysteresis Thresholding, and Output Format.

Angle Mapping

The angle is rounded to one of four angles representing the vertical, the horizontal, and the two diagonal directions. The 5-bit angle bin output of the PMA is mapped to the four directions as shown in the PEC-1 angle figure.

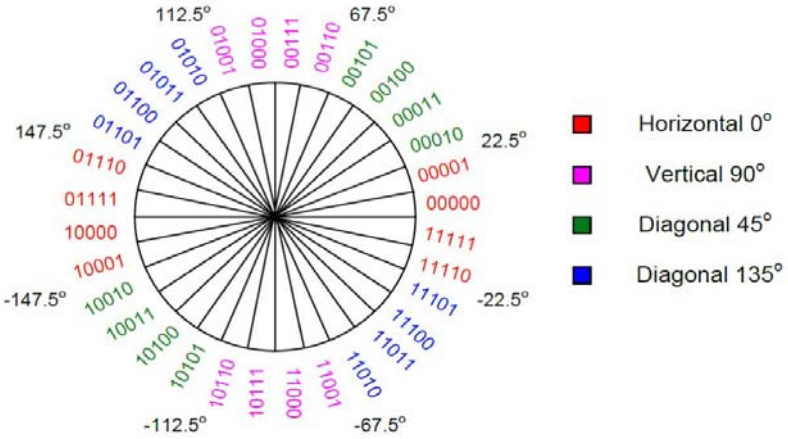


Figure 30-22: PMA Angle Bin Mapping to PEC-1 Angle

Non-Maximum Suppression

PEC-1 performs a search with the input magnitude gradient to determine if the magnitude assumes a local maximum in the gradient direction so that thin edges are produced. This operation is referred to as *Non-Maximum Suppression*.

The PEC-1 mode works on the 3x3 window with the center pixel as the pixel of interest and computes whether the pixel is a local maximum or not a maximum. Consider a 3x3 window of magnitude gradients as shown.

P11	**P12**	**P13**
P21	**P22**	**P23**
P31	**P32**	**P33**

The pixel of interest (P22) in this example is *maxima (M)* if non-maximum suppression finds:

- P22's rounded angle is 0 degrees, and P22's intensity is greater than the intensity in the west (P21) and is greater than the intensity in the east (P23) directions.
- P22's rounded angle is 90 degrees, and P22's intensity is greater than the intensity in the north (P12) and is greater than the intensity in the south (P32) directions.
- P22's rounded angle is 135 degrees, and P22's intensity is greater than the intensity in the north west (P11) and is greater than the intensity in the south east (P33) directions.
- P22's rounded angle is 45 degrees, and P22's intensity is greater than the intensity in the north east (P13) and is greater than the intensity in the south west (P31) directions.

In cases where the gradient magnitude is the same for more than one pixel along the gradient direction "equality" (E) information is also provided by PEC-1. Edge-tracing software uses M and E to find the exact location of the edge. Consider the same 3x3 window as before.

The pixel of interest (P22) is *equality (E)* if non-maximum suppression finds:

- P22's rounded angle is zero degrees, and P22's intensity is greater than or equal to the intensity in the west (P21) and is greater than or equal to the intensity in the east (P23) directions.
- P22's rounded angle is 90 degrees, and P22's intensity is greater than or equal to the intensity in the north (P12) and is greater than or equal to the intensity in the south (P32) directions.
- P22's rounded angle is 135 degrees, and P22's intensity is greater than or equal to the intensity in the north west (P11) and is greater than or equal to the intensity in the south east (P33) directions.
- P22's rounded angle is 45 degrees, and P22's intensity is greater than or equal to the intensity in the north east (P13) and is greater than or equal to the intensity in the south west (P31) directions.

Hysteresis Threshold Application

PEC-1 mode's Canny algorithm applies a *hysteresis* threshold. When a single threshold value is used, the fluctuations of the gradient magnitude (due to noise) above and below the threshold value results in pixels being classified as edges and non-edges. In such cases, the edge line appears broken. This phenomenon is commonly referred to as *streaking*. PEC-1 mode eliminates streaking by comparing the magnitude against two threshold registers, `PVP_PEC_D1TH0` and `PVP_PEC_D1TH1`.

If the magnitude lies below `PVP_PEC_D1TH0` (TL), the pixel is called *no edge*. If the magnitude lies above `PVP_PEC_D1TH1` (TH), the pixel is termed as *strong edge*. If the magnitude lies between TL and TH, the

pixel is termed as *weak edge*. The post-processing software can connect the weak edge to strong edge to create a unbroken edge line. Note that TH must be programmed higher than TL for hysteresis.

Output Format

The PEC-1 mode produces differently formatted outputs on PEC block output ports 0, 1, and 2.

PEC-1 Output on Port 0

This output is primarily used for post-processing in software. Encoded 8-bit output is produced for every input pixel as shown.

The 5-bit angle bin from the PMA or IPFn blocks is passed on to the 5 MSBs of the PEC-1 output if the pixel magnitude is greater than or equal to TL. If the pixel magnitude is less than TL, the 5 MSBs are suppressed to zeros. PEC-1 combines the maxima (M), equality (E), and hysteresis threshold value to produce a 3-bit code output (forming the 3 LSBs of the PEC-1 output) whose bits are defined in the PEC-1 code mapping table.

Table 30-39: PEC-1 Code Mapping

LSB of PEC Output	Description
000	mag < TL
001	mag >= TL, mag < TH, M=0, E=0
010	mag >= TL, mag < TH, M=0, E=1
011	mag >= TL, mag < TH, M=1, E=1
100	reserved
101	mag >= TH, M=0, E=0
110	mag >= TH, M=0, E=1
111	mag >= TH, M=1, E=1

PEC-1 Output on Port 1

This output is 32 bits wide and is used primarily for the IIM block to accumulate the edge statistics. The output can be either 8-bit bin size encoded or 16-bit bin size encoded, based on the setting of the PVP_PEC_

CTL.OSIZE control bit. If bit is 0 (indicating 8-bit bin size), the 32-bit output represents four angle bins, each having a strong/weak edge information as shown in the 8-bit bin size encoding table.

Table 30-40: Output Encoding for 8-bit Bin Size

Angle	Edge	Encoding
Mag < TL	No edge	00000000_00000000_00000000_00000000
0° or 180°	Weak	00000000_00000000_00000000_00000001
0° or 180°	Strong	00000000_00000000_00000000_00000010
45° or 225°	Weak	00000000_00000000_00000001_00000000
45° or 225°	Strong	00000000_00000000_00000010_00000000
90° or 270°	Weak	00000000_00000001_00000000_00000000
90° or 270°	Strong	00000000_00000010_00000000_00000000
135° or 315°	Weak	00000001_00000000_00000000_00000000
135° or 315°	Strong	00000010_00000000_00000000_00000000

If the PVP_PEC_CTL.OSIZE bit is 1 (indicating 16-bit bin size), the 32-bit output represents two angle bins, each having a strong/weak edge information as shown in the 16-bit bin size encoding table.

Table 30-41: Output Encoding for 16-bit Bin Size

Angle	Edge	Encoding
Mag < TL	No edge	0000000000000000_0000000000000000
0° or 180° or 90° or 270°	Weak	0000000000000000_0000000000000001
0° or 180° or 90° or 270°	Strong	0000000000000000_0000000000000010
45° or 225° or 135° or 315°	Weak	0000000000000001_0000000000000000
45° or 225° or 135° or 315°	Strong	0000000000000010_0000000000000000

The strong or weak edge is determined from the threshold values programmed in the PVP_PEC_D1TH0 (TL) and PVP_PEC_D1TH1 (TH) registers. The PVP_PEC_CTL.IGNTH1 bit supports masking the differences between strong and weak edges. If this bit is 0, weak and strong edges are treated differently, and the output is as per the 8- or 16-bit bin output encoding tables. If this bit is 1, all the edges are considered weak (irrespective of the TH value), and the output is encoded to represent a weak edge as per the 8- or 16-bit bin output encoding tables.

PEC-1 Output on Port 2

This output is 5 bits wide and is used primarily for generating the angle histograms. The 5-bit angle bin from the PMA or IPFn blocks is passed on to this port if the pixel magnitude is greater than or equal to the

PVP_PEC_D1TH0 register. If the center pixel magnitude is less than PVP_PEC_D1TH0, this output is suppressed to zeros.

PEC 2nd Derivative Mode (PEC-2)

This mode is selected by setting the PVP_PEC_CTL.MODE bit to 1. The PEC-2 mode works on 16-bit signed data from the second derivative filter output, such as DoG and LoG. The functions in this mode are Zero-Crossing Detection, Sub-Pixel Interpolation, Angle Classification, and Output Format.

Zero-Crossing Detection

Edges lie on the zero crossing points (-ve to +ve and +ve to -ve transition points) in the 2nd derivative filter output. The PEC-2 mode scans through the filtered image, determines zero crossings in horizontal (zH) and vertical (zV) directions, and assigns pre-defined 3-bit zero crossing codes. The computations for the zero crossing detection algorithm are shown in the zero crossing codes table.

Whenever there is a +ve to -ve or -ve to +ve in the filtered image, the pixel with +ve value in the image is considered to be a valid zero-crossing point. Because a 2nd derivative filtered image is more prone to errors due to noise, the PEC block uses a user-defined threshold to remove the noise during zero crossing detection. Zero crossings are considered valid only when:

- There is a sign change, and
- The difference between the 2nd derivative values of these two pixels is greater than the threshold in the PVP_PEC_D2TH0 register

When the pixel of interest is has a value of zero, the pixel is considered as a valid zero-crossing point if the two pixels on the left and right (for horizontal and top, bottom for vertical) are values with opposite signs, (+ 0 -) or (- 0 +). The MSB of the 3-bit code is set if the difference between the pixels is greater than the threshold in the PVP_PEC_D2TH1 register.

The zero crossing codes table shows the different combinations of 2nd derivative image values for the pixel of interest, the pixel on the left and the pixel on the right, and the corresponding 3-bit zero-crossing codes for horizontal direction (zH). The vertical zero-crossings (zV) are obtained by replacing left pixel with the top pixel and by replacing the right pixel with the bottom pixel in the zero crossing codes table.

Table 30-42: Zero Crossing Codes

Left	Center	Right	Computation	zH (3-bit)
Don't care	-ve	Don't care	n/a	000
+ve / 0	+ve	+ve / 0	n/a	000
-ve / 0	0	0 / -ve	n/a	000

Table 30-42: Zero Crossing Codes (Continued)

Left	Center	Right	Computation	zH (3-bit)

-ve	+ve	+ve / 0	if (center-left >= Threshold_weak) & (center-left < Threshold_strong)
			else if (center-left >= Threshold_strong)
			else
				010
				110
				000

-ve	+ve	-ve	if (center-left >= Threshold_weak) & (center-left < Threshold_strong)
			else if (center-left >= Threshold_strong)
			else if (center-right >= Threshold_weak) & (center-right < Threshold_strong)
			else if (center-right >= Threshold_strong)
			else
				010
				110
				001
				101
				000

+ve / 0	+ve	-ve	if (center-right >= Threshold_weak) & (center-right < Threshold_strong)
			else if (center-right >= Threshold_strong)
			else
				001
				101
				000

+ve / 0	0	+ve / 0		000

-ve	0	+ve	if (right-left >= 2×Threshold_weak) & (right-left < 2×Threshold_strong)
			else if (right-left >= 2×Threshold_strong)
			else
				010
				110
				000

+ve	0	-ve	if (left-right >= 2×Threshold_weak) & (left-right < 2×Threshold_strong)
			else if (left-right >= 2×Threshold_strong)
			else
				001
				101
				000

-ve	0	-ve	n/a	000

Sub-Pixel Interpolation

PEC-2 mode performs a linear interpolation of the filtered image at the zero crossings to determine a more precise location of the zero crossings. The values of (left, center, right) for horizontal direction and (top, center, bottom) for vertical direction are compared for relative differences. Additionally, PEC-2 computes edge location with half pixel resolution.

The sub-pixel computation table explains the different zero crossing codes, the pixel value sequences, and the corresponding sub-pixel values (HSub, 2-bits). The vertical sub-pixel values (VSub, 2-bits) can be computed using the same procedure, replacing left pixel with top pixel, right pixel with bottom pixel and the 2 LSBs of zH with zV.

Table 30-43: Sub-Pixel Computation

zH (2-bits)	Left	Center	Right	Computation	Sub-pixel HSub (2-bits)
00	Don't care	Don't care	Don't care	-	00
10/01	Don't care	0	Don't care	-	00
10	-	+	Don't care	if (-left) <= center else	10 00
01	Don't care	+	-	if (-right) <= center else	01 00

Angle Classification

The PEC-2 mode's filtering of an image does not convey any information about the orientation of the edges in the image. The PEC angle classification unit in PEC-2 mode extracts the angle information of the edges from the zero crossing codes of pixels in the 3x3 window. The classifier uses only the two lower bits of zH and zV, (both weak and strong zero crossings) and classifies the edges into eight basic directions (0°, 45°, 90°, 135°, 180°, 225°, 270° and 315°) if the `PVP_PEC_CTL.ZCRSS` bit is 0. The classifier outputs a 4-bit angle index as described in the angle classification table.

The angle index (shown in the angle classification table) is considered a strong edge only if all the involved zH or zV values used in the computation are strong (or have their MSB = 1). If the `PVP_PEC_CTL.IGNTH1` bit is set, the strong/weak zero-crossing information (MSB of zH and zV) is ignored by the classifier.

Table 30-44: Angle Classification

Priority	Zero Crossing Code Combination	Class	Angle Index (4-bit)
1	zH(r,c) = 00 & zV(r,c) = 00	no edge	0000
2	Not matching any of the combination 4 to 11.	no match	0001
3	Matching more than one combination in 4 to 11.	multiple match	0010

Table 30-44: Angle Classification (Continued)

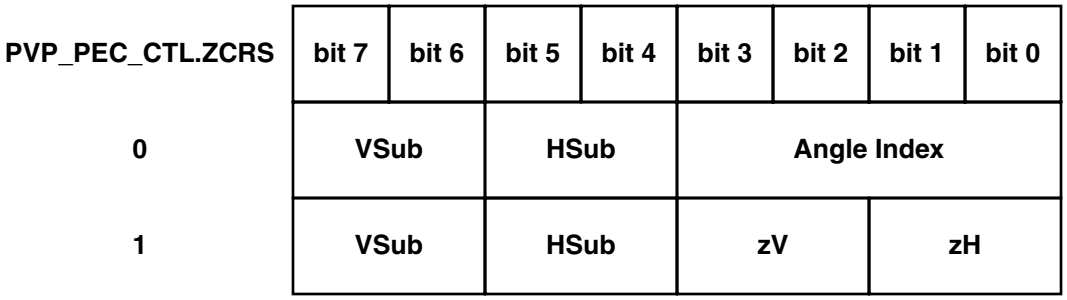
Priority	Zero Crossing Code Combination	Class	Angle Index (4-bit)
4	zV(r,c) = 01 & zV(r,c+1) = 01	0°	1000
5	(zH(r,c) = 01 || zV(r,c) = 01) & (zH(r-1,c+1) = 01 || zV(r-1,c+1) = 01)	45°	1001
6	zH(r,c) = 01 & zH(r-1,c) = 01	90°	1010
7	(zH(r-1,c-1) = 01 || zV(r-1,c-1) = 10)	135°	1011
8	zV(r,c) = 10 & zV(r,c-1) = 10	180°	1100
9	(zH(r,c) = 10 || zV(r,c) = 10) & (zH(r+1,c-1) = 10 || zV(r+1,c-1) = 10)	225°	1101
10	zH(r,c) = 10 & zH(r+1,c) = 10	270°	1110
11	(zH(r,c) = 10 || zV(r,c) = 01) & (zH(r+1,c+1) = 10 || zV(r+1,c+1) = 01)	315°	1111

Output Format

The PEC-2 mode produces differently formatted outputs on PEC block output ports 0, 1, and 2.

PEC-2 Output on Port 0

This output is used primarily for post-processing in software. The 8-bit encoded output is produced for every input pixel as shown.



PEC-2 Output on Port 1

This output is 32 bits wide and is used primarily for the IIM block to accumulate the edge statistics. The output is not valid when the PVP_PEC_CTL.ZCRSS bit =1, because the edges are not classified in this mode. PEC-2 produces a 32-bit output which is exactly similar to the output generated by PEC-1. The angle is computed as shown in the Angle Classification table. For priority 1, 2, and 3, the output is considered as no edge, and all zeros are driven on the port.

PEC-2 Output on Port 2

This output is 5 bits wide and is used primarily for the THCn block to generate the angle histograms. PEC-2 drives the 4-bit angle index described in the Angle Classification table. Zero is driven on the MSB of the

5-bit port. For priority 1, 2, and 3, the output is considered as no edge, and all zeros are driven on the port. The output is not valid when the PVP_PEC_CTL.ZCRSS bit =1, because the edges are not classified in this mode.

Integral Image Block (IIMn)

The integral image blocks (IIMn) calculate a 2-dimensional integral over the input frame and outputs the summed area table (SAT). Alternatively, the IIMn blocks can generate a rotated SAT (RSAT) or can integrate in horizontal dimension only (integral row mode). The operating mode is controlled by the PVP_IIMn_CTL.MODE bit field.

IIMn blocks can receive inputs from a number of PVP blocks. The IMMn blocks can build the integral on raw data inputs when receiving data from the input formatters. Input data can optionally be filtered by convolutions. Variances can be calculated by squaring the inputs by the ACU before building the integral. If connected to the PMA blocks, integral of edge magnitude and gradients can be built. Histogram of gradients can be done by sensing port 1 of the PEC block. If connected to the THCn blocks, the integral of a binary edge map can be generated, or data is clipped up front by the THCn blocks. The input to each IIMn block is selectable from the output of the PVP blocks, as shown in the following tables. For a graphical overview of PVP block connectivity, see *Configuring Pipe Structure*.

Table 30-45: IIM0 and IIM1 Block Connectivity

IIM0/1 Block I/O	Data Format	PVP Block Connect Selections	Port Connect
Input 0	1x s32, 1x u32, 2x s16, 2x u16, 4x s8, 4x u8	IPF0 IPF1 CNV0 CNV1 CNV2 CNV3 ACU PEC PMA THC0 THC1	0, 1, 2 0, 1, 2 0 0 0 0 0 1 0, 1 0 0
Input 1	n/a	n/a	n/a
Output 0	1x s32, 1x u32, 2x s16, 2x u16, 4x s8, 4x u8	OPF0, OPF1, OPF2, OPF3	n/a
Output 1	n/a	n/a	n/a

Table 30-45: IIM0 and IIM1 Block Connectivity (Continued)

IIM0/1 Block I/O	Data Format	PVP Block Connect Selections	Port Connect
Output 2	n/a	n/a	n/a

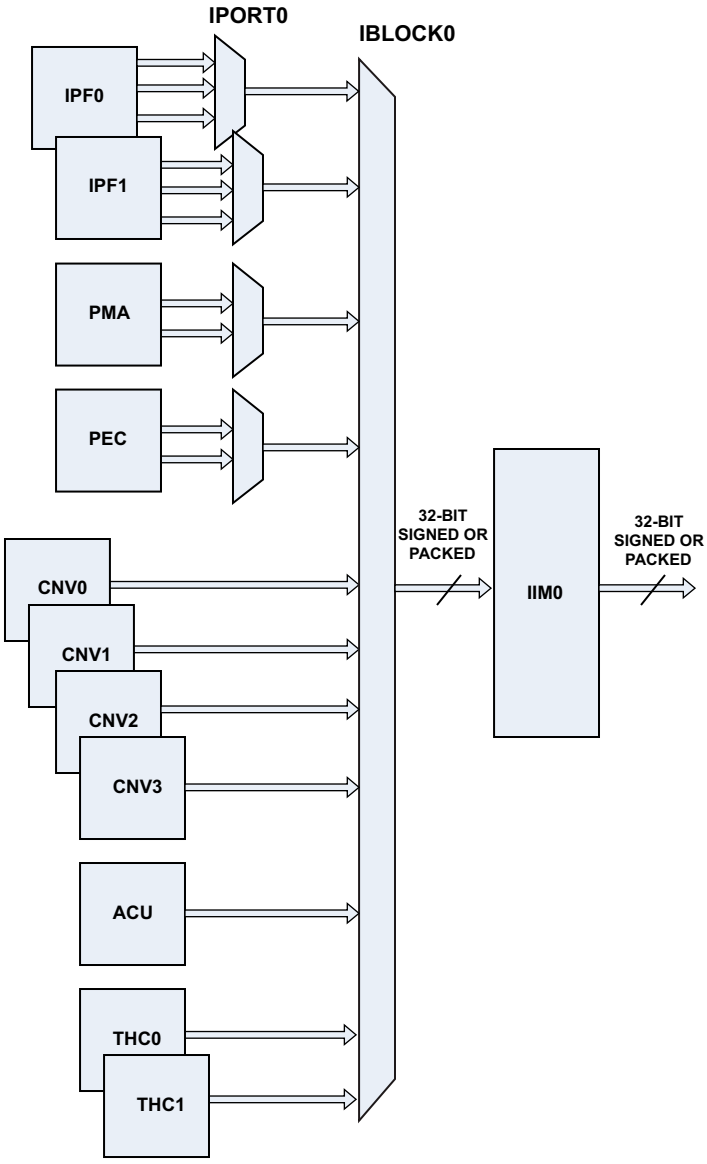


Figure 30-23: IIMn Block Connectivity

IIMn Data Types

IIMn blocks feature a 33-bit adder to operate on 32-bit data. The 33rd bit is used for overflow management. If the adder overflows, it does not saturate but continues normal operation. Software may post-analyze the

integral map to track overflows. The occurrence of an overflow depends on the input data format. For 10-bit input data, the integral counter should never overflow over the region or even the full frame.

The occurrence of overflows can be controlled by reducing the resolution of the incoming data. The PVP_IIMn_CTL.SHIFT control field instructs the IIMn block to arithmetically right shift input data. The shift may be from 0 to 31 bit positions to the right.

The IIMn blocks can operate on signed and unsigned data. There is no control bit to distinguish between signed and unsigned mode. Rather, there are two different sets of status reports. An unsigned-overflow event is reported by the PVP_STAT.IIM0UOVF bit and by the PVP_STAT.IIM1UOVF bit. When flagged, the PVP_IIMn_UOVF_STAT register latches the coordinates of the data that caused the overflow. Similarly, the PVP_STAT.IIM0SOVF bit and PVP_STAT.IIM1SOVF bit report signed overflows, and the PVP_IIMn_SOVF_STAT register latches the coordinates.

The adder can be configured to operate in dual 16-bit mode or to operate in quad 8-bit mode. This selection is controlled by the PVP_IIMn_CTL.WIDTH field. In these modes, each IIMn block can perform dual- or quad-integral operations concurrently (at the risk increasing the likelihood of overflows). The overflow reporting mechanism works the same it works in 32-bit mode where the mechanism monitors overflows in the topmost word or byte. In dual 16-bit mode or in quad 8-bit mode, the input shifter performs an unsigned right-shift operation, rather than performing an arithmetical operation (as is done in 3-bit mode).

The quad 8-bit mode is most useful when the integral block connects to port 1 of the PEC block. In this case, the IIMn block generates the integral over the 1st or 2nd derivative gradients. Dual 16-bit mode might connect to port 2 of the PMA block. In this case, the lower part of the adder builds the integral over the magnitude, while the upper half integrates the angle. Alternatively, port 2 of the PMA block might be routed through one of the THCn blocks for quantization of the magnitude.

IIMn Bandwidth Usage

In many cases the IIMn blocks receive input data with a relatively low bit count. Regardless of the input size, the output is always 32 bits wide. If not used with care, the IIMn blocks can be significant consumers of system memory bandwidth. When 32-bit resolution is not required for the IIMn block output, the connected output formatter may be configured to only take the upper or the lower 16 bits of the result. To take the lower bits, set PVP_OPFn_CTL.ISIZE =1, and set PVP_OPFn_CTL.IUP16 =0. To take the upper bits, set PVP_OPFn_CTL.ISIZE =1, and set PVP_OPFn_CTL.IUP16 =1. These bit settings---together with the PVP_IIMn_CTL.SHIFT setting---permit the weight (or data throughput bandwidth) of the integral can be controlled.

Another way to reduce the bandwidth used by the integral blocks is by scaling the IIMn output. The PVP_IIMn_SCALE register can individually enable sub-sampling of rows or columns on the output in power-of-2 steps as follows:

- If PVP_IIMn_SCALE.VSCL = 0, no vertical scaling is performed, and all rows are output.
- If PVP_IIMn_SCALE.VSCL =1, the IIMn outputs only every other row.
- If PVP_IIMn_SCALE.VSCL =3, the IIMn outputs only every 4th row.

- If PVP_IIMn_SCALE.HSCL =7, the IIMn outputs only every 8th column.

IIMn Integral Row (IR) Mode

In integral row mode, the IIMn blocks accumulate the input values over video rows. Unlike the ACU's accumulation mode, IIMn integral row mode outputs the integral every pixel. For every input data ($Inp_{I,J}$), the IIMn blocks output is as shown in the equation.

$$IROW_{I,J} = \sum_{i=0}^{I} Inp_{i,J} \qquad IROW_{I,J} = Inp_{I,J} + IROW_{I-1,J}$$

The initial value of the integral is assumed to be zero. So, $IROW_{0,J} = Inp_{0,J}$. In the IROW output map, each value contains the accumulated input values from the respective row's origin to the actual position, as shown in the Illustration of Integral Row Mode figure.

2	0	1	1	0	2	3	0
0	0	1	0	3	0	0	0
0	2	3	1	0	1	0	0
0	0	1	2	0	1	0	5
1	0	0	1	0	0	2	0
1	0	1	0	0	1	1	0

Input Data:
0+2+3+1+0+1+0 = 7

2	2	3	4	4	6	9	9
0	0	1	1	4	4	4	4
0	2	5	6	6	7	7	7
0	0	1	3	3	4	4	9
1	1	1	2	2	2	4	4
1	1	2	2	2	3	4	4

Integral Row Mode

Figure 30-24: Illustration of Integral Row Mode

IIMn Summed Area Table (SAT) Mode

In 2-dimensional summed area table (SAT) mode, the IIMn block performs the summation (see equation) for every pixel.

$$SAT_{I,J} = \sum_{j=0}^{J} \sum_{i=0}^{I} Inp_{i,j} \qquad SAT_{I,J} = Inp_{I,J} + SAT_{I-1,J} + SAT_{I,J-1} - SAT_{I-1,J-1}$$

As with integral row mode, the initial values of the integral are assumed to be zero—$SAT_{-1,J} = 0$ and $SAT_{I,-1} = 0$. In the SAT output map, each value contains the accumulated input values from the frame's origin to the actual position, as shown in the Illustration of SAT Mode figure.

2	0	1	1	0	2	3	0
0	0	1	0	3	0	0	0
0	2	3	1	0	1	0	0
0	0	1	2	0	1	0	5
1	0	0	1	0	0	2	0
1	0	1	0	0	1	1	0

Input Data:
2+0+1+1+0+0+1+0+0+2+3+1 = 11

2	2	3	4	4	6	9	9
2	2	4	5	8	10	13	13
2	4	9	11	14	17	20	20
2	4	10	14	17	21	24	29
3	5	11	16	19	23	28	33
4	6	13	18	21	26	32	37

SAT Mode

Figure 30-25: Illustration of SAT Mode

IIMn Rotated Summed Area Table (RSAT) Mode

In rotated summed area table (RSAT) mode, the IIMn block applies a -45° rotated view to the input coordinates. Each element of the RSAT output map contains the integral of all elements in the pyramid headed by the actual position. The IIMn block process according to the following equations.

First Column:

$$RSAT_{0,J} = Inp_{0,J} + Inp_{0,J-1} + RSAT_{1,J-1}$$

Last Column:

$$RSAT_{I,J} = Inp_{I,J} + Inp_{I,J-1} + RSAT_{I-1,J-1}$$

All others:

$$RSAT_{I,J} = Inp_{I,J} + Inp_{I,J-1} + RSAT_{I+1,J-1} + RSAT_{I-1,J-1} - RSAT_{I,J-2}$$

All values outside of the region are defined as zero—$Inp_{I,-1} = RSAT_{I,-1} = RSAT_{I,-2} = 0$, as shown in the Illustration of RSAT Mode figure.

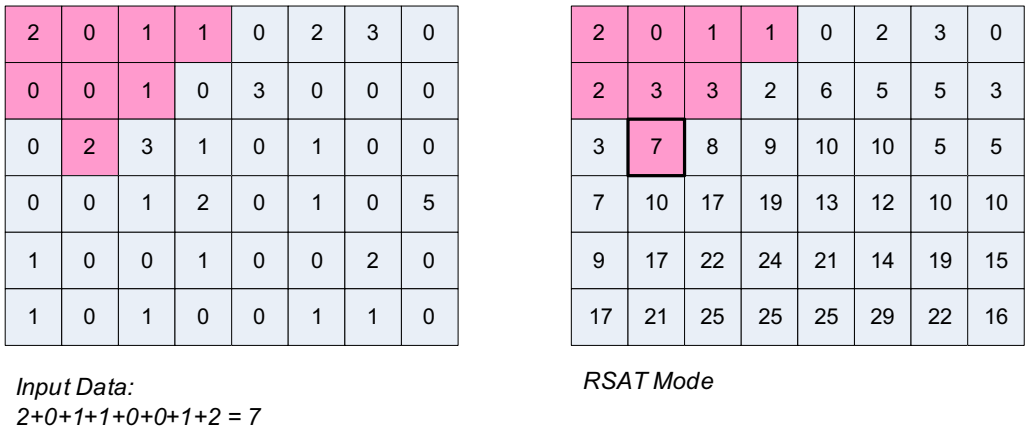


Figure 30-26: Illustration of RSAT Mode

IIMn SAT/RSAT Map Usage

The IIMn blocks generate an integral in which each position contains the sum of all input values between the actual position and the frame origin. This integral is useful, but what is more useful is---with only a small number of operations---post-processing software can determine the integral value of any rectangular region of the frame. For regular rectangles, the SAT output can be used. For 45° rotated rectangles, the RSAT output applies. This operation is illustrated in the Using SAT Maps figure.

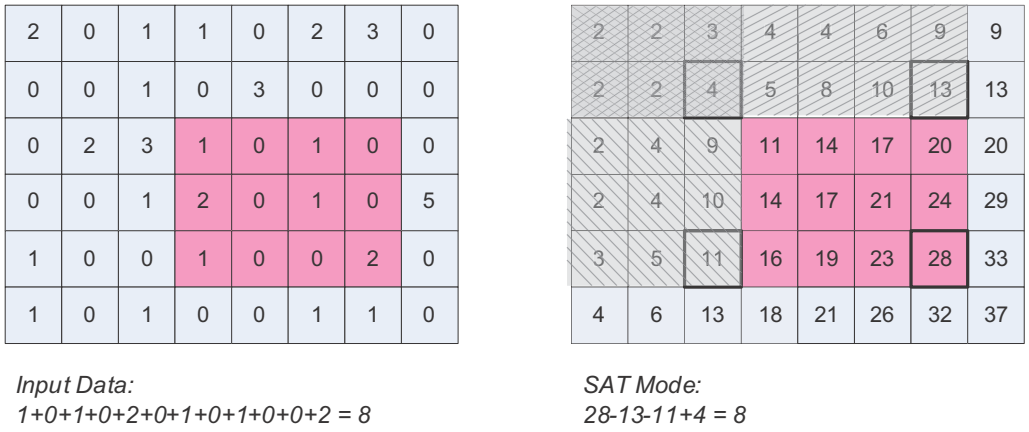


Figure 30-27: Using SAT Maps

Rather than summing up all original values of a rectangular region of the frame, software can perform simple math on the SAT output of the IIMn blocks. If the top-left corner of the rectangle was defined by the coordinates I0 and J0 *and* the bottom-right corner by I1 and J1, the integral value of the region is defined by the equation.

$$\iint_{\mathrm{Re}\,gion} Inp_{i,j} = \sum_{j=J_0}^{J_1}\sum_{i=I_0}^{J_1} Inp_{i,j} = SAT_{I_1,J_1} - SAT_{I_1,J_{0-1}} - SAT_{I_{0-1},J} + SAT_{I_{0-1},J_{0-1}}$$

Similarly, the RSAT map can be used to quickly determine the integral of rotated regions as shown in the Using the RSAT Map figure.

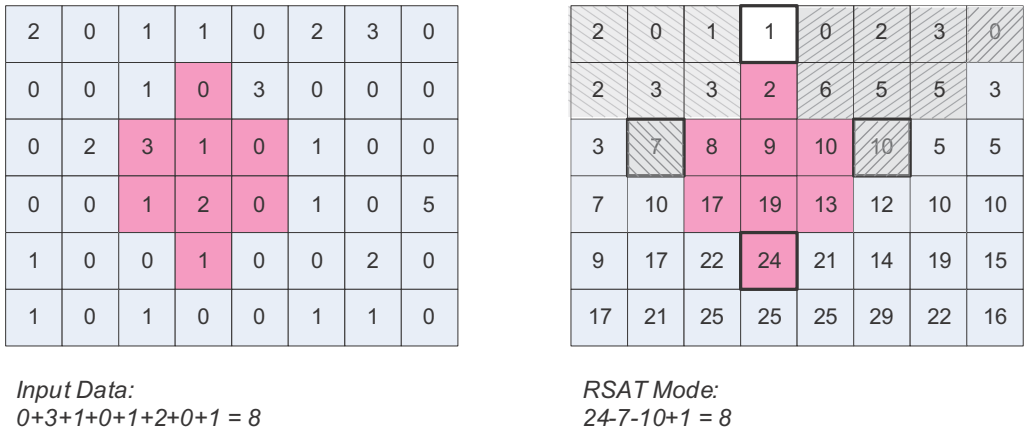


Figure 30-28: Using the RSAT Map

Up Down Scaler (UDS)

The up-down scaler (UDS) is an image resizing block. The UDS is the only block that cannot operate in the camera pipe. The input to UDS comes from IPF1, and the output from the UDS drives OPF3 as shown in the UDS Block Connections figure. For a graphical overview of PVP block connectivity, see *Configuring Pipe Structure*.

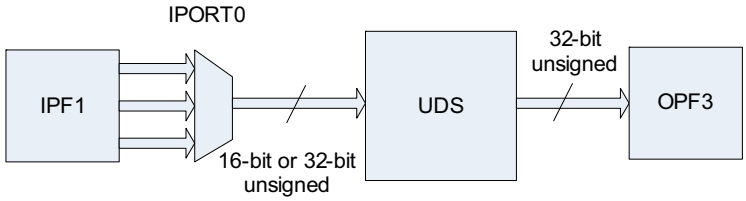


Figure 30-29: UDS Block Connections

The UDS block expects 16-bit or 32-bit unsigned input data and drives 32-bit unsigned output data. When an anti-aliasing or an averaging filter is enabled, the input must be 16 bits. Correspondingly, the output is 16 bits presented in the lower 16 bits of the 32-bit output.

The UDS Overview figure shows the data flow within the UDS block. The block performs horizontal scaling followed by vertical scaling in two independent steps. The interpolation uses bilinear arithmetic.

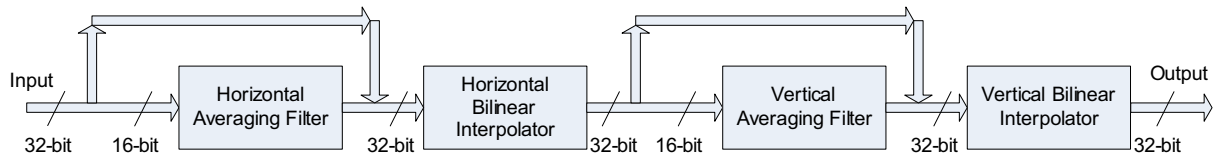


Figure 30-30: UDS Overview

The key features of the UDS block are:

- Supports input frame sizes from 10x10 to 1280x960

  The frame size is programmed in the PVP_IPFn_HCNT and PVP_IPFn_VCNT registers of IPF1.

- Supports independent scaling in both directions

  The output frame size is selectable in steps of 16. The values supported are: 16 × n × 16 × m where n,m=1,2,...,8. The output size is programmed in the PVP_UDS_OHCNT and PVP_UDS_OVCNT registers.

- Supports an optional anti-aliasing filter in the form of an averaging filter in both directions

  The filter has an option to calculate the number of taps automatically or takes a user programmable value. If the PVP_UDS_CTL.AAVG bit =1, the number of taps are selected automatically based on the input and output frame size. If the bit = 0, the number of taps must be programmed in the PVP_UDS_HAVG and PVP_UDS_VAVG registers. The UDS supports a maximum of 128 taps for horizontal direction and 64 taps for vertical direction.

- Supports an averaging filter only for 16-bit input data

  The averaging filter must be disabled for 32-bit input data by setting the PVP_UDS_CTL.AAVG bit =0, by setting the PVP_UDS_HAVG.VALUE =1, and by setting the PVP_UDS_VAVG.VALUE bit =1. The filter is bypassed when the number of taps is 1.

  The number of taps for each direction in automatic mode is calculated as:

$$N\_TAPS = \left\lceil \frac{Source\_Dimension}{Target\_Dimension} \right\rceil$$

  In user programmable mode, the number of taps is limited by the equation:

$$N\_TAPS \le \left(2 * \frac{Source\_Dimension}{Target\_Dimension} \right) - 1$$

  If an output pixel requires input pixels beyond the edges, the nearest pixel on the edge is used for computation. In effect, edge pixels are duplicated from the nearest valid pixel.

PVP Architectural Concepts

The PVP unifies two architectural concepts, the camera pipes and the memory pipe. The camera pipes serve the preprocessing requirements. Data is preprocessed on its way from the PPI input to the system memory and can be broadcast from one single video source to up to three functional branches called camera pipes. The memory pipe functions as a co-processor. This pipe loads input data from system memory using DMA and stores results back to system memory using DMA.

Camera pipes and memory pipes can operate concurrently. The pipes have separate control mechanisms, have separate data paths, and operate with independent timing. The camera and memory pipes do share

the signal processing blocks (PVP blocks). You have full flexibility to assign each of the PVP blocks to any pipe. The one exception is the up/down-scaler block (UDS), which can only operate in memory pipe mode. Each PVP block may be assigned only to one pipe at a time. But, there is support for dynamic re-configuration of the pipe assignments. For a graphical overview of all PVP block interconnections, see *Configuring Pipe Structure*.

The PVP blocks are:

- Two input formatters (IPFn, x=0, 1) supporting selection of input streams, gating pipeline processing clock, extraction of color or luminance components, windowing region of interest, separation of odd and even bit stream vectors (pixels) and frame counting. IPF0 receives input data from the video subsystem and feeds data to the camera pipes. IPF1 receives data from memory via DMA and services the Memory Pipe.
- Four convolution blocks (CNVn, x=0, 1, 2, 3) supporting 2-dimensional 5x5 convolution of data streams and constant coefficients, vector down scaling, data shifting, and data normalization.
- One polar magnitude and angle block (PMA) supporting data conversion from Cartesian form (X, Y) to a polar form (magnitude, angle).
- One pixel edge classifier (PEC) supporting edge detection in 2-dimensional data arrays using 2-dimensional 1^{st} or 2^{nd} derivation.
- One arithmetic unit (ACU) supporting 32-bit arithmetic operations (product, sum, accumulation, shift, maximum, minimum).
- Two threshold-histogram-compression cells (THCn, x=0, 1) supporting data formatting (clipping, saturation), data quantization (binning), multi-level hysteresis and data compression.
- Two integral image blocks (IIMn, x=0, 1) supporting integrals of rows and diagonals.
- One Up-down scaler (UDS) supporting interpolation and extrapolation of 2-dimensional data arrays.
- Four output formatters (OPFn, x=0, 1, 2, 3) supporting selection of output streams, data packing, and buffering (FIFO) output data streams. They collect the processing results and forward them to dedicated DMA channels.

Operating Modes

The operating modes of the PVP can be used to implement typical vision-system operations. The following sections provide operating mode examples:

- *Thresholds and Histograms*
- *Sobel with 3x3 or 5x5 Matrix Operation*
- *Sobel Output Formats*
- *Canny with PEC in 1st-Derivative Mode*

- *LoG with PEC in 2nd-Derivative Mode*
- *DoG with PEC in 2nd-Derivative Mode*
- *Integral of Input Pixels*
- *Integral of Binary Edge Map*
- *Integral of Variance*
- *Histogram of Gradients (HoG)*

Thresholds and Histograms

The THCn block can be used for a variety of threshold and histogram purposes in vision processing. The example in the computation of 16-level Histogram figure shows how the THCn block may be used to reduce the input data range to 4-bits and compute a 16-level histogram.

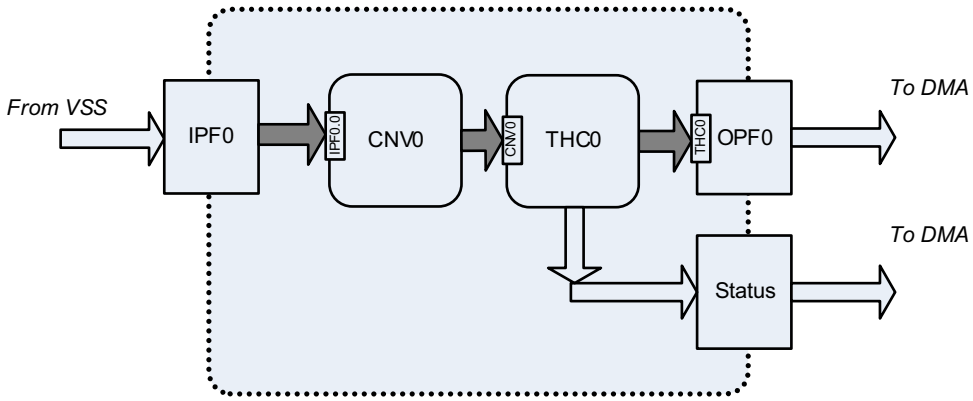


Figure 30-31: Computation of 16-level Histogram

Some key points in this camera pipe example are:

- The IPF0 block receives data from the video subsystem (VSS).
- The CNV0 block receives data from port 0 of IPF0. The CNV0 block can do low pass filtering to eliminate noise and can optionally down scale the input for the THC0 block.
- The THC0 block operates in the quantization mode by setting the PVP_THCn_CTL.MODE bit =1. The threshold values are set in the PVP_THCn_TH0 - PVP_THCn_TH15 (y = 0–15) registers. The THC0 block is programmed to output a 4-bit index without any compression when the PVP_THCn_CTL.OFRMT bit =2.The histogram function of THC0 is enabled by setting the PVP_THCn_CTL.HISTEN bit =1. The number of frames used to perform the histogram operation is set in the PVP_THCn_HFCNT.VALUE bit field.
- The OPF0 block is set to pack the 4-bit output from THC0 to a 32-bit word when the PVP_OPFn_CTL.ISIZE bit =3 and the PVP_OPFn_CTL.OSIZE bit =0. The histogram values are read using the status DMA channel. The values can also be read by directly by accessing the PVP_THCn_HCNT0_STAT - PVP_

THCn_HCNT15_STAT (y = 0–15) registers. The PVP_THCn_HFCNT_STAT register indicates the current frame counter number to which the histogram values have been accumulated.

Sobel with 3x3 or 5x5 Matrix Operation

The Sobel algorithm is a standard algorithm used for edge detection applications. The example in the Sobel data flow figure shows a simple Sobel shows how the CNVn and PMA blocks may be applied. One convolution block determines the horizontal component of 1st-derivative edge gradients, and another convolution block determines the vertical component.

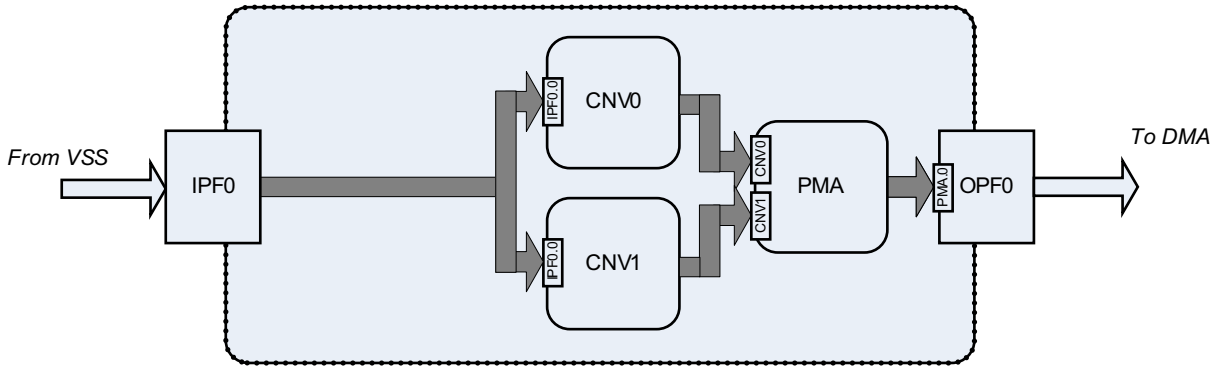


Figure 30-32: Sobel Data Flow

Some key points in this camera pipe example are:

- The IFP0 block receives input from the video subsystem (VSS).
- The CNVn blocks receive data from port 0 of input formatter IPF0.Traditional Sobel operators require simple 3x3 convolution matrices. The 5x5 coefficient matrices of the convolution blocks might be filled with 16-bit values as shown in the traditional 3x3 Sobel coefficients figure.

$$Coeff_{CNV0} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 2 & 0 & -2 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad Coeff_{CNV1} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 30-33: Traditional 3x3 Sobel Coefficients

Larger Sobel kernels can be realized using 5x5 operators. One option is shown in the example of 5x5 coefficients figure.

$$Coeff_{CNV0} = \begin{pmatrix} 1 & 2 & 0 & -2 & -1 \\ 4 & 8 & 0 & -8 & -4 \\ 6 & 12 & 0 & -12 & -6 \\ 4 & 8 & 0 & -8 & -4 \\ 1 & 2 & 0 & -2 & -1 \end{pmatrix} \qquad Coeff_{CNV1} = \begin{pmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & -12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix}$$

Figure 30-34: Example of 5x5 Coefficients

- The PMA block inputs are 16-bit signed data. The PMA block takes the 16-bit signed Cartesian values from the two convolution operations. The PMA block converts the inputs to Polar values resulting in magnitude and the angle of the resulting gradients. The magnitude is a 16-bit unsigned format, and the angle is a 5-bit unsigned value.

$$Magnitude = \sqrt{|x|^2 + |y|^2} \qquad \varphi = \arctan(\frac{y}{x})$$

Figure 30-35: PMA Operation

NOTE: Care is required in dealing with data formats and bit growth. If IPF0 receives 14-bit unsigned data that is zero-extended to 16-bits, the 3x3 Sobel operation has a bit growth of 2 in each convolution block's accumulator. Additionally, the result is signed, making it a 17-bit signed value. Therefore, both convolution outputs need to be normalized by shifting two positions to the right (`PVP_CNVn_CTL.SHIFT` =2). Alternatively, the accumulation results can be saturated to 16-bit signed values if no shift is performed and the `PVP_CNVn_CTL.SAT32` bit was kept clear. If the input formatter receives 16-bit signed values and the above 5x5 coefficient matrix is used, then the accumulator holds 22-bit signed values and a `PVP_CNVn_CTL.SHIFT` =6 setting is required.

Sobel Output Formats

The PVP blocks support a variety of output formats for the Sobel algorithm. The block configurations permit selection of output types (magnitude and/or angle) and sizes (32-bit, 16-bit, or other).

The PMA block has three output ports. PMA port 0 outputs the 16-bit unsigned magnitude. PMA port 1 outputs the 5-bit angle. PMA port 2 outputs the combination of the magnitude and angle (the lower 16 bits contain the magnitude and bits 20:16 contain the angle). All three PMA ports zero fill the upper bits, if connected to 32-bit data sinks.

If a system is only requires the 16-bit magnitude values, one of the output formatters is programmed to input data from PMA port 0 or from port 2. The output formatter uses 16-to-32-bit packing and sets `PVP_OPFn_CTL.ISIZE` =1, `PVP_OPFn_CTL.IUP16` =0, and `PVP_OPFn_CTL.OSIZE` =0.

If a system is only requires the 5-bit magnitude values, one of the output formatters is programmed to input data from PMA port 1. The output formatter uses 8-to-32-bit packing and sets PVP_OPFn_CTL.ISIZE =2, PVP_OPFn_CTL.IUP16 =0, and PVP_OPFn_CTL.OSIZE =0. Alternatively, the OPFn can take data from port 2 and set PVP_OPFn_CTL.IUP16 =1.

If the system requires both magnitude and angle values, two output formatters can be used. It is recommended (for best system resource usage) that this configuration only use one output formatter, which inputs data from PMA port 2 in 32-bit mode where (PVP_OPFn_CTL.ISIZE =0).

Often, an application does not require 16 bits of magnitude resolution, needing only one bit (binary edge map) or wanting only a few bits. The example in the thresholded edge map data flow figure shows how the threshold blocks are used for this application.

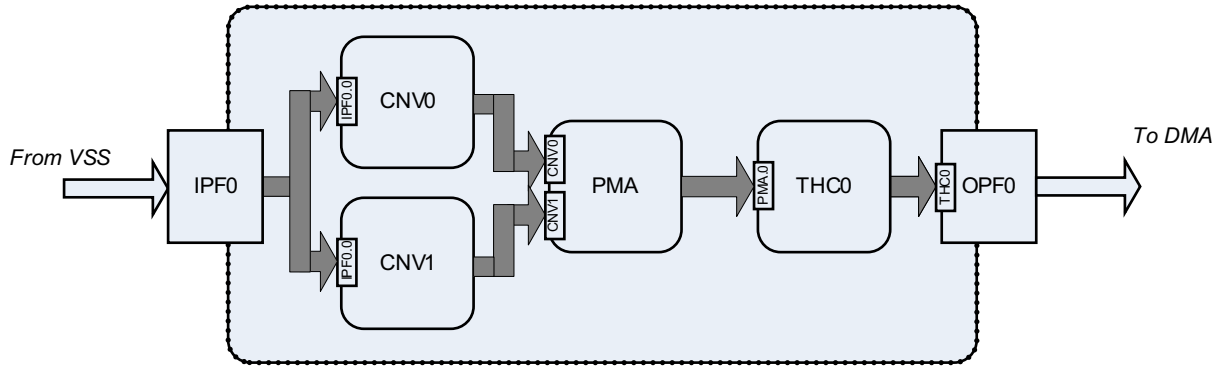


Figure 30-36: Thresholded Edge Map Data Flow

Some key points in this camera pipe example are:

- A binary edge map can be generated by the THCn using quantization mode. The PVP_THCn_TH0 register is set to zero, and the PVP_THCn_TH1 register defines the threshold. If the output format is set to the 4-bit floor value (PVP_THCn_CTL.OFRMT =1), a 4-bit magnitude value is sent. The connected output formatter may perform 4-to-32-bit packing by the setting the PVP_OPFn_CTL.ISIZE bit = 3. All 16 thresholds can be configured to apply a non-linear quantization of the magnitude value. Alternatively, hysteresis mode can be used. Clipping mode, when paired with PVP_THCn_CTL.OFRMT =0, can be used to output user-programmable 32-bit values when the magnitude is above the threshold.
- On the input side, the THCn blocks provide multiple options: if they sense Port 0 of the PMA block, they receive 16-bit values that are zero-extended to 32 bits. If they sense Port 2 instead, the 5-bits of might be unwanted. Then, the PVP_THCn_CTL.ZEXT =1 setting helps masking the upper 16 bits. This feature becomes most interesting when used with the PVP_THCn_CTL.OFRMT =4 option. This way, the THC generates an output format that groups the 4-bit floor value of the magnitude with the four significant bits of the PMA angle. The Output Formatter is best set to PVP_OPFn_CTL.ISIZE =2 mode.

Canny with PEC in 1st-Derivative Mode

The Canny algorithm is a standard algorithm for edge detection applications. The input pixels are first low pass filtered to remove noise by CNV0. One convolution block is used to determine the horizontal compo-

nent of 1st-derivative edge gradients, and another convolution blocks determines the vertical component as in the Sobel case. The Canny data flow figure shows connections for PVP block for a Canny use case.

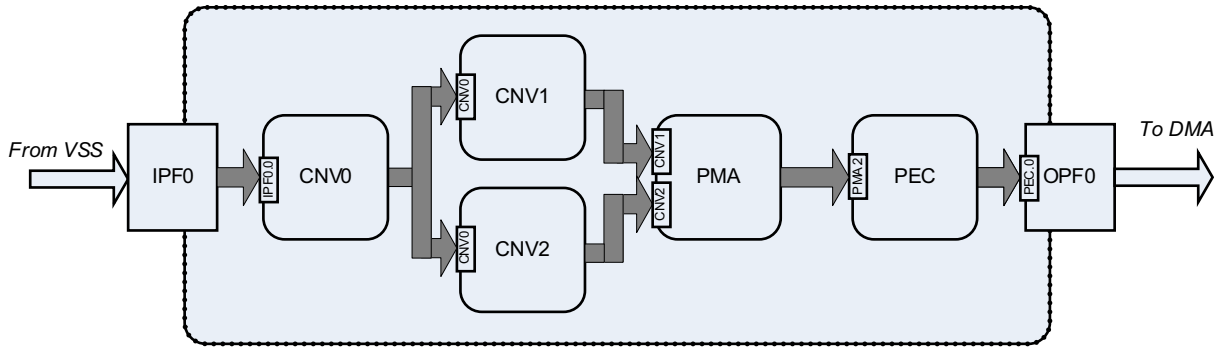


Figure 30-37: Canny Data Flow

Some key points in this camera pipe example are:

- The IPF0 block receives input from the video subsystem (VSS).
- The CNV0 block receives data from port 0 of IPF0. The fractional values are converted to 1.15 format and are programmed into the coefficient registers of CNV0. The output is shifted right by 15 bits to remove the fractional part by setting `PVP_CNVn_CTL.SHIFT` to 15. The low pass filtered output feeds the two convolution blocks, which generate the gradients as described in the *Sobel with 3x3 or 5x5 Matrix Operation* example. The Gauss filter figure shows a 5x5 Gaussian kernel (Sigma = 1) for low pass filtering.

$$Gaussian\ Kernel = \frac{1}{330}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 20 & 33 & 20 & 4 \\ 7 & 33 & 54 & 33 & 7 \\ 4 & 20 & 33 & 20 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \qquad Coeff_{CNV0} = \begin{bmatrix} 0x0063 & 0x018d & 0x02b7 & 0x018d & 0x0063 \\ 0x018d & 0x07c2 & 0x0ccd & 0x07c2 & 0x018d \\ 0x02b7 & 0x0ccd & 0x14f2 & 0x0ccd & 0x02b7 \\ 0x018d & 0x07c2 & 0x0ccd & 0x07c2 & 0x018d \\ 0x0063 & 0x018d & 0x02b7 & 0x018d & 0x0063 \end{bmatrix}$$

Figure 30-38: Gauss Filter

- The PMA block takes the 16-bit signed Cartesian values from the two convolution operations. The PMA block converts the inputs to polar values that result in magnitude and the angle of the resulting gradients. The magnitude is in a 16-bit unsigned format, and the angle is a 5-bit unsigned value.
- The PEC block interfaces to PMA port 2, which has the magnitude and angle packed into a 32-bit word. The PEC performs the non-maximum suppression to produce thin edges and hysteresis threshold to allow software to reduce streaking. Because the PEC block works on the output from 1st-derivative filters, the block should be configured for mode 1 by setting `PVP_PEC_CTL.MODE` =0. The threshold values for magnitude comparison is set in the `PVP_PEC_D1TH0` register (lower threshold) and is set in the `PVP_PEC_D1TH1` register (higher threshold). The hysteresis threshold feature can be selectively disabled by setting `PVP_PEC_CTL.IGNTH1` =1.

- The OPF0 block can be set to sense PEC port 0, which carries the 8-bit encoded output. The post-processing software can use this output for edge tracing and further processing. The OPF0 block should be configured to PVP_OPFn_CTL.ISIZE =2 and configured to PVP_OPFn_CTL.OSIZE =0.

LoG with PEC in 2nd-Derivative Mode

Laplacian on Gaussian (LoG) is an edge detection algorithm based on 2nd derivative filtering of an input image. The LoG method produces thin edges, but the algorithm is sensitive to noise. To avoid noise, the input image is filtered using a Gaussian kernel before convolving the input with the LoG kernel. The PEC block produces the angle information, which is missing in the traditional LoG algorithm. The post-processing software can use the class/angle information from the PEC to build contours for further analysis. The LoG data flow figure shows PVP block connections for this usage case.

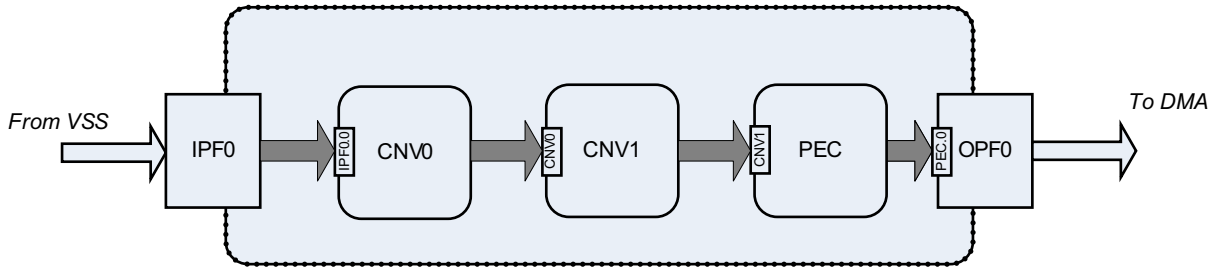


Figure 30-39: LoG Data Flow

Some key points in this camera pipe example are:

- The IPF0 block receives input from the video subsystem (VSS).
- The CNV0 block receives data from port 0 of IPF0. The CNV0 use the same Gaussian kernel described in the *Canny with PEC in 1st-Derivative Mode* example. The coefficients are programmed in 1.15 format in the CNV1 registers. The fractional part is eliminated in the output of CNV1 by setting PVP_CNVn_CTL.SHIFT =15. The LoG kernel is shown in the LoG coefficients figure.

$$LoG\ Kernel = \frac{1}{16}\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix} \qquad Coeff_{CNV1} = \begin{bmatrix} 0 & 0 & 0xf800 & 0 & 0 \\ 0 & 0xf800 & 0xf000 & 0xf800 & 0 \\ 0xf800 & 0xf000 & 0x7fff & 0xf000 & 0xf800 \\ 0 & 0xf800 & 0xf000 & 0xf800 & 0 \\ 0 & 0 & 0xf800 & 0 & 0 \end{bmatrix}$$

Figure 30-40: LoG Coefficients

- The PEC block interfaces to the CNV1 output. The PEC computes angle information and sub-pixel location of the edges. Because the PEC block works on outputs from 2nd derivative filters, the block should be configured to mode 2 by setting PVP_PEC_CTL.MODE =1. The threshold values for zero crossing detection is set in the PVP_PEC_D2TH0 (lower threshold) and is set i the PVP_PEC_D2TH1 (higher threshold) registers. The strong/weak edge distinction can be selectively disabled by setting

PVP_PEC_CTL.IGNTH1 =1. The PEC block can output either the zero-crossing codes (PVP_PEC_CTL.ZCRSS =1) or can output the angle information (PVP_PEC_CTL.ZCRSS =0) from the classifier. In both cases, the output on PEC port 0 is in 8-bit packed format. The output contains the sub-pixel location in the upper 4-bits and contains either zero-cross code or angle index.

- The OPF0 block can be set to sense PEC port 0, which carries the 8-bit encoded output. The post-processing software can use this output for building contours and further processing. The OPF0 block should be configured to PVP_OPFn_CTL.ISIZE =2 and PVP_OPFn_CTL.OSIZE =0.

The concurrency of Sobel and LoG methods figure demonstrates how both 1st-derivative and 2nd-derivative edge detection methods can be realized concurrently in the PVP.

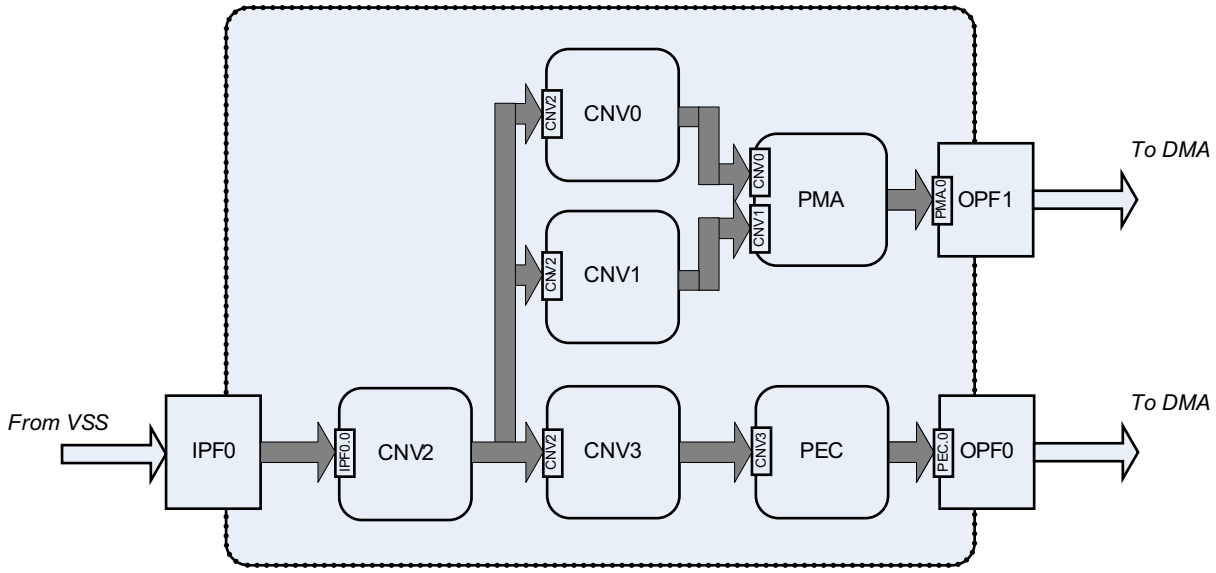


Figure 30-41: Concurrency of Sobel and LoG Methods

In this example, the CNV2 performs the initial low-pass filtering to remove noise in the input data. The output is broadcast to the three CNVn blocks. The lower signal pipe using CNV3-PEC realizes the 2nd-derivative method of edge detection using the LoG method. The upper signal pipe realizes the 1st-derivative method of edge detection using the Sobel method. The outputs are captured using two OPFn blocks concurrently.

DoG with PEC in 2nd-Derivative Mode

Difference of Gaussian (DoG) is an edge detection algorithm, which approximates the 2nd derivative filtering of an input image. The input image is filtered by two Gaussian kernels with different sigma. The resulting output is subtracted to produce an output similar to the filtering using LoG kernel. This output is fed to PEC which works in the same manner as in LoG method. The post-processing software can use the class/angle information from PEC to build contours for further analysis. The difference of Gaussian data flow figure shows the data flow for the DoG method.

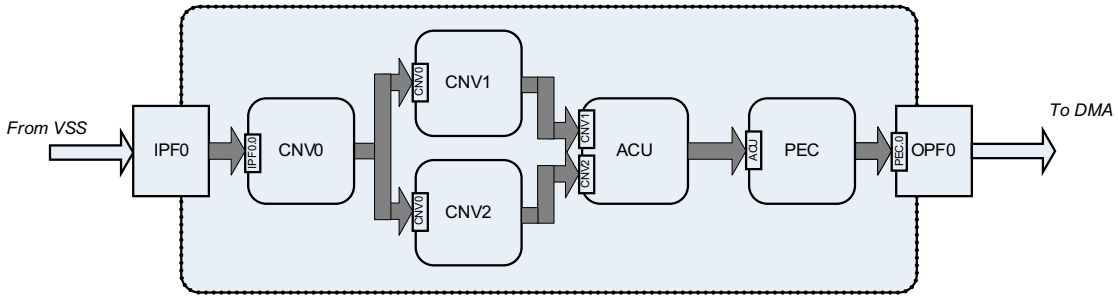


Figure 30-42: Difference of Gaussian Data Flow

Some key points in this camera pipe example are:

- The input data is first convolved with Gaussian kernel in CNV0. The output is convolved a second time with a Gaussian kernel in CNV1. The resulting output is the result from convolving the original image with a 9x9 Gaussian kernel of sum of sigmas of CNV0 and CNV1.
- The output from CNV0 can be subtracted from the output of CNV1 using the ACU. But, the two outputs have to be matched for the delay to subtract the pixels at the right instant/location. This match is achieved by inserting CNV2, which does not modify the data and does match the delay introduce by CNV1. The kernel for bypassing the input to the output is shown in the bypass coefficients for CNV2 figure.

$$Coeff_{CNV2} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 30-43: Bypass Coefficients for CNV2

- The PEC block and OPF0 block settings in this example are the same as shown in the *LoG with PEC in 2nd-Derivative Mode* example.

Integral of Input Pixels

The integral image of input pixels data flow figure shows how the IIMn block can be used to compute an integral of input pixels.

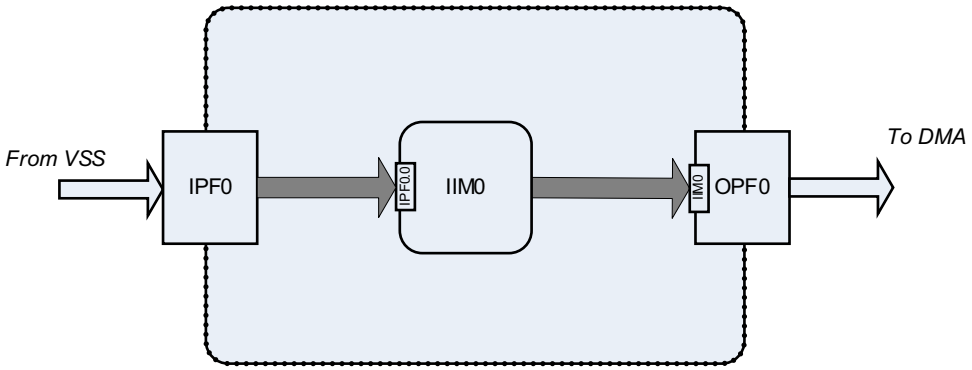


Figure 30-44: Integral Image of Input Pixels Data Flow

Some key points in this camera pipe example are:

- The IPF0 block receives input from the video subsystem (VSS).
- The IIM0 block receives data from port 0 of the IPF0 block. The IIM0 can be used to compute sum-area-table (SAT) features by setting `PVP_IIMn_CTL.MODE` =0. If rotated-sum-area-table (RSAT) features are required, configure `PVP_IIMn_CTL.MODE` =1.
- Because the integral is computed on single pixels coming in from IPF0, the `PVP_IIMn_CTL.WIDTH` =0 (32-bit input). Note that IPF0 zero extends the incoming pixel values to 32-bits before placing on them on port 0.
- To reduce overflow or underflow while performing the integral, the IIM block supports reducing the input range by shifting the LSBs out of incoming data. The `PVP_IIMn_CTL.SHIFT` bits can be set to a non-zero value to reduce the pixel data range. The IIM can optionally also scale down the output data in horizontal and vertical directions independently. The `PVP_IIMn_SCALE.HSCL` and `PVP_IIMn_SCALE.VSCL` fields can be used for setting the scale factors.
- The OPF0 block should be set to receive and transmit 32-bit data (for best operation) using the configuration: `PVP_OPFn_CTL.ISIZE` =0 and `PVP_OPFn_CTL.OSIZE` =0.

Integral of Binary Edge Map

The integral image of binary edge map data flow figure shows how the IIMn block can interface to the THCn and other blocks to compute integral of edge map.

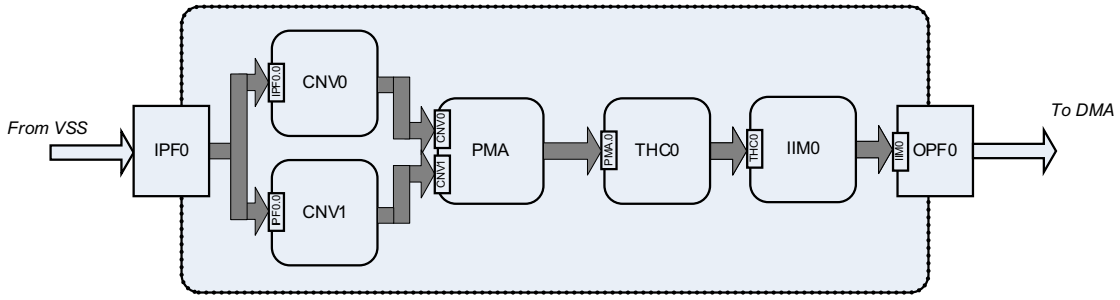


Figure 30-45: Integral Image of Binary Edge Map Data Flow

Some key points in this camera pipe example are:

- The edge map is generated as described in the *Sobel with 3x3 or 5x5 Matrix Operation* example.
- The 4-bit index output from the THC0 block forms the input to the IIM0 block. By properly choosing threshold registers (`PVP_THCn_TH0` = 0, `PVP_THCn_TH1` = threshold value, and `PVP_THCn_TH2` through `PVP_THCn_TH15` = maximum of 16-bit value, 2^{16}–1), the output from the THC0 block is only 0 and 1, a true binary edge map.
- The IIM0 block is configured in different modes as explained in the *Integral of Input Pixels* example.

Integral of Variance

The IIMn block can interface to the ACU block to compute integral of squared input values. The input data from IPF0 can be fed to both inputs of the ACU as shown in the integral of squared input values data flow figure.

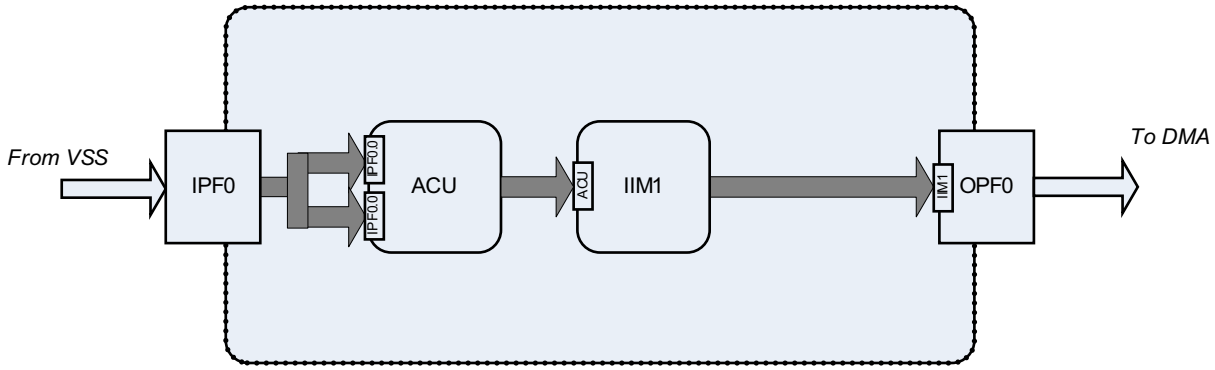


Figure 30-46: Integral of Squared Input Values Data Flow

Some key points in this camera pipe example are:

- The ACU is set to multiply the input operands (PVP_ACU_CTL.SFTINP =2, PVP_ACU_CTL.PRDINP =0, PVP_ACU_CTL.PRDOP =0, PVP_ACU_SHIFT =0). The ACU also supports user programmable clipping with values from the PVP_ACU_MIN and PVP_ACU_MAX registers.
- The IIM1 block is configured in different modes as explained in the *Integral of Binary Edge Map* example.

Histogram of Gradients (HoG)

The IIMn block can be used to compute Histogram of Gradients (HoG) features by connecting IIMn block output to PEC block (port 1) input as shown in the histogram of gradients data flow figure.

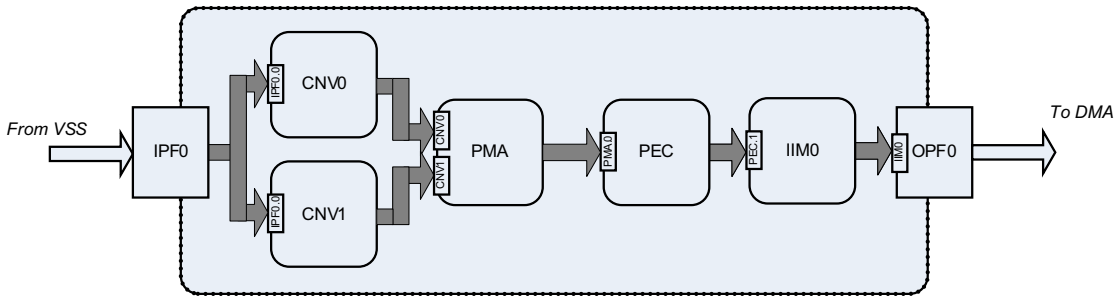


Figure 30-47: Histogram of Gradients Data Flow

Some key points in this camera pipe example are:

- The PEC block port 1 carries angle information in either 8-bit or 16-bit fields. This format is supported in both modes (1st and 2nd derivative) of the PEC. Port 1 is not valid only when PVP_PEC_CTL.ZCRSS =1 in 2nd derivative mode because the angle is not functional.
- When the PEC block is programmed to output 8-bits per angle bin, (PVP_PEC_CTL.OSIZE =0), there are four angle bins in a 32-bit word output. The IIM0 block computes the integral of the individual bins, and the IIM0 block must be set in quad 8-bit mode, (PVP_IIMn_CTL.WIDTH =3).
- When the PEC block is programmed to output 16-bits per angle bin, (PVP_PEC_CTL.OSIZE =1), there are two angle bins in a 32-bit word output. The IIM0 block computes the integral of the individual bins, and the IIM0 block must be set in dual 16-bit mode, (PVP_IIMn_CTL.WIDTH =1).
- For optimal operation, set the OPFn block to receive and transmit 32-bit data (PVP_OPFn_CTL.ISIZE =0, PVP_OPFn_CTL.OSIZE =0).

Event Control

This section describes PVP event control issues, including interrupt signals, status/error signals, and finish commands.

Interrupt Signals

Although the PVP features two interrupt outputs, much of the software interaction occurs through the interrupts of the associated DMA channels. For example, the PVP interrupt may signal that a frame has been completely received and processed, but only the completion interrupt of the data output DMA can indicate when all results are stored in system memory. For more information about specific PVP interrupts and triggers, see the processor specific interrupts and triggers listed in the *PVP Functional Description*.

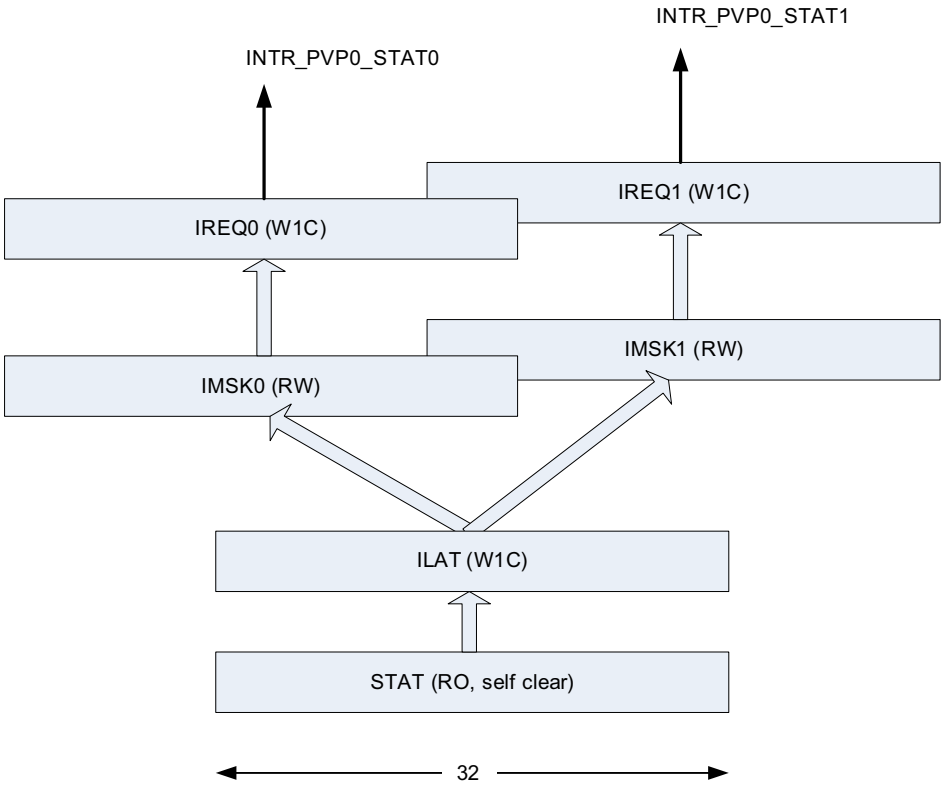


Figure 30-48: PVP Interrupt Flow

The PVP features a single, 32-bit wide event status register, `PVP_STAT`, as shown in the PVP interrupt flow figure. The `PVP_STAT` register contains read-only status bits for camera pipe control, memory pipe control, and individual processing blocks. All flags in this register are generated and cleared by hardware.

The `PVP_ILAT` register contains the latched counterparts of the status bits in the `PVP_STAT` register. If a status bit toggles from 0 to 1 the respective latch bit is set automatically. The latches in the `PVP_ILAT` register are cleared by software using the W1C method.

The `PVP_IMSKn` registers (one for each processor core) can unmask the interrupt behavior of the latches. If the `PVP_STAT.CPRDY` bit (camera pipe ready) is set by hardware, the PVP sets the corresponding latch bit, `PVP_ILAT.CPRDY`. By default, interrupts are not enabled (unmasked). If software sets the `PVP_IMSKn.CPRDY` bit enabling (unmasking) the interrupt, the latched interrupt status causes an interrupt on the PVP status output to the corresponding processor core.

The PVP_IREQn registers (on for each processor core) can be read by software to determine which of the interrupts enabled on a respective channel is pending. Interrupt request bits are a logical AND of mask and latch bits. The interrupt outputs are the logical OR of all interrupt requests in the given PVP_IREQn register.

Software can acknowledge interrupts by a W1C operation to either the PVP_IREQn or the PVP_ILAT registers. Either operation clears the latches in the respective PVP_ILAT register.

Status and Error Signals

All PVP status and error events are reported by the PVP_STAT register. It is noted that further interrupts and error signals out of PPI, PIXC and DMA channels are highly relevant to PVP operation.

The PVP_STAT register contains the following status and error bits for pipe control. There is always one status bit for the camera pipe and one for the memory pipe:

- *Daisy Chain Completion*. The PVP_STAT.CPDC and PVP_STAT.MPDC bits signal whether a configuration update is currently daisy chaining through the pipe. The bits are set along with the PVP_IPF0_CFG.START or PVP_IPF1_CFG.START bit and are cleared if all blocks in the pipe have been updated.
- *Write Error*. The PVP_STAT.CPWRERR and PVP_STAT.MPWRERR bits are set when an MMR write is attempted during ongoing daisy chaining of configuration and are cleared at the end of the daisy chain operation.
- *Completion of Drain operation*. The PVP_STAT.CPDRN and PVP_STAT.MPDRN bits get set when the pipe has finished draining in response to PVP_IPFn_PIPECTL.DRAIN bit. The bits are cleared when PVP_IPF0_CFG.START or PVP_IPF1_CFG.START bit is set next time.
- *Pipe Ready*. The PVP_STAT.CPRDY and PVP_STAT.MPRDY bits indicate that the respective pipe is ready for a new configuration. They are set along with the drain completion bits and cleared by daisy chain completion.

The PVP_STAT register contains the following status bit for the data flow.

- *Data Events*. The PVP_STAT.CPOPFDAT and PVP_STAT.MPOPFDAT bits are set when all enabled output formatters have processed the first data word. The bits are cleared with the next data row.

The PVP_STAT register contains the following error bits for signaling data transfer errors in camera pipe. Since the memory pipe is designed to not overflow it does not need the following error flags.

- *Output FIFO Overflow*. The PVP_STAT.OPF0OVF, PVP_STAT.OPF1OVF, and PVP_STAT.OPF2OVF bits are set when the FIFO overflows, which is fed by OPF outputs and drained by data output DMA. The bits are cleared when the first data value of the next row is processed by the OPF.
- *Status FIFO Overflow*. The PVP_STAT.CPSTOVF flag is similar to OPFnOVF bits. It is set when the status output FIFO overflows.
- *Input Overflow*. The PVP_STAT.CPIPF0VF flag is set when data overflows at IPF0. This flag can only be cleared by a camera pipe disable (PVP_CTL.CPEN =0).

The PVP_STAT register contains the following status and error bits to report the status of individual processing blocks.

- *Histogram Ready*. The PVP_STAT.THC0RDY and PVP_STAT.THC1RDY bits indicate that the THC histogram counters have been updated. Set at the end of a frame when PVP_IPFn_FCNT =0 or has counted down to zero and PVP_IPFn_PIPECTL.STATEN =1.
- *Integral Unsigned Overflow*. The PVP_STAT.IIM0UOVF and PVP_STAT.IIM1UOVF bits are set when bit 31 of the integral counter in the IIM overflowed during an on-going frame.
- *Integral Signed Overflow*. The PVP_STAT.IIM0SOVF and PVP_STAT.IIM1SOVF bits are set when bit 30 of the integral counter in the IIM overflowed during an on-going frame
- *Divide by Zero*. The PVP_STAT.ACUDIVERR bit is set by the ACU when a divide-by-zero operation was attempted during the on-going frame.
- *ACU Output Saturation*. The PVP_STAT.ACUOUTSAT bit is set when, during the on-going frame, data was saturated due to the PVP_ACU_MIN/PVP_ACU_MAX mechanism.
- *ACU Multiplication Saturation*. The PVP_STAT.ACUPRODSAT bit is set when, during the on-going frame, a multiplication product was saturated.
- *ACU Addition Saturation*. The PVP_STAT.ACUSUMSAT bit is set when, during the on-going frame, the sum of an addition or subtraction was saturated.

Finish Commands

Output formatters (OPFn) are the blocks that drive data output DMA processes. These PVP blocks use a special DMA feature that permit these blocks (on demand) to send a finish command to the DMA channel, signaling that the work unit has completed.

The three camera pipe output formatters (OPF0, OPF1, OPF2) send a finish command when the FIFO overflows. When an overflow occurs, the condition is reported to the PVP_STAT register, the FIFO is cleared, and a finish command is sent to the DMA channel at the end of the corrupted frame. The PVP and DMA automatically re synchronize with the next frame boundary. Due to the large depth of the data output FIFO, multiple frames may stick in the FIFO by the time the error occurs. In case of very small frames, software may not know whether the automatic recovery dropped one or multiple frames.

The Finish command is also used for signaling. The PVP_OPFn_CTL.FINISH bit configures this operation. When set, the respective output formatter always issues a Finish command at the end of each frame. This feature is valuable when the exact amount of transfers per frame is not known.

A good example applying the Finish command is run-length compression mode. In this mode, the number of output reports generated depend on the incoming data. The DMA work unit is pre configured for a certain number of transfers and is completed only if all transfers have taken place. Typically, the run-length compression output DMA is set to the maximum value of reports a frame can generate. This value depends on the PVP_THCn_RMAXREP register setting. If PVP_OPFn_CTL.FINISH =1, the output DMA can terminate before all DMA transfers have occurred. The DMA can immediately progress to the next frame's opera-

tion. When using this feature, software needs to be aware that not all memory cells reserved for the operation might be filled with new value.

Programming Model

This section describes PVP programming techniques, including configuration for pipes, blocks, daisy chains, job lists, and reports.

Configuring Pipe Structure

PVP processing blocks either have a single fixed output or have multiple outputs. Also, the PVP has some inputs containing multiplexers. This pipe connectivity provides a number of ways to connect block output as input to other blocks.

The block connectivity overview figure provides a block connectivity overview for the PVP blocks.

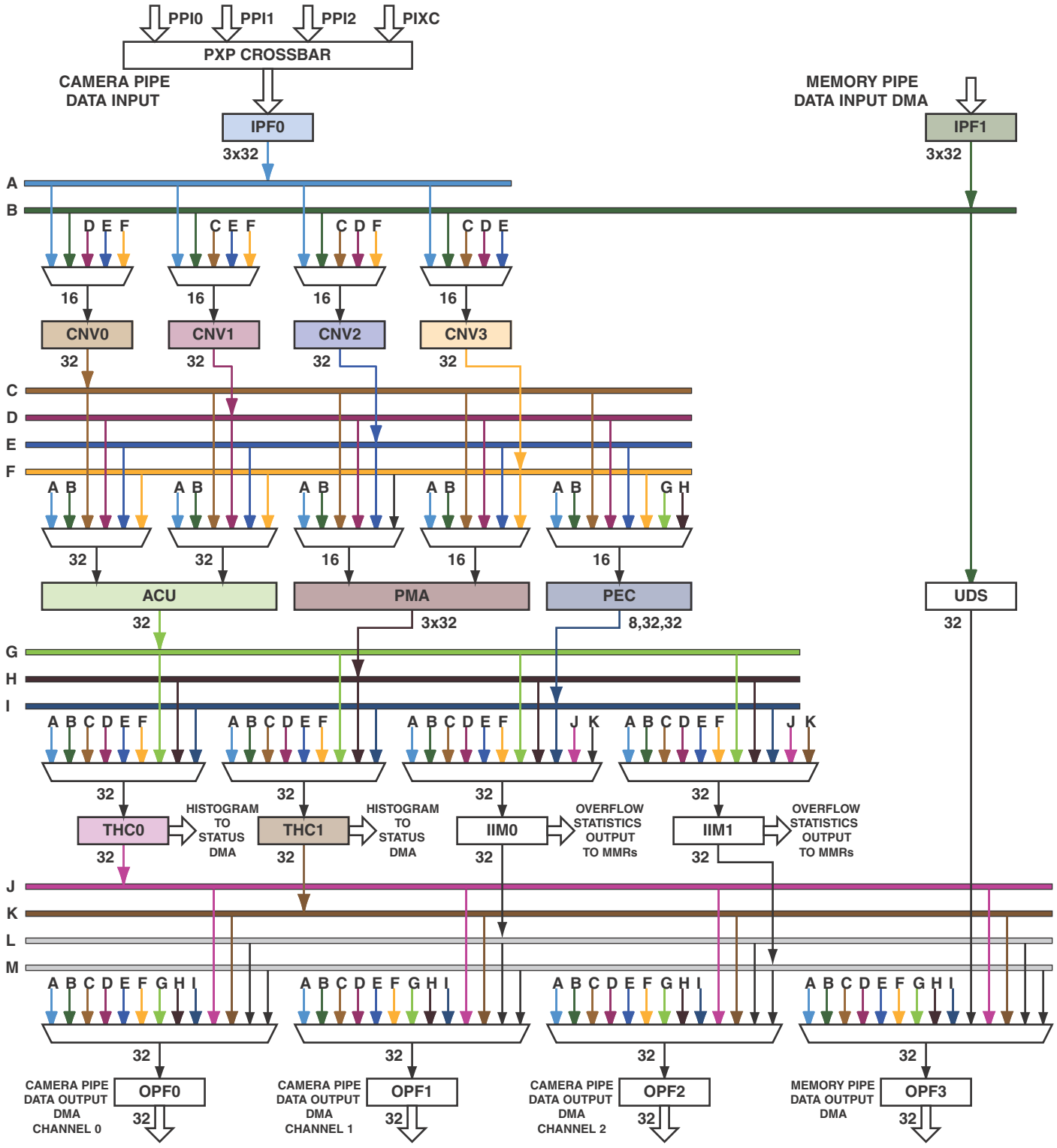


Figure 30-49: Block Connectivity Overview

The pipeline structure is composed by programming multiplexers from the output of PVP blocks to the input of other PVP blocks. By definition, this connection programming starts with one of the OPFn blocks. For example, if the input of OPF0 is connected to one of the three output ports of an empty camera pipe on IPF0, a camera pipe without any mathematical mean has been created. Alternatively, if the input of OPF0 is connected to the output of THC1, and the input of THC1 is connected to an output of IPF0, a camera pipe with a single mathematical function has been created.

To continue the pipe connection example, one output of the IPF0 can perform 3-way data broadcasting with the following configuration.

- OPF1 input is connected to the IIM0 output
- The IIM0 input is connected to the ACU output
- The ACU has two inputs and both inputs are connected to the same output of IPF0 and THC1

Similarly, to configure OPF2 to probe the PMA's output use the following configuration.

- Connect the two PMA inputs to CNV1 and CNV2
- Connect these to CNV0's output
- CNV0's input finally receives data from a different port of IPF0

The pipe structure example configuration figure illustrates this example. Additionally, the figure shows the UDS block configured in memory pipe between OPF3 and IPF1.

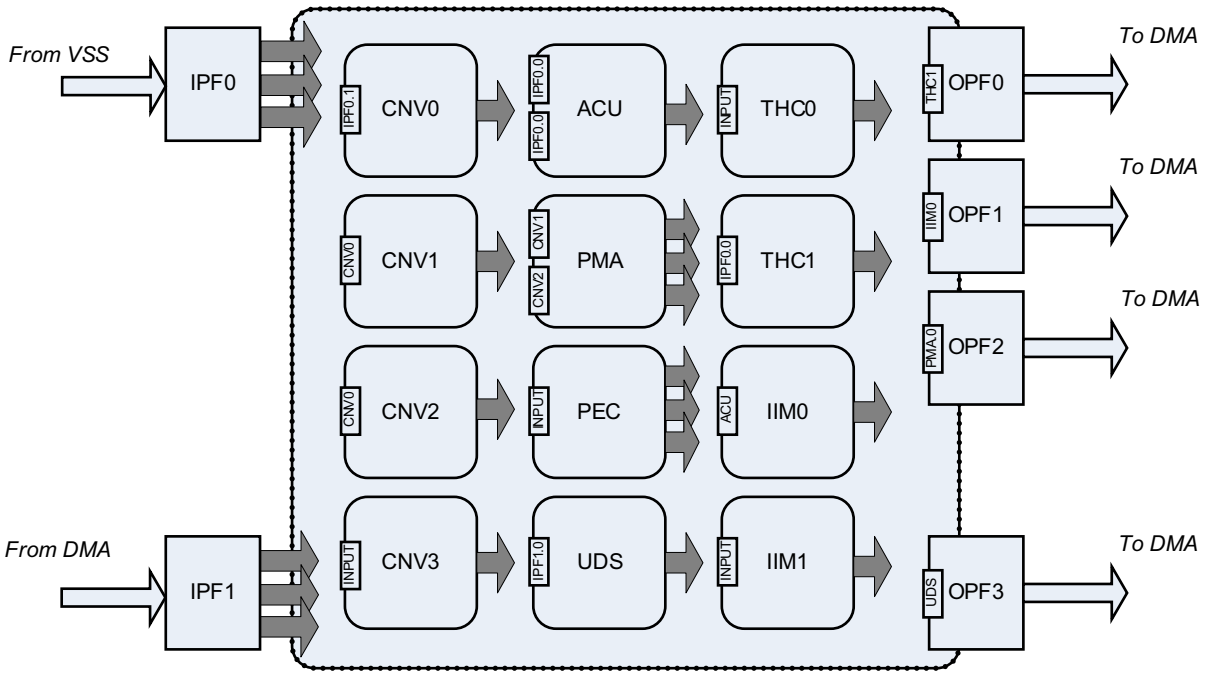


Figure 30-50: Pipe Structure Example Configuration

The next figure shows the data flows in the same example configuration. Note the broadcast at the IPF0 and CNV0 outputs. Also, note how threads are united by the ACU and PMA blocks.

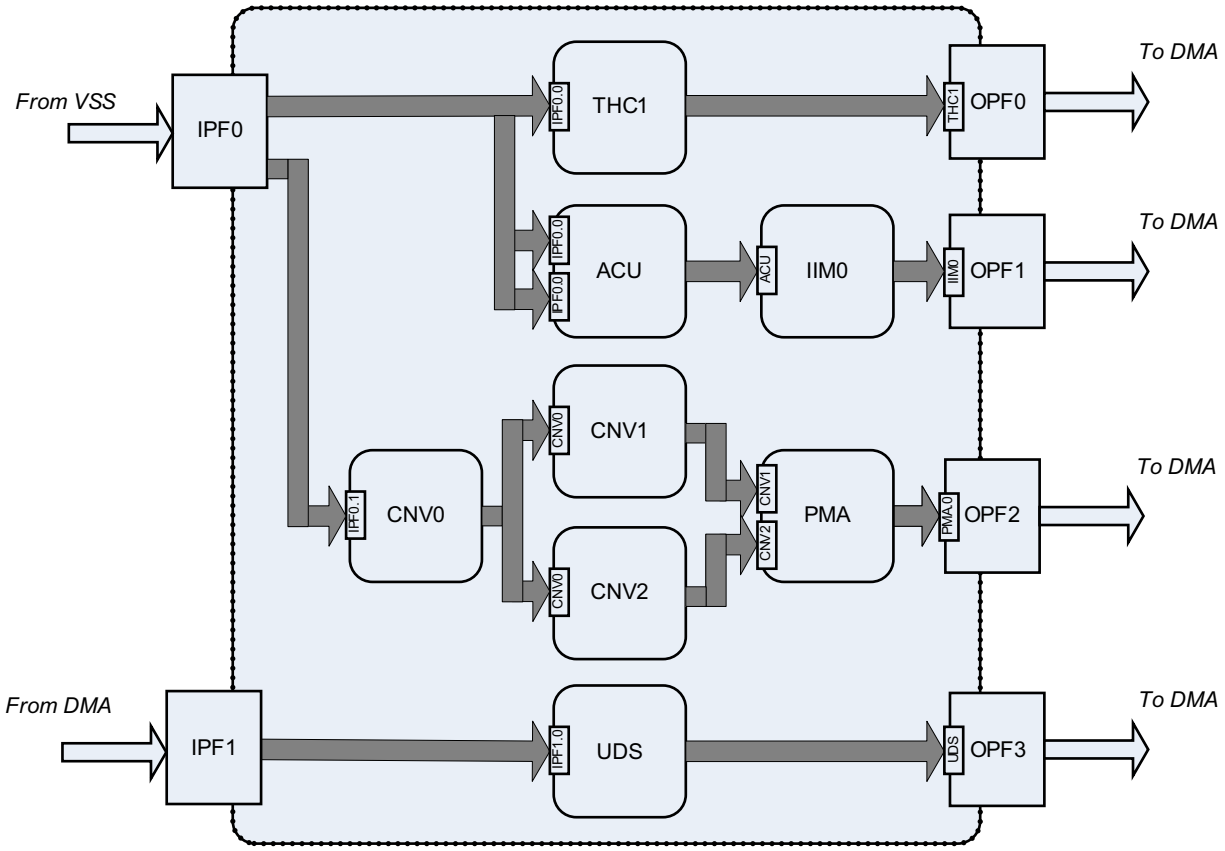


Figure 30-51: Data Flow in Example Configuration

Although the example shown has significant processing power, this pipe configuration does not use the THC0, IIM1, CNV3, and PEC processing blocks.

To program the input multiplexers, each processing element features a pipe configuration register. The PVP_OPFn_CFG.IBLOCK0, PVP_PEC_CFG.IBLOCK0, PVP_IIMn_CFG.IBLOCK0, PVP_ACU_CFG.IBLOCK0, PVP_UDS_CFG.IBLOCK0, PVP_CNVn_CFG.IBLOCK0, PVP_THCn_CFG.IBLOCK0, and PVP_PMA_CFG.IBLOCK0 fields specify the identifier of the desired source block. The PVP_OPFn_CFG.IPORT0, PVP_PEC_CFG.IPORT0, PVP_IIMn_CFG.IPORT0, PVP_ACU_CFG.IPORT0, PVP_UDS_CFG.IPORT0, PVP_CNVn_CFG.IPORT0, PVP_THCn_CFG.IPORT0, and PVP_PMA_CFG.IPORT0 fields are used if the selected source block has multiple outputs. Because the PMA and ACU have two inputs, their configuration register features include the PVP_ACU_CFG.IBLOCK1, PVP_PMA_CFG.IBLOCK1, PVP_ACU_CFG.IPORT1, and PVP_PMA_CFG.IPORT1 fields.

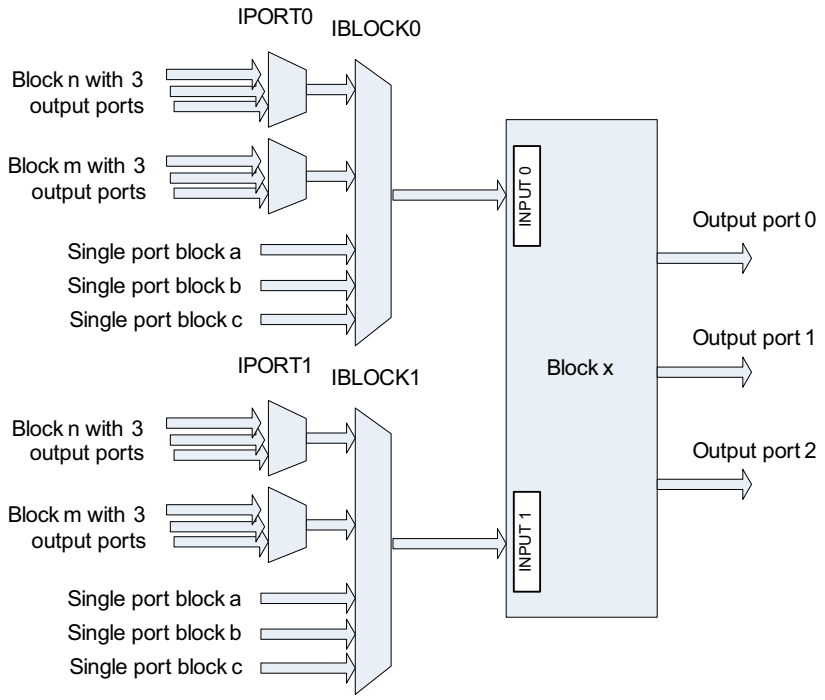


Figure 30-52: Input Port Selection

Table 30-46: PVP Block IDs

Block	Block ID	Block	Block ID	Block	Block ID	Block	Block ID
OPF0	0x01	OPF1	0x02	OPF2	0x03	OPF3	0x04
IPF0	0x0C	IPF1	0x0E	THC0	0x20	THC1	0x28
CNV0	0x10	CNV1	0x14	CNV2	0x18	CNV3	0x1C
ACU	0x08	PMA	0x30	PEC	0x05	UDS	0x0A
IIM0	0x06	IIM1	0x07				

It is mandatory that all blocks that are used have their `MPIPE` bit (`PVP_OPFn_CFG.MPIPE`, `PVP_PEC_CFG.MPIPE`, `PVP_IIMn_CFG.MPIPE`, `PVP_ACU_CFG.MPIPE`, `PVP_UDS_CFG.MPIPE`, `PVP_CNVn_CFG.MPIPE`, `PVP_THCn_CFG.MPIPE`, and `PVP_PMA_CFG.MPIPE`) properly configured. All blocks that contribute to the OPF3 output, (all blocks operating in the memory pipe), must have their `MPIPE` bit set. For all non-contributor blocks, their `MPIPE` bit must be cleared.

NOTE: It is user's responsibility to ensure that the `MPIPE` bit settings match with the assignments of OPF0 through OPF2 versus OPF3. Broadcasts and unions from the camera pipe to the memory pipe and vice versa are absolutely forbidden.

A valid pipe always starts at the input formatter and ends at the output formatter with one exception. Pipes can end at the THC blocks if the data output of the THC blocks is not of interest. Then, the THC blocks can still operate in histogram mode, where the histogram is outputted to the status DMA. A limitation applies in that such a histogram-only THC setup cannot partner with UDS block in memory pipe mode.

Configuring with Register-Based Method (Camera Pipe)

To configure the camera pipe in the static way, use the register-based programming method. This method programs all pipe control and processing blocks by writing configuration and coefficients into memory-mapped registers (MMRs). The following procedure lists the typical steps for this method:

1. Disable all involved PVP blocks, all involved VSS elements and all involved DMA channels to ensure that all data FIFOs are empty.
2. Configure video interconnect routing.
3. Configure pin muxing for PPI input operation.
4. Set the PVP_CTL.PVPEN and PVP_CTL.CPEN bits to enable camera pipe mode.
5. Configure and enable all output DMA processes using either the register-based or the descriptor-based method.
6. Configure all camera pipe processing blocks starting from the output formatters to IPF0. Ensure that the structure of the camera pipes are configured using the PVP_OPFn_CFG, PVP_PEC_CFG, PVP_IIMn_CFG, PVP_ACU_CFG, PVP_UDS_CFG, PVP_CNVn_CFG, PVP_THCn_CFG, and PVP_PMA_CFG registers. Ensure that the appropriate MPIPE bits are cleared, and ensure that the appropriate START bits are set.
7. Enable the IPF0 block. If continued camera pipe operation is desired, set PVP_IPF0_FCNT = 0. Finally, setting the PVP_IPF0_CFG.START bit enables the entire camera pipe.
8. Configure PPI DMA processes.
9. Enable PPI and (optionally) PIXC operation.

After all PVP blocks are enabled, the PVP operation starts with the next vertical frame sync (VSYNC).

There is some flexibility in the order of above procedure. For example, the PVP camera pipe can be reconfigured, while the PPI is kept enabled. The camera pipe starts operating with the next VSYNC. If frame characteristics change (for example, as in the case of the PPI's window feature), care is required to ensure that the PVP output (2-dimensional) DMA processes match with the window settings every frame.

All block registers are double buffered. All coefficient registers can be written any time and are properly synchronized with pipe progress by hardware. However, the PVP_OPFn_CFG, PVP_PEC_CFG, PVP_IIMn_CFG, PVP_ACU_CFG, PVP_UDS_CFG, PVP_CNVn_CFG, PVP_THCn_CFG, and PVP_PMA_CFG registers must only be written when a pipe re-configuration is desired for the next set of frames. Writes to the PVP_IPF0_CFG or PVP_IPF1_CFG registers need to be synchronized with ongoing activity.

NOTE: Register writes during daisy chain load period are not allowed. Writes are ignored and error bits are set. MMR reads incur 1 cycle of latency while writes incur no latency. Also, note that access to holds/read only gives MMR error.

Configuring with DMA-Based Method

Instead of programming the PVP by directly writing to memory-mapped registers, the PVP can be configured using DMA channels. This DMA-based method is derived from the descriptor-based programming method of the DMA controllers themselves and works well with DMA programming.

The PVP features two configuration DMA inputs, one for camera pipe and one for memory pipe operation. These DMA processes enable on-the-fly re-configuration of the two pipes independently. Each pipe can be configured or re-configured according to timing needs without core interaction.

The configuration DMA channels write to the memory mapped register (MMR) space. For the PVP processing blocks, it does not make any difference whether a memory-mapped register has been written by a core write or written by configuration DMA. Configuration DMA operation's write timing is ensured by hardware and not subject of interrupt latencies.

The block configuration structure (BCS) is a telegram loaded by either of the configuration DMA operations. The structure consists of a 32-bit block configuration header (BCH) which is followed by a number of 32-bit configuration words that target memory-mapped registers.

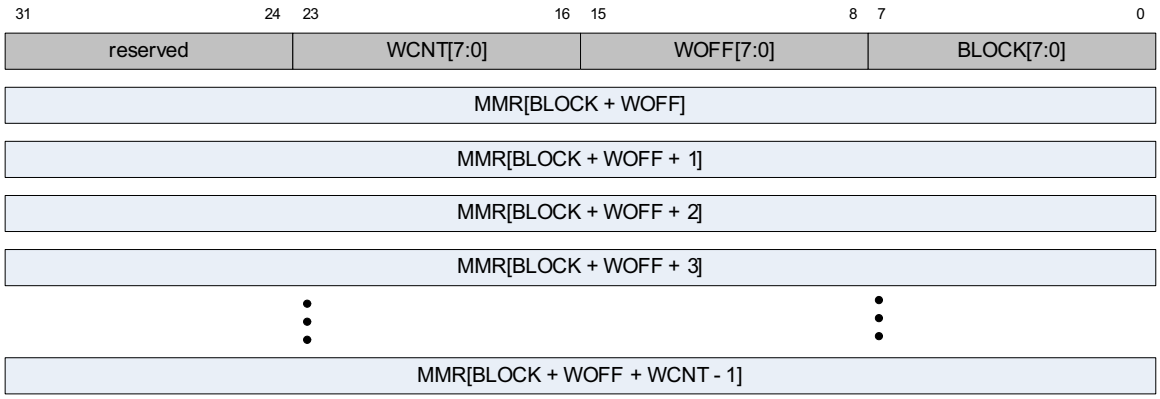


Figure 30-53: Block Configuration Structure

When the configuration DMA fetches a BCS, the BCH specifies the MMR destination of the subsequent configuration words. The 8-bit `BCH.BLOCK` field specifies to which PVP processing block the subsequent configuration words belong. This field uses the same PVP block id scheme as listed in *Configuring Pipe Structure*. The 8-bit `BCH.WOFF` field specifies the word offset versus that first register of the block's register space. The 8-bit `BCH.WCNT` field specifies how many registers are to be written to the specified register space. The `BCH.RESERVED` field must be filled with zeros.

Configuration DMA operations not limited to only fetching single BCSs. The configuration DMA operations also can fetch a series of BCSs at once. A series of BCHs is called block configuration list (BCL) and is illustrated in the block configuration list example figure.

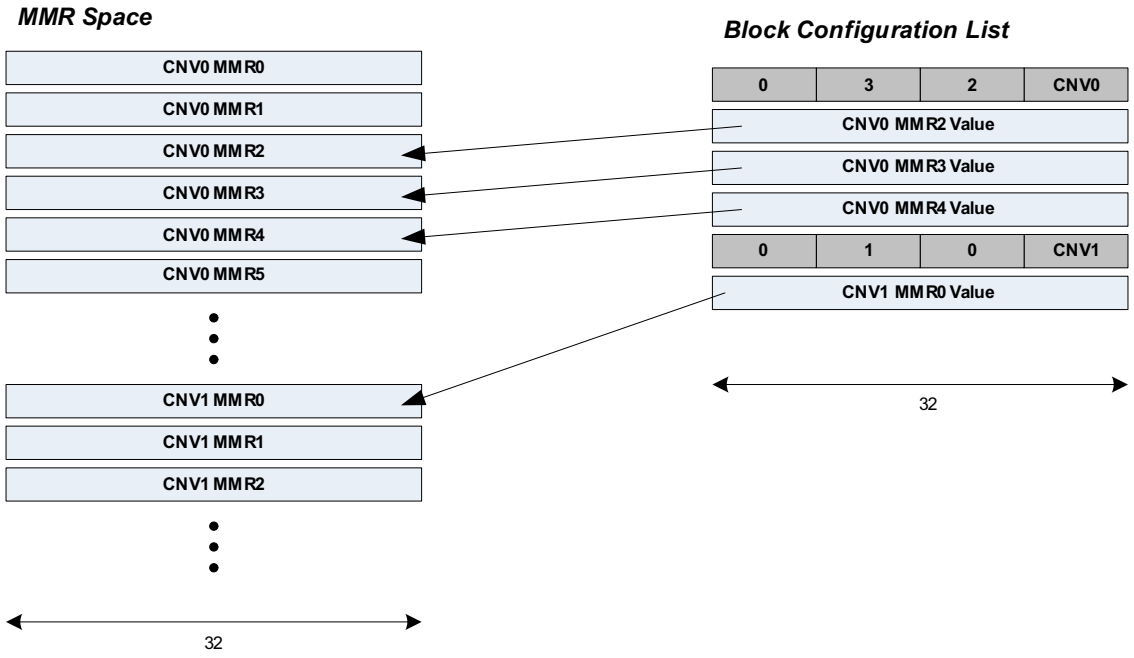


Figure 30-54: Block Configuration List Example

The example block configuration list consists of two BCSs. The first BCS targets processing element CNV0. This BCS contains three 32-bit configuration values to be written into three MMR registers that are contiguous in register space. The register space contains multiple registers dedicated to the CNV0 block. Because the example BCS does not target the first three memory mapped registers (MMRs) in CNV0 space (rather it targets the three from MMR2 on), the `BCH.WOFF` field is set to a value of 2. The second BCS targets CNV1. This BCS only writes to a single register. And, because it is the first register in CNV1 space, the `BCH.WOFF` field =0.

Fetching the Initial Configuration

After the camera pipe or memory pipe have been enabled by the `PVP_CTL.PVPEN` or `PVP_CTL.CPEN` bits, the respective input formatter (IPFn) immediately requests a block control structure list (BCL) fetch from the configuration DMA channel. If the configuration DMA does not grant the request because it is either not ready or not enabled, the pipe engine stalls until either the pipe is configured by memory mapped register (MMR) writes or the DMA starts granting. The IPFn blocks do not accept any data on their data input, until the pipe is fully configured and enabled.

When the configuration DMA is granted, the input formatters fetch BCL words until the `PVP_IPF0_CFG.START` or `PVP_IPF1_CFG.START` bit is set. The BCL is assumed to describe a valid pipe configuration and to write a 1 to the `PVP_xxx_CFG.START` bits of all involved blocks. The `START` bit can be seen as a self-clearing block enable bit. The self-clearing nature of this bit ensures that software does not need to perform garbage collection at or after pipe re-configuration. Blocks that are no longer used are automatically disabled.

The START bits of the input formatters have additional purpose also enable the entire camera or memory pipe. Therefore, while the order of BCSs inside a BCL does not matter, only the very last BCS writes into the PVP_IPF1_CFG register.

After the PVP_IPF0_CFG.START or PVP_IPF1_CFG.START bit has been written, the IPF0 starts accepting data from the video subsystem or IPF1 starts requesting data from data input DMA and the pipe starts processing.

Configuring with Descriptor-Based Method (Memory Pipe)

To enable memory pipe operation, set the PVP_CTL.PVPENand PVP_CTL.MPENbits. Additionally, the PVP_CTL.CLKDIVmust be configured properly. While the camera pipe is clocked by the data clock of the receiving PPI, the clock for the memory pipe is derived from system clock (*SCLK*). The PVP is designed for maximum operating frequency of f_{PVPCLK_MAX} MHz. For best performance:

- Clear PVP_CTL.CLKDIV if *SCLK*<= f_{PVPCLK_MAX} MHz, so the memory pipe clock equals *SCLK*
- Set PVP_CTL.CLKDIV if *SCLK*> f_{PVPCLK_MAX} MHz, so memory pipe clock equals *SCLK*/2

NOTE: Over clocking of memory pipe can lead to unpredictable behavior and thermal silicon defects. See the processor data sheet for the f_{PVPCLK_MAX} specification.

Before enabling a memory pipe job, configure the respective data input and data output DMA operations and enabled them immediately. Typically, the completion interrupt of the data output DMA is also enabled. This interrupt signals when the PVP memory pipe has completed the job and ensures that the results have been written back to memory.

A BCL structure is expected somewhere in system memory. This BCL structure can be statically pre configured or dynamically composed at run-time.

Finally, the configuration DMA is configured. The DMA_ADDRSTART registers point to the base address of the BCL. When the configuration DMA is enabled, the operation takes place autonomously. IPF1 starts requesting data from the data input DMA as soon as all BCL has been fetched (after the PVP_IPF1_CFG.START bit has been set). Software is alerted by the completion interrupt of the data output DMA connected to OPF3.

Configuring with Dynamic (on-the-fly) Method

To control how a pipe behaves after it has been initially configured, the input formatters use the frame counter (PVP_IPFn_FCNT) registers. If the values in these registers =0, the respective pipe repeatedly operates in the same mode as long as it is not disabled by the PVP_CTL.CPEN or PVP_CTL.MPEN bit. If PVP_IPFn_FCNT =1, the pipe operates on exactly one frame then stalls. If PVP_IPFn_FCNT is set to a value of *N*, exactly *N* frames are processed.

If PVP_IPFn_FCNT =0 or =1, the input formatters request a new BCL from configuration DMA every frame. If PVP_IPFn_FCNT>1, a new BCL is only requested every N^{th} frame. Similarly, status reports can only be transmitted at PVP_IPFn_FCNT frame boundaries.

Because the PVP is a pipelined engine, it takes *X*+*D* data clocks to process *X* data words, where *D* stands for the depth of the pipeline. The PVP is clocked by data, so (after *N* frames have been processed) a number of results (*D*) still stick in the pipe. These do not yet progress to the data output DMA channel because the pipe is waiting for further BCL instructions. The first data clocks of the future operation, then data implicitly clocks the older results out.

To automatically drain the remaining result words, set PVP_IPFn_PIPECTL.DRAIN =1. This configuration allows IPF0 (camera pipe) to pass further *D* data clocks into the pipe for this purpose. Keep Upstream PPI, PIXC, and video source devices enabled, so they can provide the required clocks. The IPF1 block (memory pipe) cannot rely on receiving further data clocks from data input DMA. Rather, if PVP_IPFn_PIPECTL.DRAIN = 1, IPF1 generates the required clocks itself artificially.

The PVP_IPFn_FCNT register and PVP_IPFn_PIPECTL.DRAIN bit partner to instruct the input formatter on how to control the pipe as shown in the operating modes by FCNT and DRAIN selections table.

Table 30-47: Operating Modes by FCNT and DRAIN Selections

FCNT	DRAIN	Mode
0	0	**Mode 0: Continuous Frame Mode** Frames are continuously requested by the IPFn and feed the PVP pipe. The Configuration DMA is constantly requesting BCL words every frame. If new Configuration is granted by DMA or by MMR writes, it is applied on the next frame boundary. Processing Blocks do not auto-disable. New Configuration must not set PVP_xxx_CFG.START bit of any block again.
>0	0	**Mode 1: Back-to-Back Mode** FCNT frames are requested by the IPFn, and these feed the pipe. The last data words get stuck in the pipe. The pipe stalls, and IPF x is requesting new instructions from configuration DMA. If a new configuration is granted by DMA or by MMR writes, the configuration is applied for the next FCNT set of frames. The first data clocks of the new set push the remaining words of the former set out to the data output DMA processes. Only IPFn blocks auto-disable, other processing blocks do not. A new configuration needs to reset START bit of IPFn, but must not set the START bit of the other blocks. Ideally, writes to PVP_xxx_CFG registers of PVP blocks other than IPFn are avoided altogether.

Table 30-47: Operating Modes by FCNT and DRAIN Selections (Continued)

FCNT	DRAIN	Mode
>0	1	**Mode 2: Auto-Completion Mode** FCNT frames are requested by the IPFn, and these feed the pipe. The last data words are automatically pushed out, and the operation automatically completes. If a new configuration is granted by DMA or by MMR writes, the configuration is applied for the next FCNT set of frames. All processing blocks auto-disable. A new configuration must set all wanted START bits again.
0	1	**Mode 3: Drain Now Instruction** This combination instructs the IPFn to flush all content out of the pipe. All processing blocks auto-disable. A new configuration must set all wanted START bits again.

Mode 0 (continuous frame mode) is popular for camera pipe operation. Often, the camera pipe is enabled statically, and updates to the coefficients are only required from time to time. On demand software may enable the configuration DMA in stop mode to trigger the fetch of one single BCL. In this mode, the PVP_IPFn_TAG register is useful to match status results with a respective configuration. The new BCL only updates coefficients of processing blocks. The BCL does not alter the pipe configuration.

Mode 1 (back-to-back mode) is useful in camera pipe mode when coefficients have to change every N^{th} frame. This mode supports on-the-fly reprogramming of coefficients, but it does not support on-the-fly reprogramming of pipe configurations. The following settings must not change unless the DRAIN bit is set:

- Pipe structure (PVP_xxx_CFG registers)
- Fundamental operating modes (such as 1st-derivative to 2nd-derivative mode of PEC)
- Input format and OPORT configuration (PVP_IPFn_CTL registers)
- Horizontal size of input frame (HCNT)

If any of the above settings change on-the-fly, the PVP_IPFn_PIPECTL.DRAIN bit must be set for proper operation to use Mode 2. In memory pipe mode, the PVP_IPFn_PIPECTL.DRAIN bit causes of few clock cycles of overhead depending to pipeline depth. In camera pipe mode, the PVP_IPFn_PIPECTL.DRAIN bit causes the loss of an entire frame. Operation stops after the pipe has been drained and resumes with the next incoming *VSYNC* signal.

Whenever the next PVP job stalls until the completion event of the data output DMA is flagged either as an interrupt or as a system trigger, setting the PVP_IPFn_PIPECTL.DRAIN bit is a requirement at the application level. The data output DMA work unit does not complete as long as related data sticks in the pipe. The data in the pipe does not progress until the configuration DMA and data source grant. Such deadlock situations can be avoided by using Mode 2. Once a deadlock occurs, it can be resolved by Mode 3 operation.

Mode 3 (drain now instruction) is more similar to a one-time instruction than an operating mode. This mode is used whenever software needs to respond to unpredictable events. For example, while the camera

pipe is operating in Mode 1, software notices that the future frames need to have alternating PVP_IPFn_HCNT settings. If the pipe is already stalled waiting for new BCL instructions, a BCL containing a drain now instruction can gracefully switch the pipe from Mode 1 into Mode 2. All pending results in the pipe are still properly shifted out before the pipe resumes operation with the new PVP_IPFn_HCNT settings after the subsequent BCL fetch.

The following list describes the difference between the modes in disabling the processing blocks after operation.

- In Mode 0, new configurations must not set the START bit of any block another time (not even the one of the input formatters). If a new write to the PVP_xxx_CFG register cannot be avoided, write the START bit to a zero value.
- In Mode 1, the hardware only disables the input formatters after PVP_IPFn_FCNT expires. All other blocks remain enabled. In principle, only the PVP_IPF0_CFG or PVP_IPF1_CFG register needs to be rewritten to set the START bit again to initiate another operation. New configurations must not set the START bit of any other block another time. If a new write to the PVP_xxx_CFG register cannot be avoided, the START bit shall be written by a zero value.
- In Mode 2 and Mode 3, all blocks are disabled by hardware after the drain operation. For future operation, re-write the PVP_xxx_CFG register of all included blocks to set the START bit again.

The PVP_STAT and PVP_ILAT registers report when the camera pipes (CPDRN bit) and the memory pipe (MPDRN bit) are drained. The ready bit, PVP_STAT.CPRDY and PVP_STAT.MPRDY, report whether PVP_IPF0_CFG.START and PVP_IPF1_CFG.STARTbits are ready to be set.

Working with Pipe Latency (Data Buffering)

Most of PVP processing blocks have internal data buffers, if not row buffers. Before a first result can be generated, these buffers need to be filled with valid input data, so that the mathematical operation can be performed. The amount of data buffering needed for the operation defines the latency that a block is adding to the chain if that block is inserted into a pipe. This latency equals the number of artificial clocks that need to be applied in the case of a drain operation. Due to their 2-dimensional nature, the CNVn, PEC, and UDS blocks have local row buffers where they intermediately store one or multiple data rows. The latency of these blocks is much higher than the latency of the IPFn, ACU, PMA, or OPFn blocks. The IIMn block is a special case, in that (although it performs 2-dimensional operations) this block can generate outputs early (because results do not depend on future data).

Table 30-48: Block Latencies

PVP Block	Latency [Data Clocks]
IPFn	5
OPFn	1
THCn	3
CNVn	2x HCNT + 14

Table 30-48: Block Latencies (Continued)

PVP Block	Latency [Data Clocks]
PMA	5
ACU	7
PEC	2x HCNT + 11 if 2nd derivative mode with ZCRSS=0, otherwise 1xHCNT + 6
IIMn	6
UDS	varies with settings

Configuring with Daisy Chain Method

Configuration and coefficient memory mapped registers (MMRs) of all processing blocks are double buffered. Whenever PVP_IPFn_FCNT is zero and the PVP_IPF0_CFG.START and PVP_IPF1_CFG.START bits =1, the values are copied from MMR registers into application buffers at the frame boundary. Due to the pipelined nature of the PVP, not all values are copied at the same time. Rather, the timing of value copying is closely related to how the VSYNC of the next frame progresses through the pipe. New settings apply to the first pixel of the new frame immediately after the last pixel of the old frame has been processed. The update command is daisy chained through the pipe, starting from the IPFn blocks through to the OPFn blocks.

While sequencing through the daisy chain operation, hardware clears the PVP_xxx_CFG.START bits of all involved processing blocks in MMR space. Write conflicts can occur if software attempts to write the PVP_xxx_CFG registers during the daisy chain operation.

The PVP_STAT register provides two status bits that report whether a daisy chain operation is ongoing in the camera pipes (PVP_STAT.CPDC) or in the memory pipe (PVP_STAT.MPDC). These bits are set along with the PVP_IPF0_CFG.START or PVP_IPF1_CFG.START bit and are cleared if the output formatters have been updated.

In camera pipe mode, the PVP_STAT.CPDC bit is cleared only after the daisy chain progresses through the furthest of the enabled OPFn blocks. During this time, software should not write the PVP registers. If software mistakenly does write to these registers, the event is reported by the PVP_STAT.CPWRERR and PVP_STAT.MPWRERR flags. While all status bits in the PVP_STAT register are self clearing, the PVP_ILAT register latches the events until cleared by a software handshake.

A PVP block which is enabled in either the camera pipe or memory pipe can be moved to the other pipe only after the drain done command is received for the former pipe. This restriction ensures that the PVP completely processes all the pixels in one pipe and disables the PVP block, before the PVP is enables the PVP block for pixel processing in the next pipe.

NOTE: Camera pipes and memory pipes can be configured by MMR writes or BCL fetches. These are separate control mechanisms and are not intended to be mixed for a given pipe during ongoing operation. Before switching from one method to the other, inspect the daisy chain and drain status bits in the PVP_STAT register to ensure no operation from the other method is pending.

Working with DMA Job Lists

The PVP memory pipe and camera pipe may be fed data continuously for sequential processing. The advantages of using DMA job lists are:

- Sequential DMA processing requires minimal overhead for the core.
- This DMA method provides high PVP performance.

A number of programming consideration influence how systems set up and maintain job lists for the multiple operations required for sequential processing. There are two approaches for operating these job lists. These are described in *Static DMA Job List Operation* and *Dynamic DMA Job List Operation*.

Note that the techniques presented in these sections assume that the data to be processed within the PVP memory pipe is completely stored in memory. This assumption is in contrast to typical camera pipe operation, in which the PVP processes data streams on the fly with data gated by an external (PIXEL) clock. In that case, the processing data is not fully loaded into memory, and additional measures are required to control the input data DMA to prevent running out of valid data. While this approach is technically possible, the techniques required are not included in the *Static DMA Job List Operation* or *Dynamic DMA Job List Operation* descriptions.

Also note that understanding and using these descriptions require basic knowledge of the processor's DMA functionality. It is especially useful to understand the application of linked list descriptors, 1-dimensional DMA operations, and 2-dimensional DMA operations. For more information about DMA, see the Direct Memory Access (DMA) chapter and the Trigger Routing Unit (TRU) chapter.

Whether static or dynamic, DMA job lists for PVP operations have some common features: *Job List Setup*, *Job List Global Trigger*, and *Job List Start*.

- **Job List Setup**

  To sequentially schedule multiple PVP memory pipe operations, correctly initiate and trigger the following data flows (each handled by a different DMA move engine).

  – DMA 45 moves the PVP memory pipe configuration (BCS block control structure list) of a specific PVP memory pipe job from memory into the PVP, configuring the processing functionality of the elements building the PVP memory pipe.

  – DMA 43 moves the data set to be processed from memory into IPF1 using this specific PVP memory pipe configuration.

  – DMA 42 moves the results of the PVP memory pipe operation (OPF3) to a destination, usually to memory.

  The PVP can optionally generate an additional status (report) data stream originating from IPF1, THC0, or THC1. This data stream also needs to be moved from the PVP memory pipe status output to a destination, (usually to memory) using DMA 44.

  To initiate the PVP memory pipe job list linked list, set up three DMA descriptors. Each linked descriptor stores the DMA configuration parameters within memory to maintain the DMA sequences

to handle the three data flows, BCS, data IN, data OUT, and optionally status data OUT. If the descriptors are all completed by the core, multiple PVP memory pipe jobs can be chained without any further core intervention as the DMA operations run through the chained lists and handle the multiple data flows autonomously. The following description assumes the memory pipe is inactive, (a non initialized state PVP_CTL.MPEN =0).

To run a PVP memory pipe job from the PVP memory pipe job list, the PVP first loads the memory pipe configuration into the respective shadow registers of the respective PCP elements to build the PVP memory pipe. The configuration within the shadow register is activated (transferred from shadow register to active register) using hardware triggers (staggered transfer). This configuration stream into the shadow registers is handled by DMA 45 which is controlled by a linked descriptor list. The last valid descriptor must contain a DMA_CFG.FLOW field =0 (STOP Mode) to gracefully stop the DMA 45 after completing the PVP memory pipe job list and avoid loading non valid data into PVP memory pipe configuration register. All DMA_CFG descriptor fields within the linked list have DMA_CFG.FLOW =6 or =7 (descriptor list mode) or alternatively DMA_CFG.FLOW =4 (descriptor array mode).

A 1-dimensional DMA operation is shown in the following figure. Note however, a 2-dimensional DMA operation can also be used.

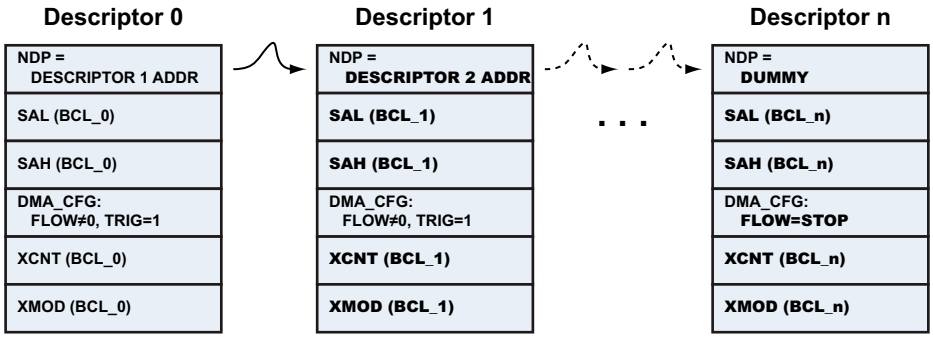


Figure 30-55: Static Linked Descriptor List for Configuration Data (BCL Data)

- **Job List Global Trigger**

  After the first initialization of the linked descriptor list the parameter registers of DMA 45 have to be initialized, particularly the DMA_DSCPTR_NXT and DMA_CFG register. The DMA 45 is started but then stalls until the PVP memory pipe is enabled and the PVP_IPFn_FCNT =0.

  To generate a unique synchronization signal that synchronizes the Data Out (DMA 42) and Data In (DMA 43) stream, all DMA_CFG descriptor fields of DMA 45 should have the bit field DMA_CFG.TRIG =1. This generates a trigger signal after a new configuration is loaded into the shadow registers.

- **Job List Start**

  After setting up the three DMA linked lists, the memory pipe has to be activated (PVP_CTL.MPEN =1).

Static DMA Job List Operation

This section describes how to setup and use static DMA job lists with the PVP memory pipe. The important points in this description are:

- Data out descriptor lists
- Data in descriptor lists
- Data flow in auto-completion mode
- Data flow in back-to-back mode

Data Out Descriptor Lists for Static Job Operation—After successfully setting up and starting the DMA 45 linked descriptor list for the configuration data of the PVP memory pipe, set up an additional DMA (DMA 42) to move data out (result data) from the PVP memory pipe (out of OPF3) into memory. A second linked descriptor list that handles the output data (result) is set up according to the previous description of the DMA_45 linked description list. Optionally, set up DMA 44 using an additional linked descriptor list that handles status data out. These data streams, the data out and the status data out, are handled by dedicated DMA processes. DMA 42 handles the data out stream, and DMA 44 handles the status data out stream.

NOTE: This setup for DMA job lists is described in *Working with DMA Job Lists*.

The data out streams are controlled by their respective linked descriptor list. The last valid descriptor of each linked list must contain a `DMA_CFG.FLOW` field =0 (STOP mode) to gracefully stop the DMA 42 and (optionally) DMA 45 after completing the PVP memory pipe job list. This graceful stop avoids loading non valid data into PVP memory pipe configuration register. All `DMA_CFG` descriptor fields within the linked list have `DMA_CFG.FLOW` =6 or =7 (descriptor mode) or alternatively `DMA_CFG.FLOW` =4 (descriptor array mode). Configure the TRU to route the trigger of DMA 45 to the input trigger of DMA 42 and DMA 44.

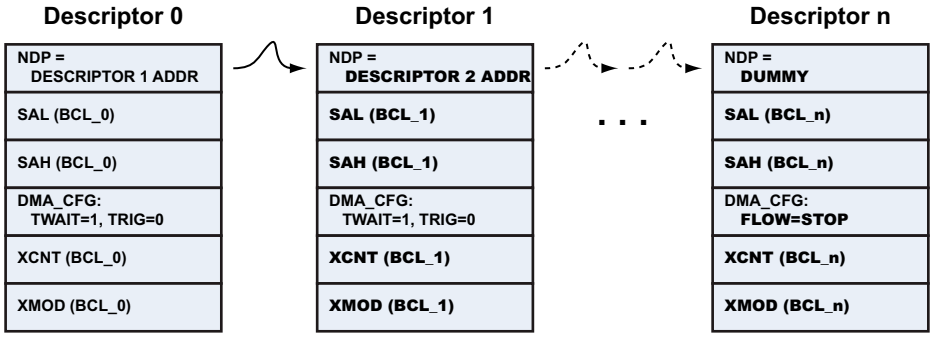


Figure 30-56: Static Linked Descriptor List for Data OUT (Result Data) and Optionally Status Data Out

Data In Descriptor List for Static Job Operation—After successfully setting up and starting DMA 42, DMA 45, and (optionally) DMA 44, set up the DMA 43 linked descriptor list for the input data (data in) to be moved into IPF1. A third linked descriptor list that handles the data in is set up according to the previous description of the DMA_42 linked description list. This data stream is handled by a dedicated DMA (DMA 43), which is controlled by a linked descriptor list. The last valid descriptor of the linked list must contain a `DMA_CFG.FLOW` field =0 (STOP mode) to gracefully stop the DMA 43 after completing the

PVP memory pipe job List. This graceful stop avoids loading non valid data into PVP memory pipe configuration register. All DMA_CFG descriptor fields within the linked list have DMA_CFG.FLOW =6 or =7 (descriptor list mode) or alternatively DMA_CFG.FLOW =4 (descriptor array mode). Configure the TRU to route the trigger of DMA 45 (back-to-back mode) or DMA 42 (auto-completion mode) to input trigger of DMA 43. All DMA_CFG descriptor field must therefore have a DMA_CFG.TWAIT field =1.

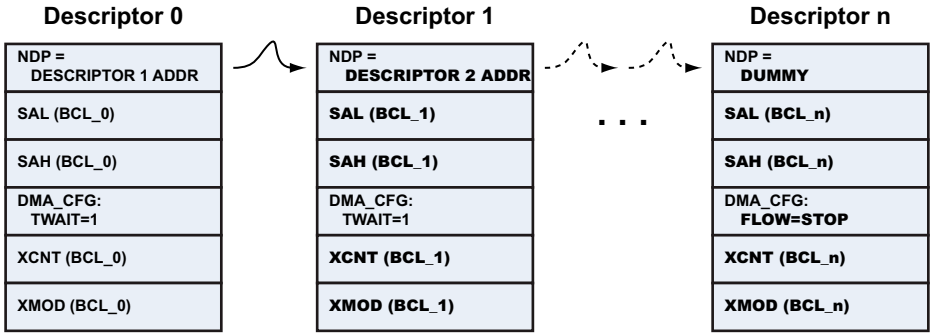


Figure 30-57: Static Linked Descriptor List for Data IN

Data Flow in Auto-Completion Mode for Static Job Operation—The following figure and procedure shows the data flow using a static job list and auto-completion mode.

1. DMA45 generates a trigger for DMA 43
2. DMA 43 generates a trigger for DMA 42
3. Software generates the first trigger for DMA 42
4. DMA45 is triggered when the memory pipe requests a new configuration. In this case the PVP memory pipe is enabled and IPF1's PVP_IPFn_FCNT field =0 (memory pipe request)

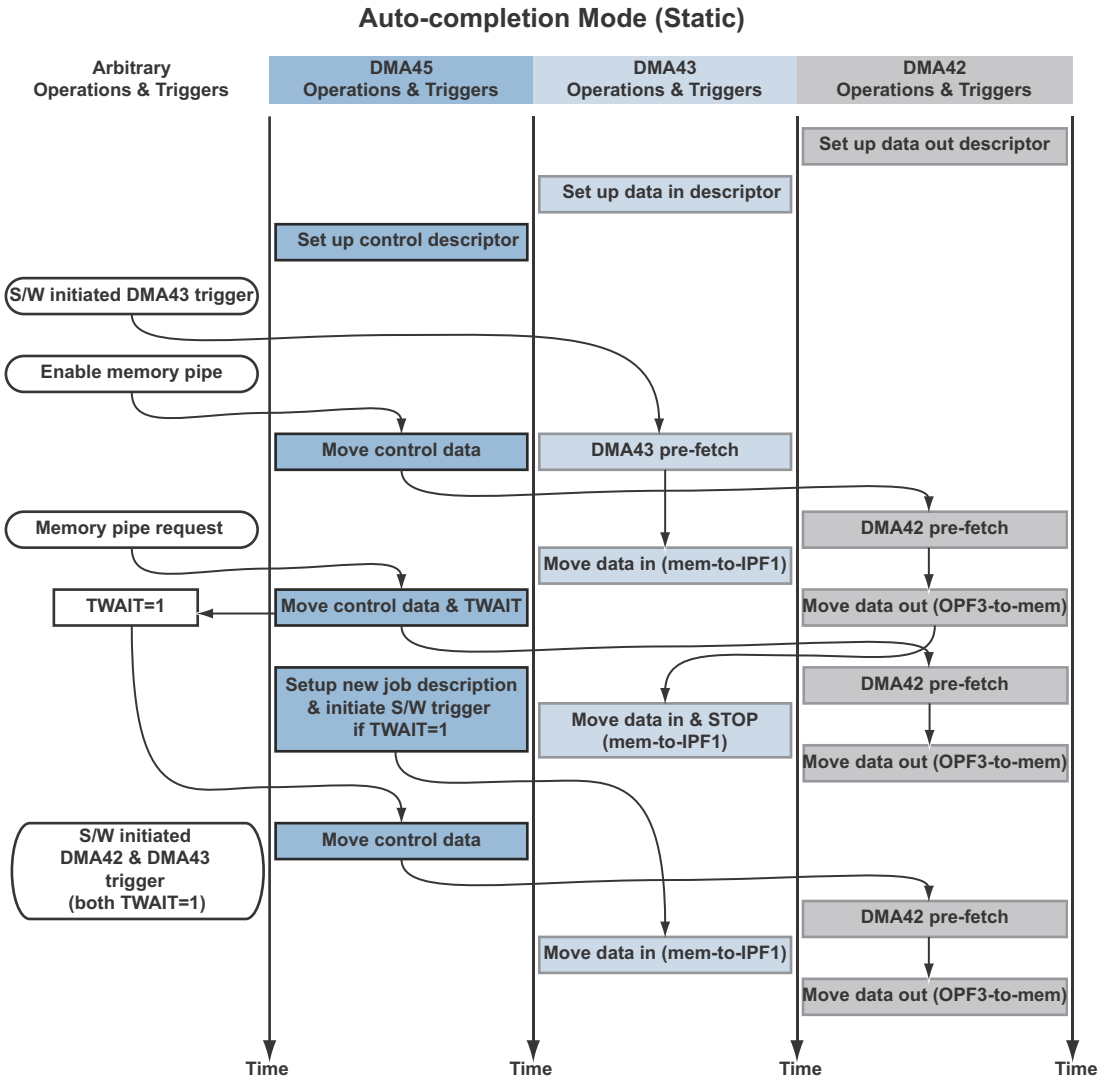


Figure 30-58: Static Job List Data Flow (Auto-Completion Mode)

Data Flow in Back-to-Back Mode for Static Job Operation—The following figure shows the data flow using a static job list and back-to-back mode. The DMA45 generates a trigger for DMA 43 and DMA 42. DMA45 in return is triggered by the memory pipe requesting a new configuration. In this case the PVP memory pipe is enabled and IPF1's `PVP_IPFn_FCNT`field =0 (memory pipe request).

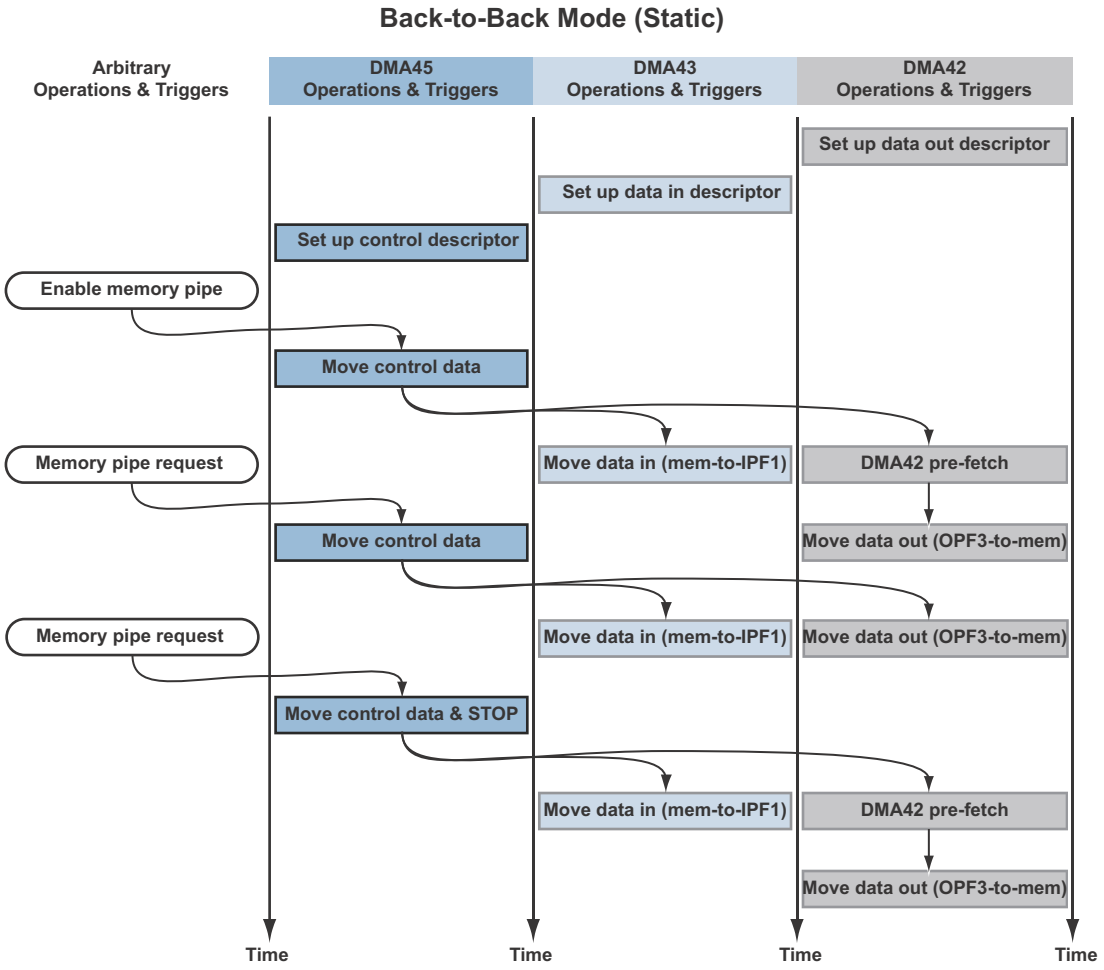


Figure 30-59: Static Job List Data Flow (Back-to-Back Mode)

Dynamic DMA Job List Operation

The setup for DMA job lists is described in *Working with DMA Job Lists*, and the DMA channel operations for these job lists is described in *Static DMA Job List Operation*. This section builds on those concepts and adds information for using dynamic operations. The important points in this description are:

- IPF1 mode selection
- Data in descriptor lists
- Data flow in auto-completion mode
- Data flow in back-to-back mode

IPF1 Mode Selection for Dynamic Job Operation—When the memory pipe job list is started before the creation of the PVP memory pipe job list is completed by the core (dynamic creation of PVP memory pipe jobs) the following issues must be considered.

- Use IPF1 mode 1 (back-to-back mode) is for maximum performance of the job list. This mode moves data of a new job into the PVP memory pipe, while the data of the previous job has not been completely moved out of the PVP memory pipe. This overlap is due to memory pipe latency and the output buffer within OPF3. To avoid conflict, there must not be any data dependency between input and output data (no recursive data processing). Further, the PVP memory pipe structure must not be modified, because the staggered loading of the configuration may not work correctly. Yet another challenge related to this mode is the fact that the last job cannot be completed. So, "dummy" data must be pushed explicitly into the PVP memory pipe to shift out the last results.
- Using IPF1 mode 2 (auto-completion mode), a job list may alternatively use dummy data (bubbles) that are implicitly (automatically) appended to every data set to entirely empty the PVP memory pipe before the new data are processed. Due to the bubbles moving through the PVP memory pipe, this mode decreases the performance of the PVP, but allows fully interlocked pipelined processing without the need for special care of control conflicts. Mixing both modes is usually not possible, because it requires dynamic modifications of the IPF1 configuration and DMA trigger configuration.

NOTE: Special care has to be exercised when using back-to-back mode if *either* the PVP memory pipe job list has started before the creation of the PVP memory pipe job list has been completed by the core *or* the data to be processed within the PVP memory pipe job list is not yet available in memory. Auto-completion mode is the preferable mode in these cases.

NOTE: Special care has to be exercised when the BCS includes `PVP_xxx_CFG` register updates, (with exception of `PVP_IPF1_CFG`). This requires processing the PVP memory pipe job list in mode 2 (auto-completion mode) as the staggered configuration load of the pipeline elements may no longer work correctly because the position of an element within the pipeline may change. So, a different staggering order may be required

Control Data Flow for Dynamic Job Operation—Support for dynamic job lists requires that the last valid descriptor (descriptor n) must contain a `DMA_CFG.TWAIT` =1 (halt and wait for incoming trigger) instead of containing `DMA_CFG.FLOW` =0 (stop mode) in order to pause and not stop the DMA channel and avoid loading non valid data (job creation not finalized) into the PVP memory pipe configuration register. An additional last descriptor (descriptor m) is reserved in memory, but only contains dummy values and a `STOP` mode in the `DMA_CFG` field to handle erroneous descriptor overrun and enable dynamic memory allocation.

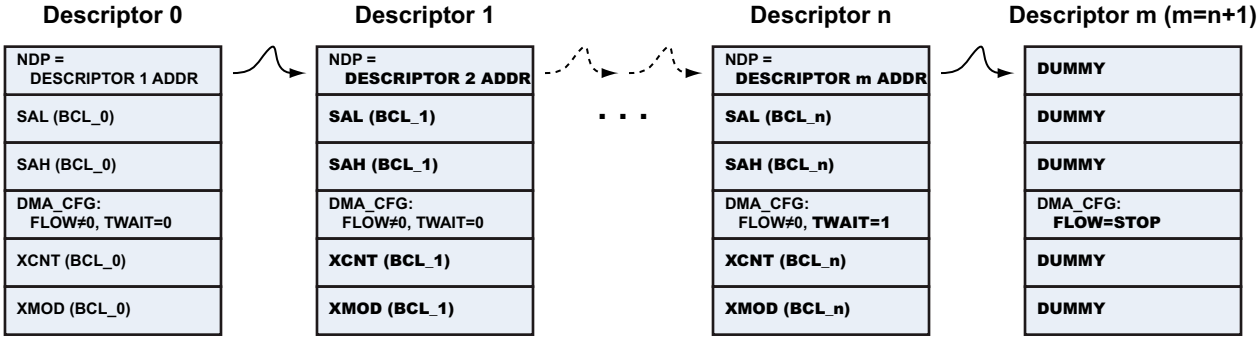


Figure 30-60: Linked Descriptor List for Configuration Data (BCL Data) Enabling Dynamic Extension

If adding an additional job to an existing PVP memory pipe job, this new job descriptor replaces the DUMMY descriptor (descriptor m) in memory. This updated descriptor m must now link to a new DUMMY descriptor (descriptor j).

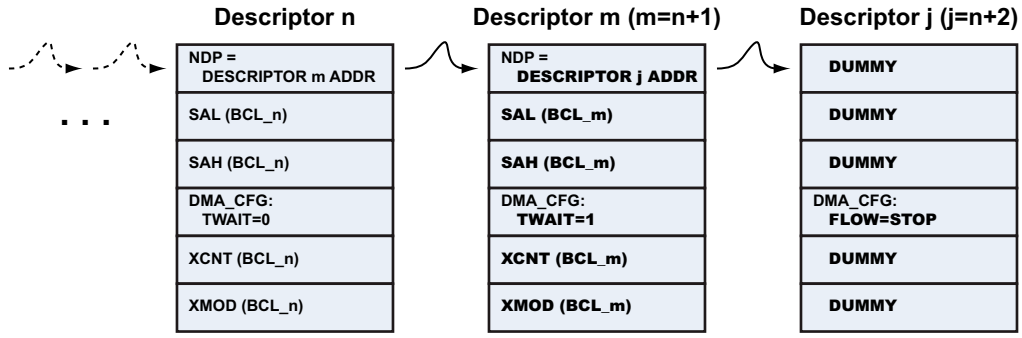


Figure 30-61: Dynamic Extending Linked Descriptor List for Configuration Data (Job List)

After completing the descriptor m and the job list extension that handles data in and data out, modify (clear) the `DMA_CFG.TWAIT` field of descriptor n. To grant a seamless DMA operation where:

- The DMA has already loaded the unmodified descriptor n.
- And, The DMA has already copied the `DMA_CFG.TWAIT` =1 descriptor (original descriptor n) into the `DMA_CFG` register.

Generate a software generated trigger for DMA 45 (and DMA 43 if in back-to-back mode). Therefore if `DMA_CFG.TWAIT` =1, a trigger is generated within the trigger routing unit (TRU) by software.

Additionally, to grant a seamless DMA operation using auto-completion mode where:

- DMA 42 has already loaded the unmodified descriptor n.
- And, DMA 42 has already copied the `DMA_CFG.TWAIT` = 1 descriptor (descriptor n) into the `DMA_CFG` register.

Generate a software generated trigger for DMA 43. Therefore if `DMA_CFG.TWAIT` = 1, a trigger is generated within the TRU by software.

Take special care if writing the DMA_CFG.TWAIT field of the DMA_CFG field and if reading the DMA_CFG registers for DMA 45 and DMA 42 takes longer than the DMA operation loading the configuration data.

NOTE: Alternatively, instead of setting DMA_CFG.TWAIT =1, use DMA_CFG.FLOW =0 as a semaphore (STOP mode). Set the DMA_CFG register (write operations) for DMA 45 to descriptor mode if the DMA stopped.

Data Out Flow for Dynamic Job Operation—The following situations describe handling data out data streams using back-to-back mode or auto-completion mode. Mixing modes is not possible because it requires modifications of the IPF1 configuration and the DMA trigger configuration.

- Use mode 1 (back-to-back mode) for maximum performance. Note that this mode moves data to be processed by a new job into the IPF1 while the resulting data processed by the previous job has not been completely moved out of the PVP memory pipe. This is because there may not be any data dependency between the input and output data (recursive data processing) in the output buffer within OPF3, as well as PVP memory pipe latency. Further, the PVP memory pipe structure may not be modified because the staggered loading of the configuration may not work correctly. A further challenge of this mode is it requires "dummy" data to be pushed explicitly into the PVP memory pipe to shift out the last results and therefore the last job cannot be completed.
- Use mode 2 (auto-completion mode) for fully interlocked pipelined processing without care for data hazards (read after write hazard). This mode appends dummy data (bubbles) implicitly (automatically) to every data set to empty the PVP memory pipe before the new data is processed in IPF1. Because the bubbles move through the PVP memory pipe, this mode decreases the performance of the PVP.

Special care must be exercised when using back-to-back mode if the PVP memory pipe job list has been started before the creation of the PVP memory pipe job list has been completed by the core, or if the data to be processed within the PVP memory pipe job list is not yet available in memory. Auto-completion mode should be used in these cases.

Data Out Flow for Dynamic Job Operation in Back-to-back Mode—Dynamic job lists require that the last valid descriptor (descriptor n) for DMA 42 (and optionally DMA 44) to contain a DMA_CFG.TWAIT field =1 (halt and wait for incoming trigger). This is so the DMA channel pauses instead of stops, and avoids loading non valid data (job creation not finalized) into the PVP memory pipe configuration register. An additional last descriptor (descriptor m) is reserved in memory which contains dummy values and a DMA_CFG.FLOW =0 (STOP mode) to handle erroneous overrun.

If adding an additional job to an existing PVP memory pipe job, the new job descriptor replaces the DUMMY descriptor (descriptor m) in memory. This updated descriptor m must now link to a new DUMMY descriptor (descriptor j).

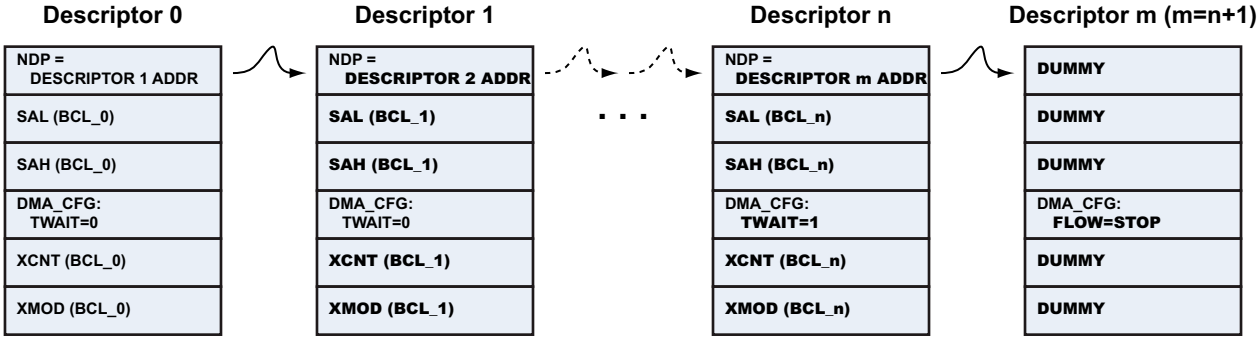


Figure 30-62: Linked Descriptor List for Data OUT (Results) Enabling Dynamic Extension (Back-to-Back Mode)

After completing the descriptor m and the job list extension to data in and data out, clear the `DMA_CFG.TWAIT` field of descriptor n.

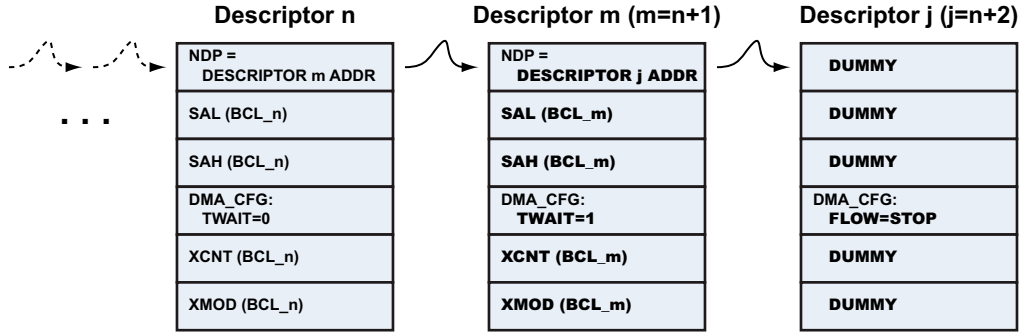


Figure 30-63: Dynamic Extending Linked Descriptor List for Data OUT (Job List) in Back-to-Back Mode

Data Out Flow for Dynamic Job Operation in Auto-Completion Mode—In auto-completion mode the DMA handling the data out (DMA 42) triggers (synchronizes) the DMA handling the data in (DMA 43). Therefore DMA 42 has to generate a respective trigger. Configure the TRU to route the trigger of DMA 42 to the input trigger of DMA 43.

All descriptors must have `DMA_CFG.TRIG` =1 to enable trigger generation. Only the last valid descriptor has `DMA_CFG.TRIG` =0 which puts the DMA 43 into pause state.

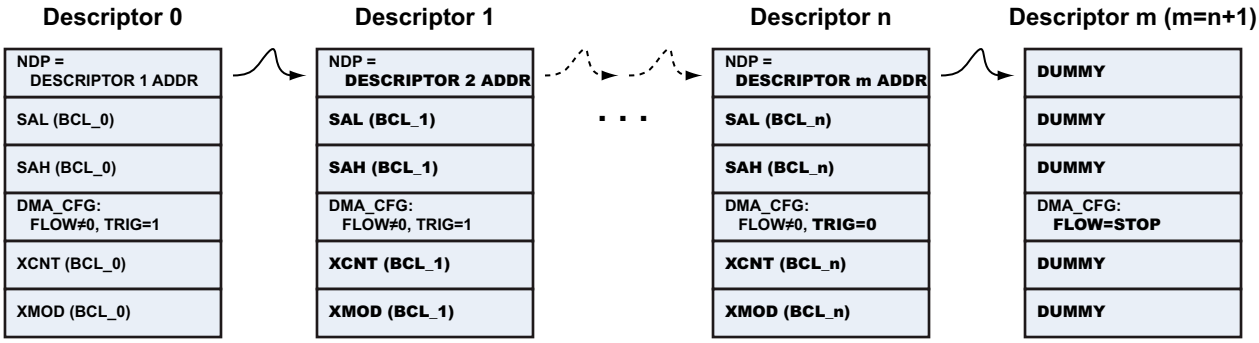


Figure 30-64: Linked Descriptor List for Data OUT (Results) Enabling Dynamic Extension (Auto-Completion Mode)

If adding an additional job to an existing PVP memory pipe job, this new job descriptor replaces the DUMMY descriptor (descriptor m) in memory. This updated descriptor m must now link to a new DUMMY descriptor (descriptor j).

After completing the descriptor m and the job list extension to data in and data out, change the setting of descriptor n to DMA_CFG.TRIG =1.

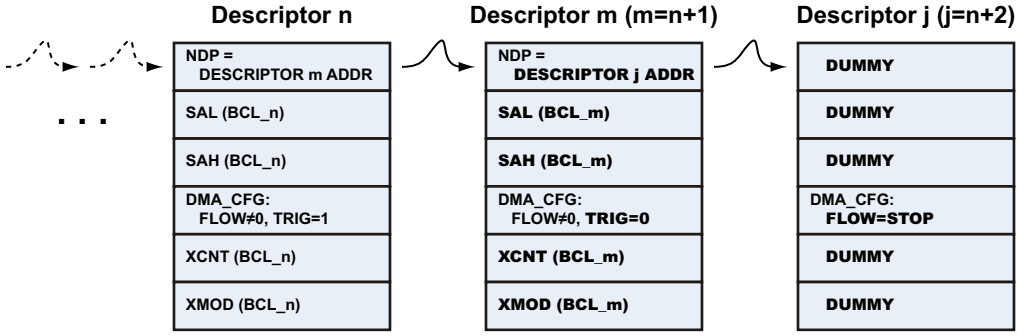


Figure 30-65: Dynamic Extending Linked Descriptor List for Data OUT (Job List) in Auto-Completion Mode

Data In Flow for Dynamic Job Operation—Dynamic job lists require that the last valid descriptor (descriptor n) for DMA 43 contain a DMA_CFG.TWAIT field =1 (halt and wait for incoming trigger) instead of DMA_CFG.FLOW =0 (STOP mode). This is so the DMA channel pauses instead of stops, and avoids loading non valid data (job creation not finalized) into the PVP memory pipe configuration register. An additional last descriptor (descriptor m) is reserved in memory which contains dummy values and a DMA_CFG.FLOW =0 (STOP mode) to handle erroneous overruns.

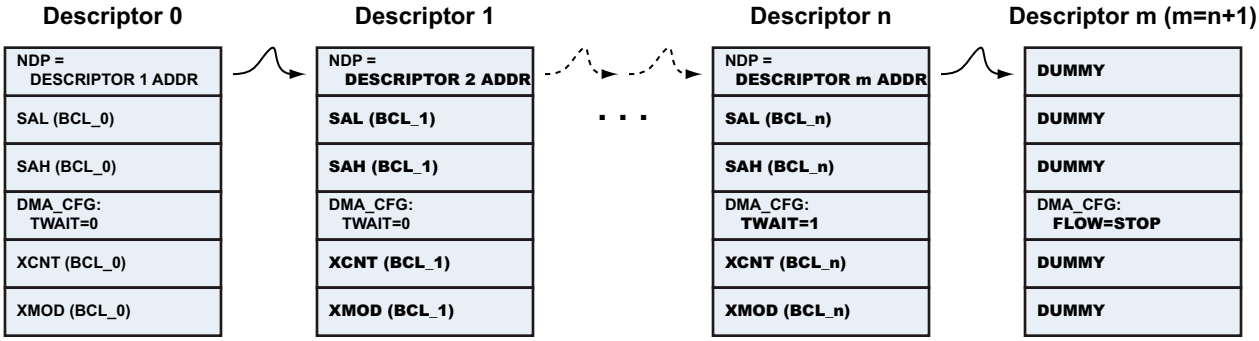


Figure 30-66: Linked Descriptor List for Data IN Enabling Dynamic Extension

If adding an additional job to an existing PVP memory pipe job, this new job descriptor has to replace the DUMMY descriptor (descriptor m) in memory. This updated descriptor m must now link to a new DUMMY descriptor (descriptor j).

Auto-completion mode and back-to-back mode require identical Data Out descriptor list.

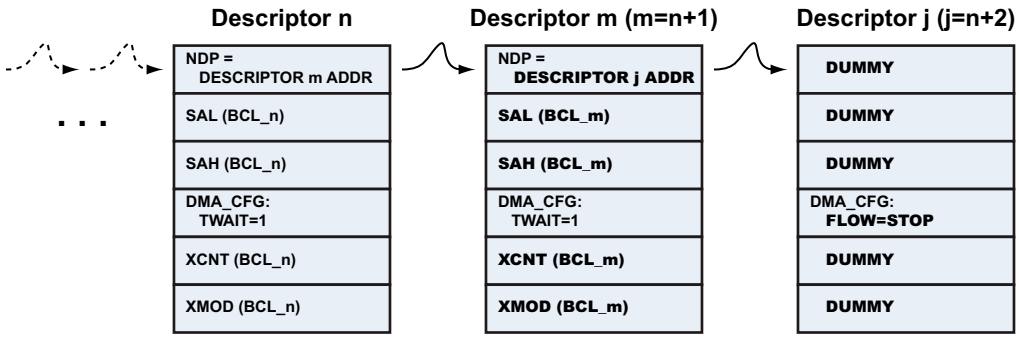


Figure 30-67: Dynamic Extending Linked Descriptor List for Data IN

Data Flow for Dynamic Job Operation in Back-to-Back Mode—The following figure shows the data flow using a dynamic job list and back-to-back mode. This figure illustrates that job creation processed by the processor core when it is slower than the PVP. The PVP reaches the end of the valid job list before the processor core generates a new job.

The software reads `DMA_CFG.TWAIT` of DMA45 and as a consequence generates a trigger by software to start the new job in DMA45.

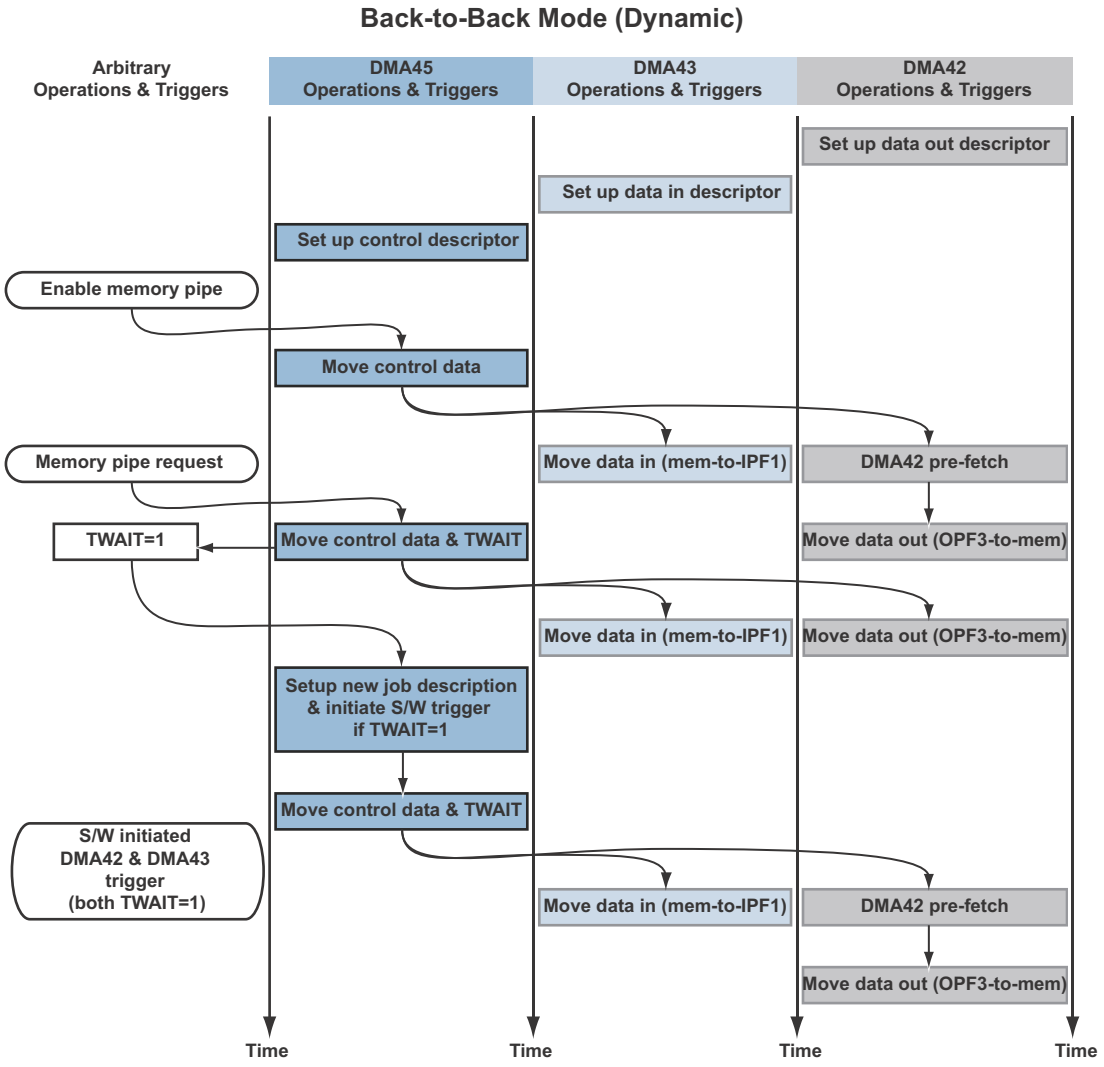


Figure 30-68: Dynamic Job List Data Flow (Back-to-Back Mode)

Data Flow for Dynamic Job Operation in Auto-Completion—The following figure shows the data flow using a dynamic job list and auto-completion mode. This figure and procedure demonstrate that job creation processed by the CPU is slower than the PVP processing the valid job list. Therefore, the PVP reaches the end of the valid job list before the processor core generates a new job.

1. The software reads `DMA_CFG.TWAIT` of DMA45 and as a consequence generates a trigger to start the new job in DMA45.
2. The software then reads `DMA_CFG.TWAIT` of DMA43 and as a consequence generates a trigger to start the new job in DMA42.

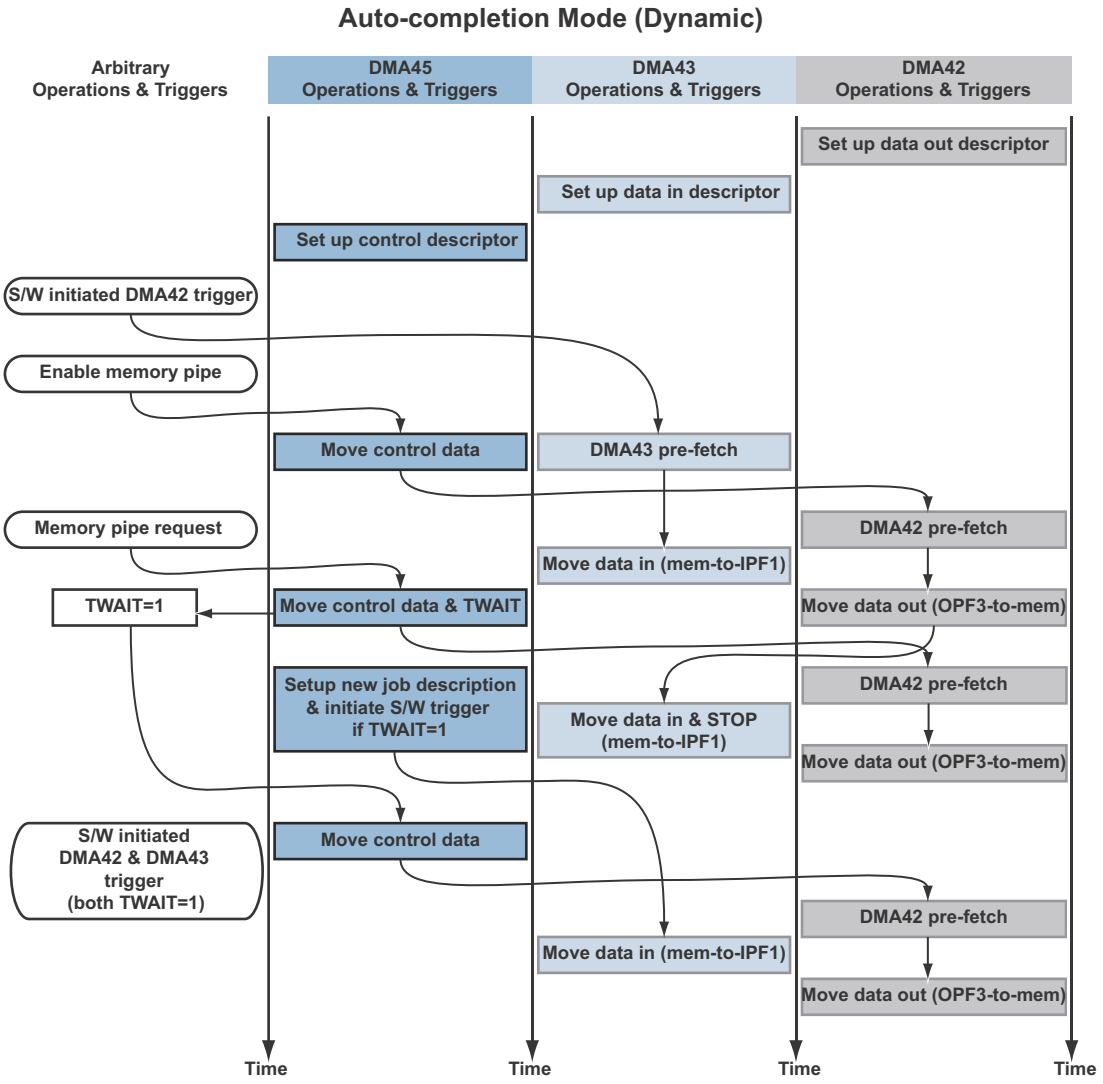


Figure 30-69: Dynamic Job List Data Flow (Auto-Completion Mode)

Working with Status (Histogram) Reports

Beside the PVP_STAT register (which flags synchronization events and errors) the PVP also provides an automatic method to output status reports (primarily to output the histogram results). The THCn blocks latch the status of their histogram counters into memory mapped register (MMR) space every frame. Software can rely on histogram ready events as signaled by the PVP_STAT.THC0RDY and PVP_STAT.THC1RDY status flags, then manually collect the numerous MMR registers. To ensure consistency of the reports, the MMR reads can be performed by an atomic procedure ensuring interrupts cannot interfere.

The PVP features two status output DMA channels that output the histogram reports more easily. After the channels are configured and enabled, the status DMA operate independently from software. Report consistency and proper read-out timing is ensured by hardware.

One status DMA is associated with the camera pipe, and the other status DMA is associated with the memory pipe. The feature is enabled by the PVP_IPFn_PIPECTL.STATEN control bits. If enabled, a status report is sent every time PVP_IPFn_FCNT counts to zero. If PVP_IPFn_FCNT has been programmed to zero, the report is sent after every frame.

If PVP configuration is frequently updated on the fly, system software might be exposed to risk to lose track of which configuration a specific report belongs to. For this purpose the input formatters feature the PVP_IPFn_TAG registers. At configuration time any 16-bit tag can be loaded into these registers. If PVP_IPFn_FCNT expires, the tag value progresses to the PVP_IPFn_TAG_STAT counterpart register in the MMR space. It can then optionally be stored via DMA paired with the histogram results.

Status Word Counters

When a report is sent to status output DMA the content of the report can be structured by the program. Only the IPFn and THCn modules can contribute to the report as follows.

- IPF0 can send only to camera pipe status DMA
- IPF1 only can send to memory pipe status DMA
- THC0 and THC1 always send the status to the status DMA of the pipe they belong to

If the PVP_IPFn_PIPECTL.STATEN status enable bit is set, the PVP_IPF0_CFG.STATWCNT and PVP_IPF1_CFG.STATWCNT fields control how many status words the respective input formatter sends to the status channel. Since the input formatters feature only one status word contained in the PVP_IPFn_TAG_STAT register, only two values are valid for the PVP_IPF0_CFG.STATWCNT or PVP_IPF1_CFG.STATWCNT fields. If =0, the input formatter does not contribute to the status report. If =1, the input formatter sends one 32-bit word, the 32-bit padded PVP_IPFn_TAG_STAT register to the status channel.

NOTE: Because the STATWCNT field resides in the PVP_IPFn_CFG register, its value cannot be altered without setting the corresponding DRAIN bit =1. The STATEN bit resides in the same register as the DRAIN bit. So, the structure of the status report must not change while DRAIN =0. STATWCNT is allowed to alter the STATEN bit with every configuration, even when DRAIN =0.

The threshold blocks feature many more status registers. Each features a histogram frame counter register (PVP_THCn_HFCNT_STAT), histogram counter registers (PVP_THCn_HCNT0_STAT to PVP_THCn_HCNT15_STAT), and a run-length reports per frame register (PVP_THCn_RREP_STAT).

- If the PVP_THCn_CFG.STATWCNT field =0, the respective threshold block does not contribute to the status report.
- If the PVP_THCn_CFG.STATWCNT field =1, only the PVP_THCn_HFCNT_STAT register is sent.
- If the PVP_THCn_CFG.STATWCNT field =9, the PVP_THCn_HFCNT_STAT and PVP_THCn_HCNT0_STAT to PVP_THCn_HCNT7_STAT registers are sent.
- If the PVP_THCn_CFG.STATWCNT field =17, the PVP_THCn_HFCNT_STAT and PVP_THCn_HCNT0_STAT to PVP_THCn_HCNT15_STAT registers are sent.

- If the PVP_THCn_CFG.STATWCNT field =18, the PVP_THCn_HFCNT_STAT and PVP_THCn_HCNT0_STAT to PVP_THCn_HCNT15_STAT, and PVP_THCn_RREP_STAT registers are sent.

The STATWCNT field of all processing blocks other than IPFs and THCs are reserved and must be initialized with zero values.

Status reports only output status registers. There is no programmable offset as in the case of block configuration.

Block Status Structure

When a processing block contributes to a status report, its data words are headed by a block status header (BSH). The BSH is similar to the block configuration header (BCH). It consists of the block ID value and also has a word count field. The latter mirrors the STATWCNT value as specified in the block configuration register. Unlike BCHs the BSHs do not have an offset field. The entity of Block Status Header and respective status words is called Block Status Structure (BSS). A BSS always consists of STATWCNT+1 32-bit words.

A status report consists of a list of block status structures. Such a list is referred to as block status list (BSL). An example is shown in the BSL example figure. This example assumes that both threshold blocks are configured in camera pipe mode. IPF0 and THC0 blocks are configured to send status to the report. The THC1 block is not configured to send status, so its PVP_THCn_CFG.STATWCNT field =0.

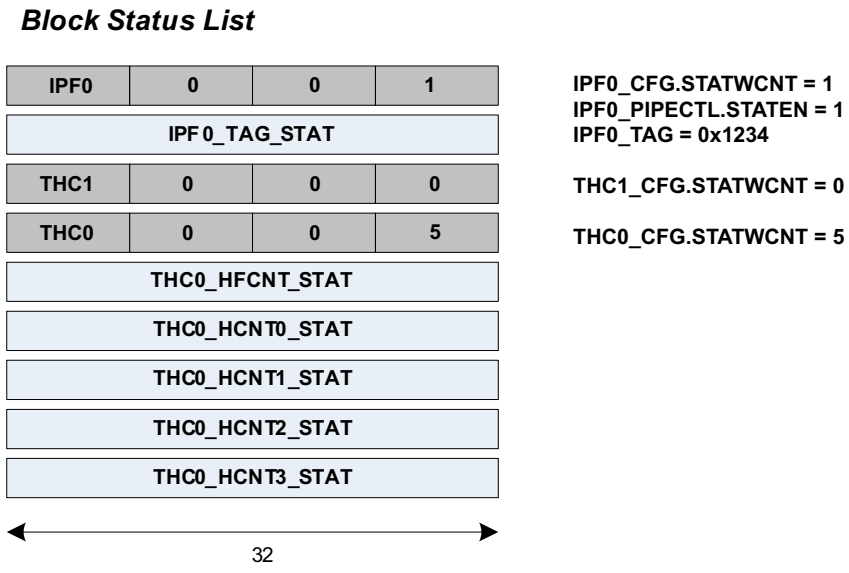


Figure 30-70: BSL Example

The input formatters make always the first entries in the block status list. If both threshold blocks are enabled in the same pipe, the one that comes first is closer to the IPFn in the pipe arrangement. In case the pipe configuration is not known, the block ID in the BSHs can be consulted.

Although it is not a hardware requirement, it is recommended that the work unit transfer count of the DMA associated with the status output matches the number of 32-bit words in the BSL.

ADSP-BF60x PVP Register Descriptions

PVP (PVP) contains the following registers.

Table 30-49: ADSP-BF60x PVP Register List

Name	Description
PVP_CTL	Control
PVP_IMSKn	Interrupt Mask n
PVP_STAT	Status
PVP_ILAT	Interrupt Latch Status n
PVP_IREQn	Interrupt Request n
PVP_OPFn_CFG	OPFn (Camera Pipe) Configuration
PVP_OPFn_CTL	OPFn (Camera Pipe) Control
PVP_OPF3_CFG	OPF3 (Memory Pipe) Configuration
PVP_OPF3_CTL	OPF3 (Memory Pipe) Control
PVP_PEC_CFG	PEC Configuration
PVP_PEC_CTL	PEC Control
PVP_PEC_D1TH0	PEC Lower Hysteresis Threshold
PVP_PEC_D1TH1	PEC Upper Hysteresis Threshold
PVP_PEC_D2TH0	PEC Weak Zero Crossing Threshold
PVP_PEC_D2TH1	PEC Strong Zero Crossing Threshold
PVP_IIMn_CFG	IIMn Configuration
PVP_IIMn_CTL	IIMn Control
PVP_IIMn_SCALE	IIMn Scaling Values
PVP_IIMn_SOVF_STAT	IIMn Signed Overflow Status
PVP_IIMn_UOVF_STAT	IIMn Unsigned Overflow Status

Table 30-49: ADSP-BF60x PVP Register List (Continued)

Name	Description
PVP_ACU_CFG	ACU Configuration
PVP_ACU_CTL	ACU Control
PVP_ACU_OFFSET	ACU SUM Constant
PVP_ACU_FACTOR	ACU PROD Constant
PVP_ACU_SHIFT	ACU Shift Constant
PVP_ACU_MIN	ACU Lower Sat Threshold Min
PVP_ACU_MAX	ACU Upper Sat Threshold Max
PVP_UDS_CFG	UDS Configuration
PVP_UDS_CTL	UDS Control
PVP_UDS_OHCNT	UDS Output HCNT
PVP_UDS_OVCNT	UDS Output VCNT
PVP_UDS_HAVG	UDS HAVG
PVP_UDS_VAVG	UDS VAVG
PVP_IPF0_CFG	IPF0 (Camera Pipe) Configuration
PVP_IPFn_PIPECTL	IPFn (Camera/Memory Pipe) Pipe Control
PVP_IPFn_CTL	IPFn (Camera/Memory Pipe) Control
PVP_IPFn_TAG	IPFn (Camera/Memory Pipe) TAG Value
PVP_IPFn_FCNT	IPFn (Camera/Memory Pipe) Frame Count
PVP_IPFn_HCNT	IPFn (Camera/Memory Pipe) Horizontal Count
PVP_IPFn_VCNT	IPFn (Camera/Memory Pipe) Vertical Count
PVP_IPF0_HPOS	IPF0 (Camera Pipe) Horizontal Position
PVP_IPF0_VPOS	IPF0 (Camera Pipe) Vertical Position

Table 30-49: ADSP-BF60x PVP Register List (Continued)

Name	Description
PVP_IPFn_TAG_STAT	IPFn (Camera/Memory Pipe) TAG Status
PVP_IPF1_CFG	IPF1 (Memory Pipe) Configuration
PVP_CNVn_CFG	CNVn Configuration
PVP_CNVn_CTL	CNVn Control
PVP_CNVn_C00C01	CNVn Coefficients 0,0 and 0,1
PVP_CNVn_C02C03	CNVn Coefficients 0,2 and 0,3
PVP_CNVn_C04	CNVn Coefficient 0,4
PVP_CNVn_C10C11	CNVn Coefficients 1,0 and 1,1
PVP_CNVn_C12C13	CNVn Coefficients 1,2 and 1,3
PVP_CNVn_C14	CNVn Coefficient 1,4
PVP_CNVn_C20C21	CNVn Coefficients 2,0 and 2,1
PVP_CNVn_C22C23	CNVn Coefficients 2,2 and 2,3
PVP_CNVn_C24	CNVn Coefficient 2,4
PVP_CNVn_C30C31	CNVn Coefficients 3,0 and 3,1
PVP_CNVn_C32C33	CNVn Coefficients 3,2 and 3,3
PVP_CNVn_C34	CNVn Coefficient 3,4
PVP_CNVn_C40C41	CNVn Coefficients 4,0 and 4,1
PVP_CNVn_C42C43	CNVn Coefficients 4,2 and 4,3
PVP_CNVn_C44	CNVn Coefficient 4,4
PVP_CNVn_SCALE	CNVn Scaling Factor
PVP_THCn_CFG	THCn Configuration
PVP_THCn_CTL	THCn Control

Table 30-49: ADSP-BF60x PVP Register List (Continued)

Name	Description
PVP_THCn_HFCNT	THCn Histogram Frame Count
PVP_THCn_RMAXREP	THCn Max RLE Reports
PVP_THCn_CMINVAL	THCn Min Clip Value
PVP_THCn_CMINTH	THCn Clip Min Threshold
PVP_THCn_CMAXTH	THCn Clip Max Threshold
PVP_THCn_CMAXVAL	THCn Max Clip Value
PVP_THCn_TH0	THCn Threshold Value 0
PVP_THCn_TH1	THCn Threshold Value 1
PVP_THCn_TH2	THCn Threshold Value 2
PVP_THCn_TH3	THCn Threshold Value 3
PVP_THCn_TH4	THCn Threshold Value 4
PVP_THCn_TH5	THCn Threshold Value 5
PVP_THCn_TH6	THCn Threshold Value 6
PVP_THCn_TH7	THCn Threshold Value 7
PVP_THCn_TH8	THCn Threshold Value 8
PVP_THCn_TH9	THCn Threshold Value 9
PVP_THCn_TH10	THCn Threshold Value 10
PVP_THCn_TH11	THCn Threshold Value 11
PVP_THCn_TH12	THCn Threshold Value 12
PVP_THCn_TH13	THCn Threshold Value 13
PVP_THCn_TH14	THCn Threshold Value 14
PVP_THCn_TH15	THCn Threshold Value 15

Table 30-49: ADSP-BF60x PVP Register List (Continued)

Name	Description
PVP_THCn_HHPOS	THCn Histogram Horizontal Position
PVP_THCn_HVPOS	THCn Histogram Vertical Position
PVP_THCn_HHCNT	THCn Histogram Horizontal Count
PVP_THCn_HVCNT	THCn Histogram Vertical Count
PVP_THCn_RHPOS	THCn RLE Horizontal Position
PVP_THCn_RVPOS	THCn RLE Vertical Position
PVP_THCn_RHCNT	THCn RLE Horizontal Count
PVP_THCn_RVCNT	THCn RLE Vertical Count
PVP_THCn_HFCNT_STAT	THCn Histogram Frame Count Status
PVP_THCn_HCNT0_STAT	THCn Histogram Counter Value 0
PVP_THCn_HCNT1_STAT	THCn Histogram Counter Value 1
PVP_THCn_HCNT2_STAT	THCn Histogram Counter Value 2
PVP_THCn_HCNT3_STAT	THCn Histogram Counter Value 3
PVP_THCn_HCNT4_STAT	THCn Histogram Counter Value 4
PVP_THCn_HCNT5_STAT	THCn Histogram Counter Value 5
PVP_THCn_HCNT6_STAT	THCn Histogram Counter Value 6
PVP_THCn_HCNT7_STAT	THCn Histogram Counter Value 7
PVP_THCn_HCNT8_STAT	THCn Histogram Counter Value 8
PVP_THCn_HCNT9_STAT	THCn Histogram Counter Value 9
PVP_THCn_HCNT10_STAT	THCn Histogram Counter Value 10
PVP_THCn_HCNT11_STAT	THCn Histogram Counter Value 11
PVP_THCn_HCNT12_STAT	THCn Histogram Counter Value 12

Table 30-49: ADSP-BF60x PVP Register List (Continued)

Name	Description
PVP_THCn_HCNT13_STAT	THCn Histogram Counter Value 13
PVP_THCn_HCNT14_STAT	THCn Histogram Counter Value 14
PVP_THCn_HCNT15_STAT	THCn Histogram Counter Value 15
PVP_THCn_RREP_STAT	THCn Number of RLE Reports
PVP_PMA_CFG	PMA Configuration

Control

The PVP_CTL register enables the module, enables the memory pipe, enables the camera pipe, and selects the SCLK-to-PVPCLK clock divisor. Note that PVP_CTL.PVPEN, PVP_CTL.MPEN, and/or PVP_CTL.CPEN disable and re-enable MUST be followed by a fresh write of all required values in all shadow registers.

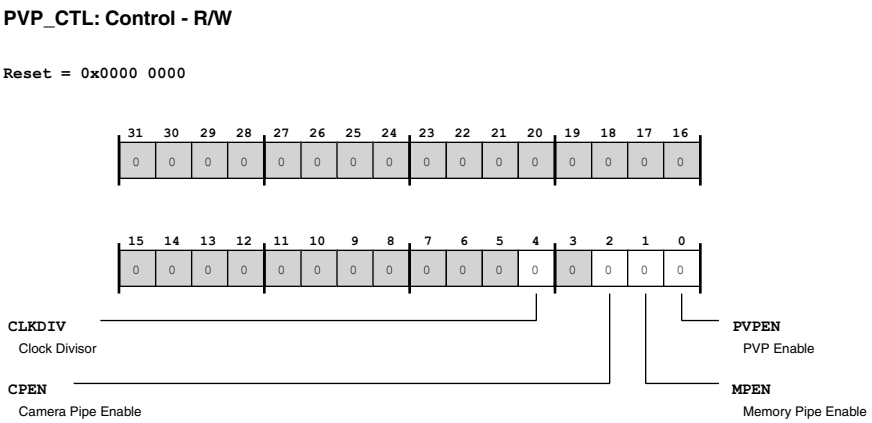


Figure 30-71: PVP_CTL Register Diagram

Table 30-50: PVP_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	CLKDIV	Clock Divisor. The PVP_CTL.CLKDIV selects the SCLK-to-PVPCLK clock divisor as: PVPCLK = SCLK / ($2^{PVP\_CTL.CLKDIV}$)	
		0	PVPCLK = SCLK
		1	PVPCLK = SCLK/2

Table 30-50: PVP_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	CPEN	Camera Pipe Enable. The PVP_CTL.CPEN enables or disables all processing blocks in the camera pipe.	
		0	Disable Camera Pipe
		1	Enable Camera Pipe
1 (R/W)	MPEN	Memory Pipe Enable. The PVP_CTL.MPEN enables/disables all processing blocks in the memory pipe.	
		0	Disable Memory Pipe
		1	Enable Memory Pipe
0 (R/W)	PVPEN	PVP Enable. The PVP_CTL.PVPEN enables/disables the PVP.	
		0	Disable PVP
		1	Enable PVP

Interrupt Mask n

The PVP_IMSKn register enables (unmasks) or disables (masks) status related interrupts for the PVP. For all PVP_IMSKn bits, setting the bit enables the interrupt, and clearing the bit disables the interrupt.

PVP_IMSKn: Interrupt Mask n - R/W

Reset = 0x0000 0000

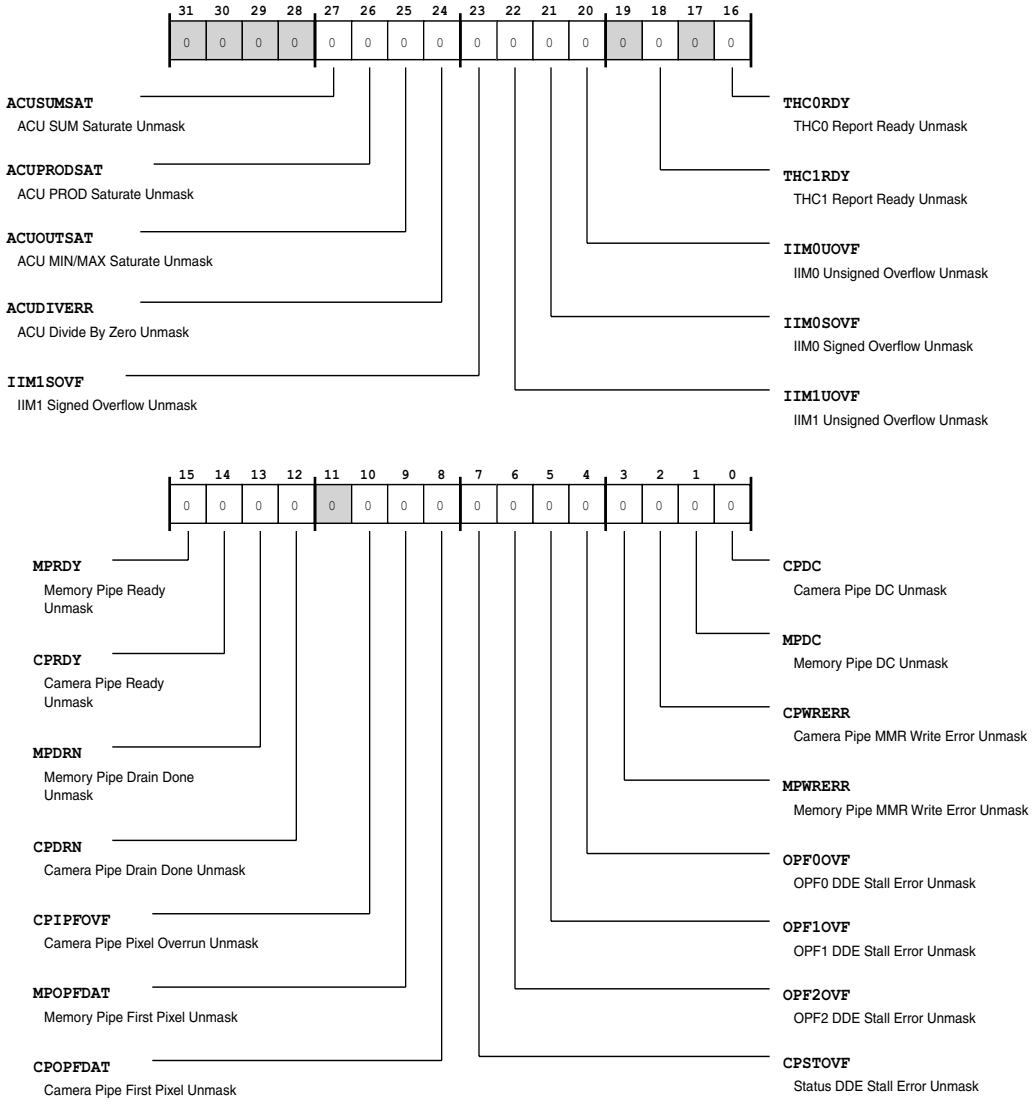


Figure 30-72: PVP_IMSKn Register Diagram

Table 30-51: PVP_IMSKn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27 (R/W)	ACUSUMSAT	ACU SUM Saturate Unmask. The PVP_IMSKn.ACUSUMSAT bit enables/disables the PVP_IMSKn.ACUSUMSAT interrupt.

Table 30-51: PVP_IMSKn Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
26 (R/W)	ACUPRODSAT	ACU PROD Saturate Unmask. The PVP_IMSKn.ACUPRODSAT bit enables/disables the PVP_IMSKn.ACUPRODSAT interrupt.
25 (R/W)	ACUOUTSAT	ACU MIN/MAX Saturate Unmask. The PVP_IMSKn.ACUOUTSAT bit enables/disables the PVP_IMSKn.ACUOUTSAT interrupt.
24 (R/W)	ACUDIVERR	ACU Divide By Zero Unmask. The PVP_IMSKn.ACUDIVERR bit enables/disables the PVP_IMSKn.ACUDIVERR interrupt.
23 (R/W)	IIM1SOVF	IIM1 Signed Overflow Unmask. The PVP_IMSKn.IIM1SOVF bit enables/disables the PVP_IMSKn.IIM1SOVF interrupt.
22 (R/W)	IIM1UOVF	IIM1 Unsigned Overflow Unmask. The PVP_IMSKn.IIM1UOVF bit enables/disables the PVP_IMSKn.IIM1UOVF interrupt.
21 (R/W)	IIM0SOVF	IIM0 Signed Overflow Unmask. The PVP_IMSKn.IIM0SOVF bit enables/disables the PVP_IMSKn.IIM0SOVF interrupt.
20 (R/W)	IIM0UOVF	IIM0 Unsigned Overflow Unmask. The PVP_IMSKn.IIM0UOVF bit enables/disables the PVP_IMSKn.IIM0UOVF interrupt.
18 (R/W)	THC1RDY	THC1 Report Ready Unmask. The PVP_IMSKn.THC1RDY bit enables/disables the PVP_IMSKn.THC1RDY interrupt.
16 (R/W)	THC0RDY	THC0 Report Ready Unmask. The PVP_IMSKn.THC0RDY bit enables/disables the PVP_IMSKn.THC0RDY interrupt.
15 (R/W)	MPRDY	Memory Pipe Ready Unmask. The PVP_IMSKn.MPRDY bit enables/disables the PVP_IMSKn.MPRDY interrupt.
14 (R/W)	CPRDY	Camera Pipe Ready Unmask. The PVP_IMSKn.CPRDY bit enables/disables the PVP_IMSKn.CPRDY interrupt.
13 (R/W)	MPDRN	Memory Pipe Drain Done Unmask. The PVP_IMSKn.MPDRN bit enables/disables the PVP_IMSKn.MPDRN interrupt.

Table 30-51: PVP_IMSKn Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	CPDRN	Camera Pipe Drain Done Unmask. The PVP_IMSKn.CPDRN bit enables/disables the PVP_IMSKn.CPDRN interrupt.
10 (R/W)	CPIPFOVF	Camera Pipe Pixel Overrun Unmask. The PVP_IMSKn.CPIPFOVF bit enables/disables the PVP_IMSKn.CPIPFOVF interrupt.
9 (R/W)	MPOPFDAT	Memory Pipe First Pixel Unmask. The PVP_IMSKn.MPOPFDAT bit enables/disables the PVP_IMSKn.MPOPFDAT interrupt.
8 (R/W)	CPOPFDAT	Camera Pipe First Pixel Unmask. The PVP_IMSKn.CPOPFDAT bit enables/disables the PVP_IMSKn.CPOPFDAT interrupt.
7 (R/W)	CPSTOVF	Status DDE Stall Error Unmask. The PVP_IMSKn.CPSTOVF bit enables/disables the PVP_IMSKn.CPSTOVF interrupt.
6 (R/W)	OPF2OVF	OPF2 DDE Stall Error Unmask. The PVP_IMSKn.OPF2OVF bit enables/disables the PVP_IMSKn.OPF2OVF interrupt.
5 (R/W)	OPF1OVF	OPF1 DDE Stall Error Unmask. The PVP_IMSKn.OPF1OVF bit enables/disables the PVP_IMSKn.OPF1OVF interrupt.
4 (R/W)	OPF0OVF	OPF0 DDE Stall Error Unmask. The PVP_IMSKn.OPF0OVF bit enables/disables the PVP_IMSKn.OPF0OVF interrupt.
3 (R/W)	MPWRERR	Memory Pipe MMR Write Error Unmask. The PVP_IMSKn.MPWRERR bit enables/disables the PVP_IMSKn.MPWRERR interrupt.
2 (R/W)	CPWRERR	Camera Pipe MMR Write Error Unmask. The PVP_IMSKn.CPWRERR bit enables/disables the PVP_IMSKn.CPWRERR interrupt.
1 (R/W)	MPDC	Memory Pipe DC Unmask. The PVP_IMSKn.MPDC bit enables/disables the PVP_IMSKn.MPDC interrupt.
0 (R/W)	CPDC	Camera Pipe DC Unmask. The PVP_IMSKn.CPDC bit enables/disables the PVP_IMSKn.CPDC interrupt.

Status

The PVP_STAT register indicates whether or not a PVP status related interrupts is pending. For all PVP_STAT bits, a set bit indicates a pending interrupt, and a cleared bit indicates that no interrupt is pending. The bits in this register are automatically set or cleared by changes to PVP status, and all bits are read-only.

PVP_STAT: Status - R/NW

Reset = 0x0000 c000

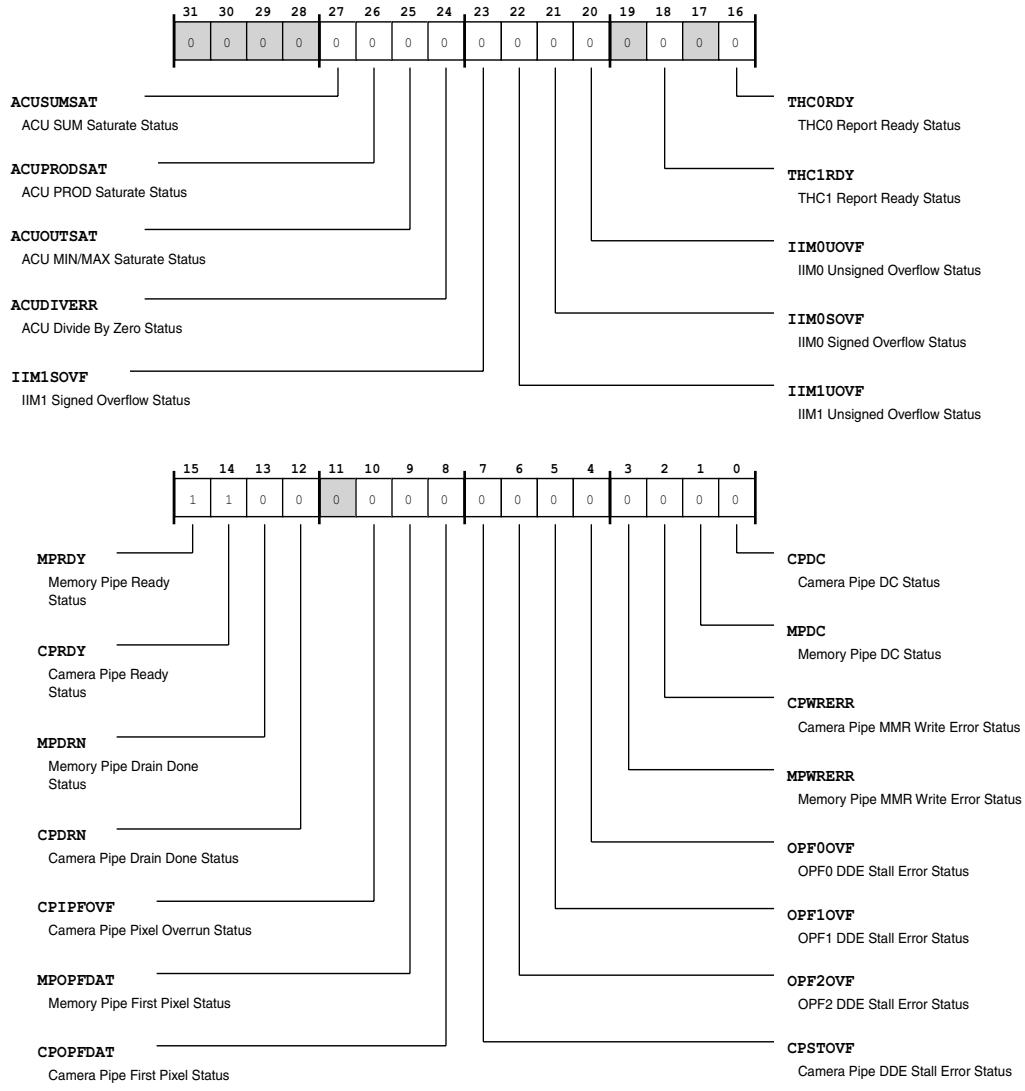


Figure 30-73: PVP_STAT Register Diagram

Table 30-52: PVP_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
27 (R/NW)	ACUSUMSAT	ACU SUM Saturate Status. The PVP_STAT.ACUSUMSAT bit indicates whether or not the PVP_STAT.ACUSUMSAT interrupt is pending. This interrupt is set by an ACU on a frame boundary having sum saturation, and this interrupt is cleared on the next frame boundary.	
		0	No Pending Interrupt
		1	Pending Interrupt
26 (R/NW)	ACUPRODSAT	ACU PROD Saturate Status. The PVP_STAT.ACUPRODSAT bit indicates whether or not the PVP_STAT.ACUPRODSAT interrupt is pending. This interrupt is set by an ACU on a frame boundary having product saturation, and this interrupt is cleared on the next frame boundary.	
		0	No Pending Interrupt
		1	Pending Interrupt
25 (R/NW)	ACUOUTSAT	ACU MIN/MAX Saturate Status. The PVP_STAT.ACUOUTSAT bit indicates whether or not the PVP_STAT.ACUOUTSAT interrupt is pending. This interrupt is set by an ACU on a frame boundary having min or max saturation, and this interrupt is cleared on the next frame boundary.	
		0	No Pending Interrupt
		1	Pending Interrupt
24 (R/NW)	ACUDIVERR	ACU Divide By Zero Status. The PVP_STAT.ACUDIVERR bit indicates whether or not the PVP_STAT.ACUDIVERR interrupt is pending. This interrupt is set by an ACU on a frame boundary having divide by zero error, and this interrupt is cleared on the next frame boundary.	
		0	No Pending Interrupt
		1	Pending Interrupt
23 (R/NW)	IIM1SOVF	IIM1 Signed Overflow Status. The PVP_STAT.IIM1SOVF bit indicates whether or not the PVP_STAT.IIM1SOVF interrupt is pending. This interrupt is set by an IIM every frame boundary when signed saturation occurs, and this interrupt is cleared on the next frame boundary.	
		0	No Pending Interrupt
		1	Pending Interrupt

Table 30-52: PVP_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/NW)	IIM1UOVF	IIM1 Unsigned Overflow Status. The PVP_STAT.IIM1UOVF bit indicates whether or not the PVP_STAT.IIM1UOVF interrupt is pending. This interrupt is set by an IIM every frame boundary when unsigned saturation occurs, and this interrupt is cleared on the next frame boundary.	
		0	No Pending Interrupt
		1	Pending Interrupt
21 (R/NW)	IIM0SOVF	IIM0 Signed Overflow Status. The PVP_STAT.IIM0SOVF bit indicates whether or not the PVP_STAT.IIM0SOVF interrupt is pending. This interrupt is set by an IIM every frame boundary when signed saturation occurs, and this interrupt is cleared on the next frame boundary.	
		0	No Pending Interrupt
		1	Pending Interrupt
20 (R/NW)	IIM0UOVF	IIM0 Unsigned Overflow Status. The PVP_STAT.IIM0UOVF bit indicates whether or not the PVP_STAT.IIM0UOVF interrupt is pending. This interrupt is set by an IIM every frame boundary when unsigned saturation occurs, and this interrupt is cleared on the next frame boundary.	
		0	No Pending Interrupt
		1	Pending Interrupt
18 (R/NW)	THC1RDY	THC1 Report Ready Status. The PVP_STAT.THC1RDY bit indicates whether or not the PVP_STAT.THC1RDY interrupt is pending. This interrupt is set by THC1 on every HFCNT expiry boundary or when auto-disabled, and this interrupt is cleared at the beginning of the next line of the next frame.	
		0	No Pending Interrupt
		1	Pending Interrupt
16 (R/NW)	THC0RDY	THC0 Report Ready Status. The PVP_STAT.THC0RDY bit indicates whether or not the PVP_STAT.THC0RDY interrupt is pending. This interrupt is set by THC0 on every HFCNT expiry boundary or when auto-disabled, and this interrupt is cleared at the beginning of the next line of the next frame.	
		0	No Pending Interrupt
		1	Pending Interrupt

Table 30-52: PVP_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/NW)	MPRDY	Memory Pipe Ready Status. The PVP_STAT.MPRDY bit indicates whether or not the PVP_STAT.MPRDY interrupt is pending. This interrupt is set when the IPF1 application start bit is cleared, and this interrupt is cleared when the IPF1 application start bit is set.	
		0	No Pending Interrupt
		1	Pending Interrupt
14 (R/NW)	CPRDY	Camera Pipe Ready Status. The PVP_STAT.CPRDY bit indicates whether or not the PVP_STAT.CPRDY interrupt is pending. This interrupt is set when the IPF0 application start bit is cleared, and this interrupt is cleared when the IPF0 application start bit is set.	
		0	No Pending Interrupt
		1	Pending Interrupt
13 (R/NW)	MPDRN	Memory Pipe Drain Done Status. The PVP_STAT.MPDRN bit indicates whether or not the PVP_STAT.MPDRN interrupt is pending. This interrupt is set when the memory pipes are drained and all memory pipe modules are disabled, and this interrupt is cleared when the next memory pipe daisy chain load is started.	
		0	No Pending Interrupt
		1	Pending Interrupt
12 (R/NW)	CPDRN	Camera Pipe Drain Done Status. The PVP_STAT.CPDRN bit indicates whether or not the PVP_STAT.CPDRN interrupt is pending. This interrupt is set when all three camera pipes are drained and all camera pipe modules are disabled, and this interrupt is cleared when the next camera pipe daisy chain load is started.	
		0	No Pending Interrupt
		1	Pending Interrupt

Table 30-52: PVP_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/NW)	CPIPFOVF	Camera Pipe Pixel Overrun Status. The PVP_STAT.CPIPFOVF bit indicates whether or not the PVP_STAT.CPIPFOVF interrupt is pending. This interrupt is set when pixel overrun occurs in IPF0, and this interrupt is cleared when the first pixel of the next line of that frame has entered memory pipe OPF.	
		0	No Pending Interrupt
		1	Pending Interrupt
9 (R/NW)	MPOPFDAT	Memory Pipe First Pixel Status. The PVP_STAT.MPOPFDAT bit indicates whether or not the PVP_STAT.MPOPFDAT interrupt is pending. This interrupt is set when the first pixel of the frame has entered memory pipe OPF, and this interrupt is cleared when the first pixel of the next line of that frame has entered memory pipe OPF.	
		0	No Pending Interrupt
		1	Pending Interrupt
8 (R/NW)	CPOPFDAT	Camera Pipe First Pixel Status. The PVP_STAT.CPOPFDAT bit indicates whether or not the PVP_STAT.CPOPFDAT interrupt is pending. This interrupt is set when the first pixel of the frame has entered all 3 camera pipe OPFs, and this interrupt is cleared when the first pixel of the next line of that frame has entered all 3 OPFs.	
		0	No Pending Interrupt
		1	Pending Interrupt
7 (R/NW)	CPSTOVF	Camera Pipe DDE Stall Error Status. The PVP_STAT.CPSTOVF bit indicates whether or not the PVP_STAT.CPSTOVF interrupt is pending. This interrupt is set when FIFO overrun occurs in status DDE, and this interrupt is cleared when first pixel of the next frame enters the FIFO.	
		0	No Pending Interrupt
		1	Pending Interrupt

Table 30-52: PVP_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/NW)	OPF2OVF	OPF2 DDE Stall Error Status. The PVP_STAT.OPF2OVF bit indicates whether or not the PVP_STAT.OPF2OVF interrupt is pending. This interrupt is set when FIFO overrun occurs in OPF2, and this interrupt is cleared when first pixel of the next frame enters the FIFO.	
		0	No Pending Interrupt
		1	Pending Interrupt
5 (R/NW)	OPF1OVF	OPF1 DDE Stall Error Status. The PVP_STAT.OPF1OVF bit indicates whether or not the PVP_STAT.OPF1OVF interrupt is pending. This interrupt is set when FIFO overrun occurs in OPF1, and this interrupt is cleared when first pixel of the next frame enters the FIFO.	
		0	No Pending Interrupt
		1	Pending Interrupt
4 (R/NW)	OPF0OVF	OPF0 DDE Stall Error Status. The PVP_STAT.OPF0OVF bit indicates whether or not the PVP_STAT.OPF0OVF interrupt is pending. This interrupt is set when FIFO overrun occurs in OPF0, and this interrupt is cleared when first pixel of the next frame enters the FIFO.	
		0	No Pending Interrupt
		1	Pending Interrupt
3 (R/NW)	MPWRERR	Memory Pipe MMR Write Error Status. The PVP_STAT.MPWRERR bit indicates whether or not the PVP_STAT.MPWRERR interrupt is pending. This interrupt is set when a MMR write was attempted to a memory pipe module during daisy chain load, and this interrupt is cleared when daisy chain load is done.	
		0	No Pending Interrupt
		1	Pending Interrupt
2 (R/NW)	CPWRERR	Camera Pipe MMR Write Error Status. The PVP_STAT.CPWRERR bit indicates whether or not the PVP_STAT.CPWRERR interrupt is pending. This interrupt is set when a MMR write was attempted to a camera pipe module during daisy chain load, and this interrupt is cleared when daisy chain load is done.	
		0	No Pending Interrupt
		1	Pending Interrupt

Table 30-52: PVP_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	MPDC	Memory Pipe DC Status. The PVP_STAT.MPDC bit indicates whether or not the PVP_STAT.MPDC interrupt is pending. This interrupt is set when memory pipe daisy chain load status equals done, and this interrupt is cleared by daisy chain load start.	
		0	No Pending Interrupt
		1	Pending Interrupt
0 (R/NW)	CPDC	Camera Pipe DC Status. The PVP_STAT.CPDC bit indicates whether or not the PVP_STAT.CPDC interrupt is pending. This interrupt is set when all camera pipes daisy chain load equal done, and this interrupt is cleared by daisy chain load start.	
		0	No Pending Interrupt
		1	Pending Interrupt

Interrupt Latch Status n

The PVP_ILAT register indicates that latched status of interrupts for the PVP. For all PVP_ILAT bits, a set the bit indicates that the interrupt has occurred, and cleared bit indicates that the interrupt has not occurred. The bits in PVP_ILAT are set by PVP status, but are cleared with a write-1-to-clear operation. Note that clearing a bit in PVP_ILAT clears the corresponding bit in the PVP_IREQn register.

PVP_ILAT: Interrupt Latch Status n - R/W

Reset = 0x0000 c000

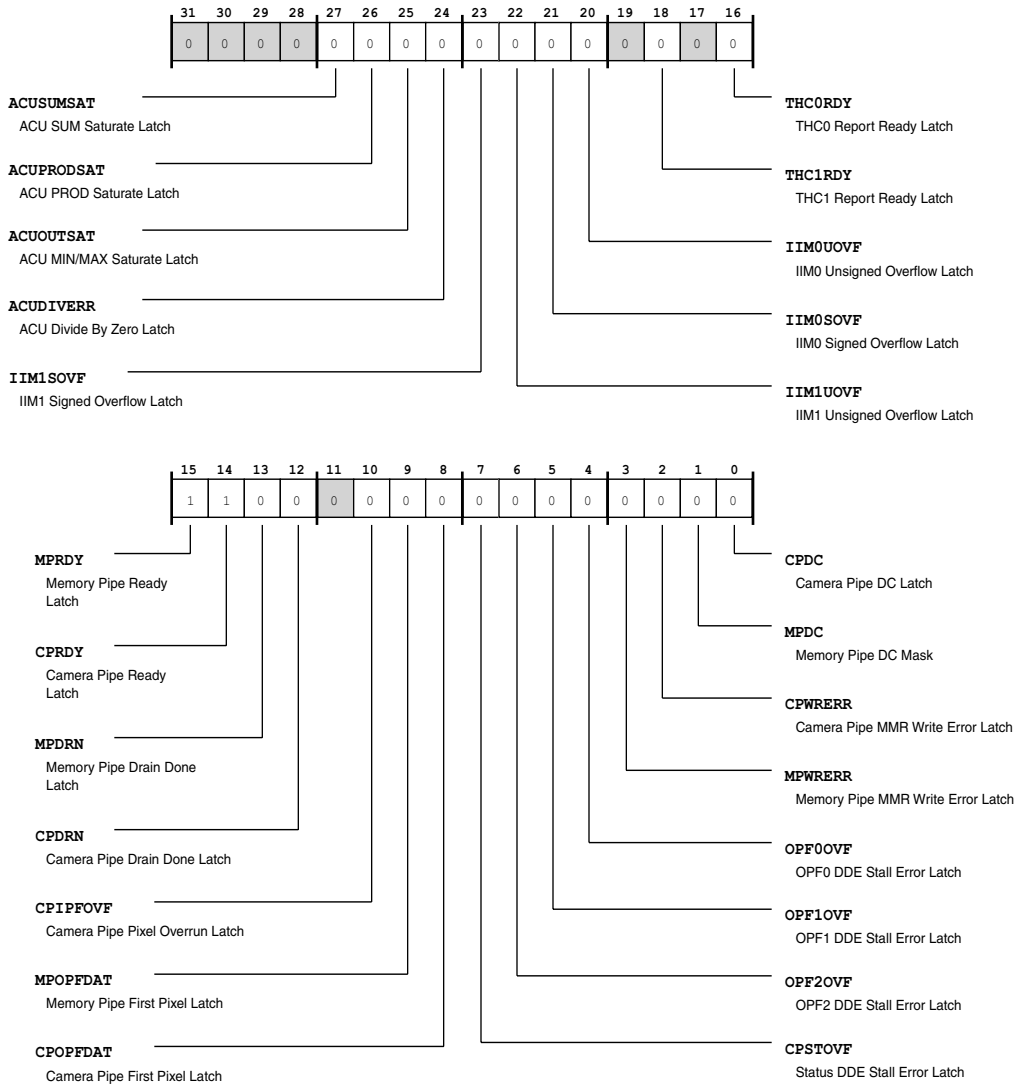


Figure 30-74: PVP_ILAT Register Diagram

Table 30-53: PVP_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
27 (R/W1C)	ACUSUMSAT	ACU SUM Saturate Latch. The PVP_ILAT.ACUSUMSAT bit indicates that the PVP_ILAT.ACUSUMSAT has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched

Table 30-53: PVP_ILAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/W1C)	ACUPRODSAT	ACU PROD Saturate Latch. The PVP_ILAT.ACUPRODSAT bit indicates that the PVP_ILAT.ACUPRODSAT has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
25 (R/W1C)	ACUOUTSAT	ACU MIN/MAX Saturate Latch. The PVP_ILAT.ACUOUTSAT bit indicates that the PVP_ILAT.ACUOUTSAT has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
24 (R/W1C)	ACUDIVERR	ACU Divide By Zero Latch. The PVP_ILAT.ACUDIVERR bit indicates that the PVP_ILAT.ACUDIVERR has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
23 (R/W1C)	IIM1SOVF	IIM1 Signed Overflow Latch. The PVP_ILAT.IIM1SOVF bit indicates that the PVP_ILAT.IIM1SOVF has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
22 (R/W1C)	IIM1UOVF	IIM1 Unsigned Overflow Latch. The PVP_ILAT.IIM1UOVF bit indicates that the PVP_ILAT.IIM1UOVF has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
21 (R/W1C)	IIM0SOVF	IIM0 Signed Overflow Latch. The PVP_ILAT.IIM0SOVF bit indicates that the PVP_ILAT.IIM0SOVF has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
20 (R/W1C)	IIM0UOVF	IIM0 Unsigned Overflow Latch. The PVP_ILAT.IIM0UOVF bit indicates that the PVP_ILAT.IIM0UOVF has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
18 (R/W1C)	THC1RDY	THC1 Report Ready Latch. The PVP_ILAT.THC1RDY bit indicates that the PVP_ILAT.THC1RDY has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched

Table 30-53: PVP_ILAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W1C)	THC0RDY	THC0 Report Ready Latch. The PVP_ILAT.THCORDY bit indicates that the PVP_ILAT.THCORDY has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
15 (R/W1C)	MPRDY	Memory Pipe Ready Latch. The PVP_ILAT.MPRDY bit indicates that the PVP_ILAT.MPRDY has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
14 (R/W1C)	CPRDY	Camera Pipe Ready Latch. The PVP_ILAT.CPRDY bit indicates that the PVP_ILAT.CPRDY has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
13 (R/W1C)	MPDRN	Memory Pipe Drain Done Latch. The PVP_ILAT.MPDRN bit indicates that the PVP_ILAT.MPDRN has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
12 (R/W1C)	CPDRN	Camera Pipe Drain Done Latch. The PVP_ILAT.CPDRN bit indicates that the PVP_ILAT.CPDRN has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
10 (R/W1C)	CPIPFOVF	Camera Pipe Pixel Overrun Latch. The PVP_ILAT.CPIPFOVF bit indicates that the PVP_ILAT.CPIPFOVF has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
9 (R/W1C)	MPOPFDAT	Memory Pipe First Pixel Latch. The PVP_ILAT.MPOPFDAT bit indicates that the PVP_ILAT.MPOPFDAT has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
8 (R/W1C)	CPOPFDAT	Camera Pipe First Pixel Latch. The PVP_ILAT.CPOPFDAT bit indicates that the PVP_ILAT.CPOPFDAT has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched

Table 30-53: PVP_ILAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	CPSTOVF	Status DDE Stall Error Latch. The PVP_ILAT.CPSTOVF bit indicates that the PVP_ILAT.CPSTOVF has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
6 (R/W1C)	OPF2OVF	OPF2 DDE Stall Error Latch. The PVP_ILAT.OPF2OVF bit indicates that the PVP_ILAT.OPF2OVF has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
5 (R/W1C)	OPF1OVF	OPF1 DDE Stall Error Latch. The PVP_ILAT.OPF1OVF bit indicates that the PVP_ILAT.OPF1OVF has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
4 (R/W1C)	OPF0OVF	OPF0 DDE Stall Error Latch. The PVP_ILAT.OPF0OVF bit indicates that the PVP_ILAT.OPF0OVF has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
3 (R/W1C)	MPWRERR	Memory Pipe MMR Write Error Latch. The PVP_ILAT.MPWRERR bit indicates that the PVP_ILAT.MPWRERR has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
2 (R/W1C)	CPWRERR	Camera Pipe MMR Write Error Latch. The PVP_ILAT.CPWRERR bit indicates that the PVP_ILAT.CPWRERR has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
1 (R/W1C)	MPDC	Memory Pipe DC Mask. The PVP_ILAT.MPDC bit indicates that the PVP_ILAT.MPDC has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched
0 (R/W1C)	CPDC	Camera Pipe DC Latch. The PVP_ILAT.CPDC bit indicates that the PVP_ILAT.CPDC has occurred. Write 1 to clear this status bit.	
		1	Interrupt Latched

Interrupt Request n

The PVP_IREQn register contains the latched interrupt request status. These requests reflect the status of interrupts that are latched in the PVP_ILAT register and are unmasked (enabled) in the PVP_IMSKn. The PVP_IREQn bits are set by PVP status, but cleared through a write-1-to-clear operation. Note that clearing a bit in PVP_IREQn clears the corresponding bit in the PVP_ILAT register.

PVP_IREQn: Interrupt Request n - R/W

Reset = 0x0000 0000

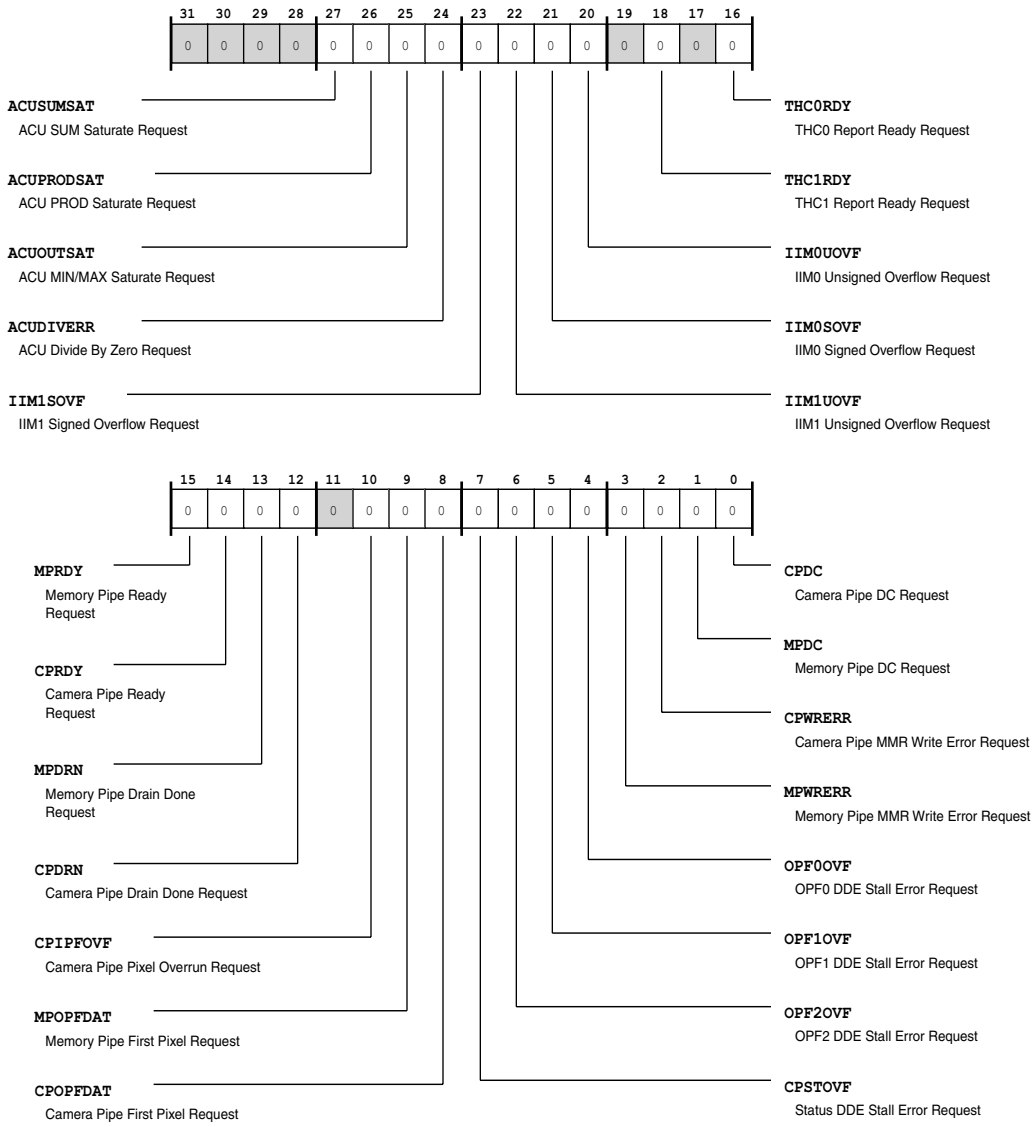


Figure 30-75: PVP_IREQn Register Diagram

Table 30-54: PVP_IREQn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
27 (R/W1C)	ACUSUMSAT	ACU SUM Saturate Request.	
		1	Unmasked Int Req Latched
26 (R/W1C)	ACUPRODSAT	ACU PROD Saturate Request.	
		1	Unmasked Int Req Latched
25 (R/W1C)	ACUOUTSAT	ACU MIN/MAX Saturate Request.	
		1	Unmasked Int Req Latched
24 (R/W1C)	ACUDIVERR	ACU Divide By Zero Request.	
		1	Unmasked Int Req Latched
23 (R/W1C)	IIM1SOVF	IIM1 Signed Overflow Request.	
		1	Unmasked Int Req Latched
22 (R/W1C)	IIM1UOVF	IIM1 Unsigned Overflow Request.	
		1	Unmasked Int Req Latched
21 (R/W1C)	IIM0SOVF	IIM0 Signed Overflow Request.	
		1	Unmasked Int Req Latched
20 (R/W1C)	IIM0UOVF	IIM0 Unsigned Overflow Request.	
		1	Unmasked Int Req Latched
18 (R/W1C)	THC1RDY	THC1 Report Ready Request.	
		1	Unmasked Int Req Latched
16 (R/W1C)	THC0RDY	THC0 Report Ready Request.	
		1	Unmasked Int Req Latched
15 (R/W1C)	MPRDY	Memory Pipe Ready Request.	
		1	Unmasked Int Req Latched
14 (R/W1C)	CPRDY	Camera Pipe Ready Request.	
		1	Unmasked Int Req Latched
13 (R/W1C)	MPDRN	Memory Pipe Drain Done Request.	
		1	Unmasked Int Req Latched
12 (R/W1C)	CPDRN	Camera Pipe Drain Done Request.	
		1	Unmasked Int Req Latched
10 (R/W1C)	CPIPFOVF	Camera Pipe Pixel Overrun Request.	
		1	Unmasked Int Req Latched

Table 30-54: PVP_IREQn Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W1C)	MPOPFDAT	Memory Pipe First Pixel Request.	
		1	Unmasked Int Req Latched
8 (R/W1C)	CPOPFDAT	Camera Pipe First Pixel Request.	
		1	Unmasked Int Req Latched
7 (R/W1C)	CPSTOVF	Status DDE Stall Error Request.	
		1	Unmasked Int Req Latched
6 (R/W1C)	OPF2OVF	OPF2 DDE Stall Error Request.	
		1	Unmasked Int Req Latched
5 (R/W1C)	OPF1OVF	OPF1 DDE Stall Error Request.	
		1	Unmasked Int Req Latched
4 (R/W1C)	OPF0OVF	OPF0 DDE Stall Error Request.	
		1	Unmasked Int Req Latched
3 (R/W1C)	MPWRERR	Memory Pipe MMR Write Error Request.	
		1	Unmasked Int Req Latched
2 (R/W1C)	CPWRERR	Camera Pipe MMR Write Error Request.	
		1	Unmasked Int Req Latched
1 (R/W1C)	MPDC	Memory Pipe DC Request.	
		1	Unmasked Int Req Latched
0 (R/W1C)	CPDC	Camera Pipe DC Request.	
		1	Unmasked Int Req Latched

OPFn (Camera Pipe) Configuration

The output data formatter (OPF) blocks house FIFOs that buffer data for a sustained peak bandwidth output. The `PVP_OPFn_CFG` register enables the block's FIFO and configures blocks' connections within thePVP.

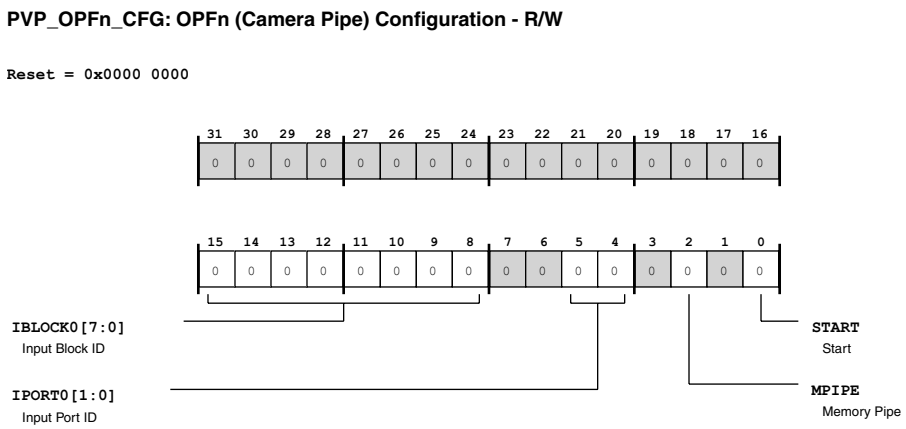


Figure 30-76: PVP_OPFn_CFG Register Diagram

Table 30-55: PVP_OPFn_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:8 (R/W)	IBLOCK0	Input Block ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_OPFn_CFG.IBLOCK0 and PVP_OPFn_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s).	
		5	PEC
		6	IIM0
		7	IIM1
		8	ACU
		12	IPF0
		16	CNV0
		20	CNV1
		24	CNV2
		28	CNV3
		32	THC0
		40	THC1
		48	PMA

Table 30-55: PVP_OPFn_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W)	IPORT0	Input Port ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_OPFn_CFG.IBLOCK0 and PVP_OPFn_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s). For more information, see the connectivity tables for each input block in the architectural concepts section.	
2 (R/NW)	MPIPE	Memory Pipe. The PVP_OPFn_CFG.MPIPE bit always reads as 0 (camera pipe).	
0 (R/W)	START	Start. The PVP_OPFn_CFG.START enables the PVP. This bit reads 0 after the pipe is drained.	
		1	Enable (W1A)

OPFn (Camera Pipe) Control

The PVP_OPFn_CTL register controls the output data formatter (OPF) block's pipeline features, including input and output data size.

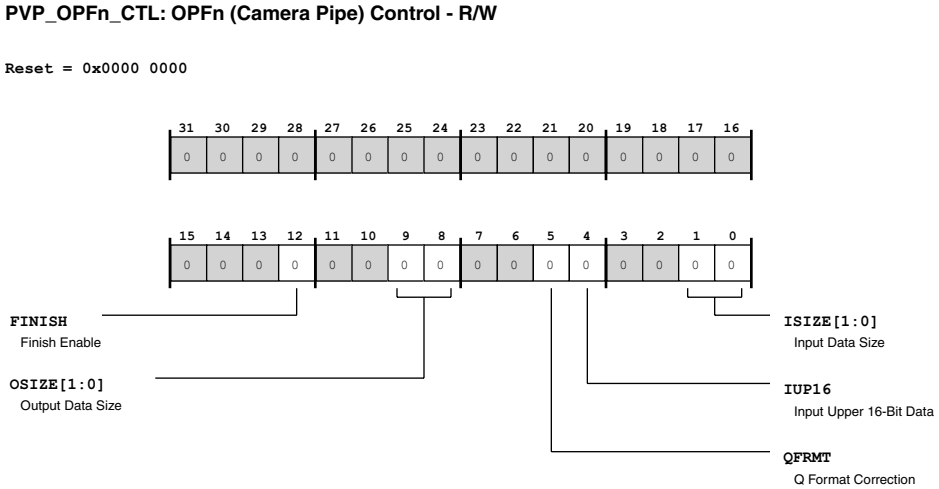


Figure 30-77: PVP_OPFn_CTL Register Diagram

Table 30-56: PVP_OPFn_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	FINISH	Finish Enable. The PVP_OPFn_CTL.FINISH enables the PVP to signal finished on frame boundary.	
		0	Disable Finish Signal
		1	Enable Finish Signal
9:8 (R/W)	OSIZE	Output Data Size. The PVP_OPFn_CTL.OSIZE selects the output data size. Note that the PVP_OPFn_CTL.OSIZE selection should be equal to or more than that of PVP_OPFn_CTL.ISIZE.	
		0	32-Bit Output Data Size
		1	16-Bit Output Data Size
		2	8-Bit Output Data Size
5 (R/W)	QFRMT	Q Format Correction. The PVP_OPFn_CTL.QFRMT enables Q format correction, converting data to unsigned by 1 left shift.	
		0	Disable Q Format Correction
		1	Enable Q Format Correction
4 (R/W)	IUP16	Input Upper 16-Bit Data. The PVP_OPFn_CTL.IUP16 selects whether the input data word is the upper or lower 16 bits (for input data sizes of less than 32 bits). Note that for PVP_OPFn_CTL.ISIZE of 8 bit or 4 bit, the input data is in the least significant bits of the lower 16 bits.	
		0	Lower 16 Bits
		1	Upper 16 Bits
1:0 (R/W)	ISIZE	Input Data Size. The PVP_OPFn_CTL.ISIZE selects the input data size.	
		0	32-Bit Input Data Size
		1	16-Bit Input Data Size
		2	8-Bit Input Data Size
		3	4-Bit Input Data Size

OPF3 (Memory Pipe) Configuration

The output data formatter (OPF) blocks house FIFOs that buffer data for a sustained peak bandwidth output. The PVP_OPF3_CFG register enables the block's FIFO and configures blocks' connections within thePVP.

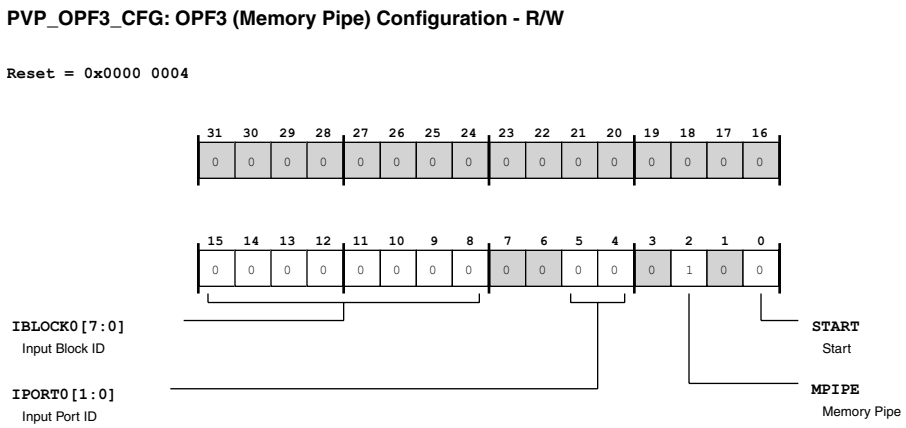


Figure 30-78: PVP_OPF3_CFG Register Diagram

Table 30-57: PVP_OPF3_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:8 (R/W)	IBLOCK0	Input Block ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_OPF3_CFG.IBLOCK0 and PVP_OPF3_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s).	
		5	PEC
		6	IIM0
		7	IIM1
		8	ACU
		10	UDS
		14	IPF1
		16	CNV0
		20	CNV1
		24	CNV2
		28	CNV3
		32	THC0
		40	THC1
		48	PMA
5:4 (R/W)	IPORT0	Input Port ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_OPF3_CFG.IBLOCK0 and PVP_OPF3_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s). For more information, see the connectivity tables for each input block in the architectural concepts section.	
2 (R/NW)	MPIPE	Memory Pipe. The PVP_OPF3_CFG.MPIPE bit always reads as 1 (memory pipe).	
0 (R/W)	START	Start. The PVP_OPF3_CFG.START enables the PVP. This bit reads 0 after the pipe is drained.	
		1	Enable (W1A)

OPF3 (Memory Pipe) Control

The PVP_OPF3_CTL register controls the OPF3 block's pipeline features, including input and output data size.

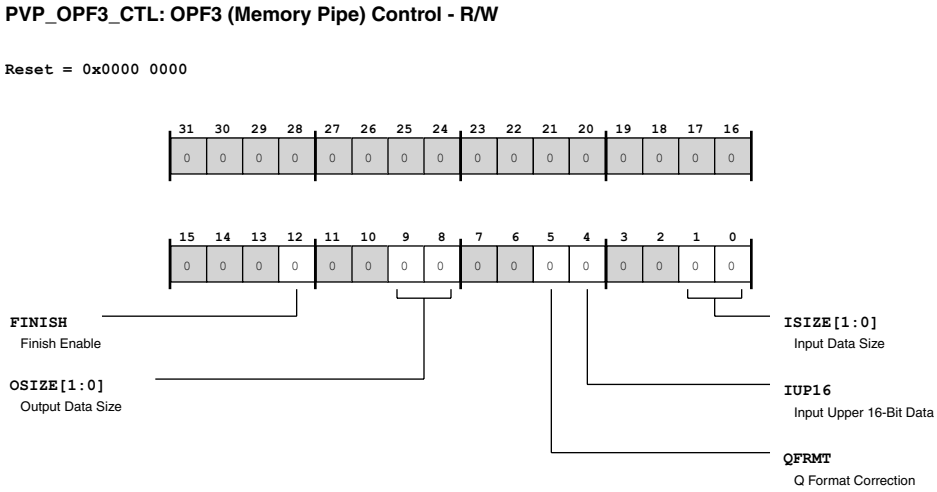


Figure 30-79: PVP_OPF3_CTL Register Diagram

Table 30-58: PVP_OPF3_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	FINISH	Finish Enable. The PVP_OPF3_CTL.FINISH enables the PVP to signal finished on frame boundary.	
		0	Disable Finish Signal
		1	Enable Finish Signal
9:8 (R/W)	OSIZE	Output Data Size. The PVP_OPF3_CTL.OSIZE selects the output data size. Note that the PVP_OPF3_CTL.OSIZE selection should be equal to or more than that of PVP_OPF3_CTL.ISIZE.	
		0	32-Bit Output Data Size
		1	16-Bit Output Data Size
		2	8-Bit Output Data Size
5 (R/W)	QFRMT	Q Format Correction. The PVP_OPF3_CTL.QFRMT enables Q format correction, converting data to unsigned by 1 left shift.	
		0	Disable Q Format Correction
		1	Enable Q Format Correction

Table 30-58: PVP_OPF3_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	IUP16	Input Upper 16-Bit Data. The PVP_OPF3_CTL.IUP16 selects whether the input data word is the upper or lower 16 bits (for input data sizes of less than 32 bits). Note that for PVP_OPF3_CTL.ISIZE of 8 bit or 4 bit, the input data is in the least significant bits of the lower 16 bits.	
		0	Lower 16 Bits
		1	Upper 16 Bits
1:0 (R/W)	ISIZE	Input Data Size. The PVP_OPF3_CTL.ISIZE selects the input data size.	
		0	32-Bit Input Data Size
		1	16-Bit Input Data Size
		2	8-Bit Input Data Size
		3	4-Bit Input Data Size

PEC Configuration

The pixel edge classifier (PEC) block performs pre-processing for the first derivative canny edge detection algorithm. The PVP_PEC_CFG register enables the block's FIFO and configures blocks' connections within thePVP.

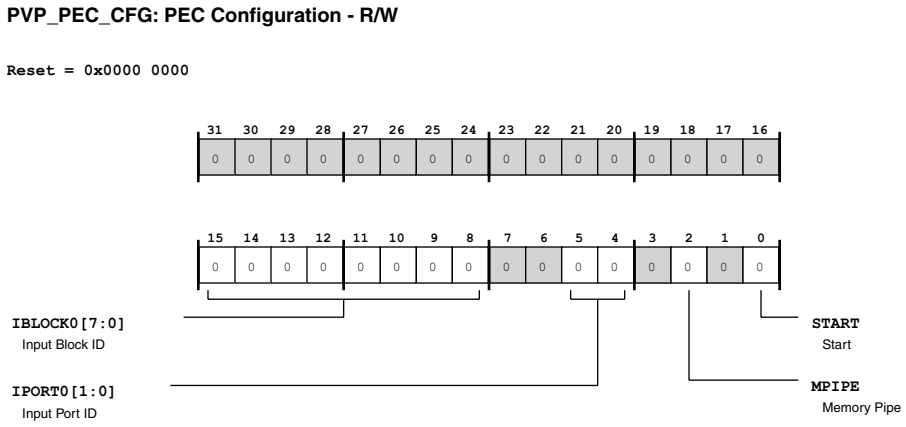


Figure 30-80: PVP_PEC_CFG Register Diagram

Table 30-59: PVP_PEC_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:8 (R/W)	IBLOCK0	Input Block ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_PEC_CFG.IBLOCK0 and PVP_PEC_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s).	
		8	ACU
		12	IPF0
		14	IPF1
		16	CNV0
		20	CNV1
		24	CNV2
		28	CNV3
		48	PMA
5:4 (R/W)	IPORT0	Input Port ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_PEC_CFG.IBLOCK0 and PVP_PEC_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s). For more information, see the connectivity tables for each input block in the architectural concepts section.	
2 (R/W)	MPIPE	Memory Pipe. The PVP_PEC_CFG.MPIPE bit selects memory pipe or camera pipe.	
		0	Camera Pipe
		1	Memory Pipe
0 (R/W)	START	Start. The PVP_PEC_CFG.START enables the PVP. This bit reads 0 after the pipe is drained.	
		1	Enable (W1A)

PEC Control

The PVP_PEC_CTL register selects operation modes for the pixel edge classifier (PEC) block, including 1st derivative mode (PEC1) and 2nd derivative mode (PEC2).

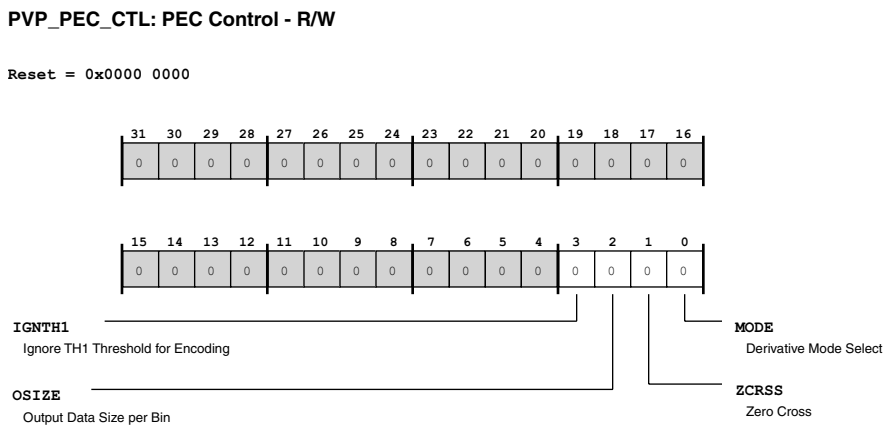


Figure 30-81: PVP_PEC_CTL Register Diagram

Table 30-60: PVP_PEC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	IGNTH1	Ignore TH1 Threshold for Encoding. The PVP_PEC_CTL.IGNTH1 selects PEC encoding control for strong and weak edges. This bit is valid in both 1st derivative and 2nd derivative modes of PEC.	
		0	Different Strong/Weak Edge Encoding
		1	Identical Strong/Weak Edge Encoding
2 (R/W)	OSIZE	Output Data Size per Bin. The PVP_PEC_CTL.OSIZE selects the output data size per bin. This bit is valid in both 1st derivative and 2nd derivative modes of PEC.	
		0	8 Bits Per Bin PEC Output Data Size
		1	16 Bits Per Bin PEC Output Data Size
1 (R/W)	ZCRSS	Zero Cross. The PVP_PEC_CTL.ZCRSS selects the sub-pixel value output to include sub-pixel values and either zero crossing codes or classified angle indices.	
		0	Angle Indices and Sub-Pixel Values
		1	Zero Crossing Codes and Sub-Pixel Values

Table 30-60: PVP_PEC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	MODE	Derivative Mode Select. The PVP_PEC_CTL.MODE selects that derivative mode for the PEC. Note that the PVP_PEC_CTL.MODE bits only can be changed when the PVP_IPFn_PIPECTL.DRAIN bit is set. If set, the PVP works on 16-bit, signed, 2nd derivative inputs. If cleared, the PVP works on 16-bit, unsigned, gradient magnitude and 5-bit angle bin.	
		0	1st Derivative Mode
		1	2nd Derivative Mode

PEC Lower Hysteresis Threshold

The PVP_PEC_D1TH0 provides the lower threshold value used by the Canny algorithm's hysteresis thresholding for streaking elimination.

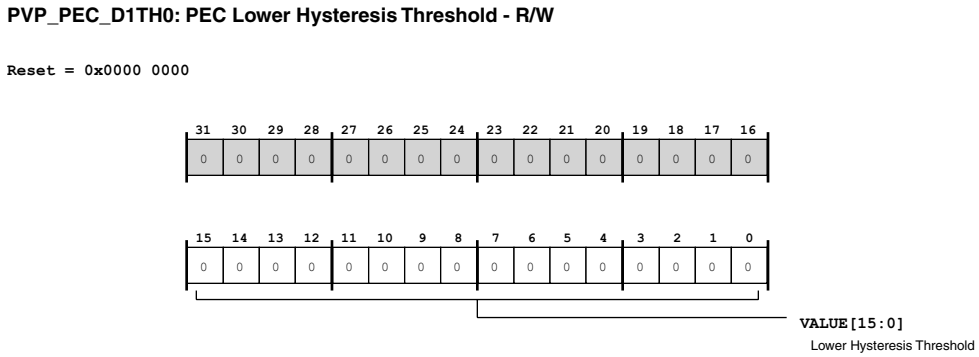


Figure 30-82: PVP_PEC_D1TH0 Register Diagram

Table 30-61: PVP_PEC_D1TH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Lower Hysteresis Threshold.

PEC Upper Hysteresis Threshold

The PVP_PEC_D1TH1 provides the upper threshold value used by the Canny algorithm's hysteresis thresholding for streaking elimination.

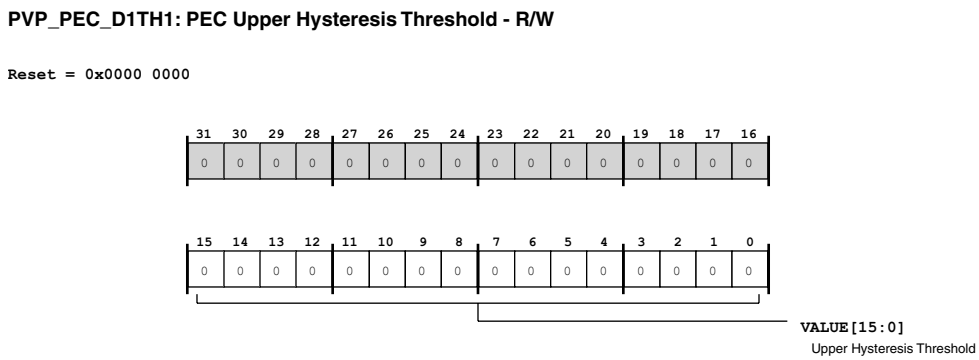


Figure 30-83: PVP_PEC_D1TH1 Register Diagram

Table 30-62: PVP_PEC_D1TH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Upper Hysteresis Threshold.

PEC Weak Zero Crossing Threshold

The PVP_PEC_D2TH0 provides an upper threshold that the PEC uses to remove the noise during weak zero crossing detection. Zero crossings are considered valid only if there is a sign change and id the difference between the 2nd derivative values of these two pixels is greater than the threshold.

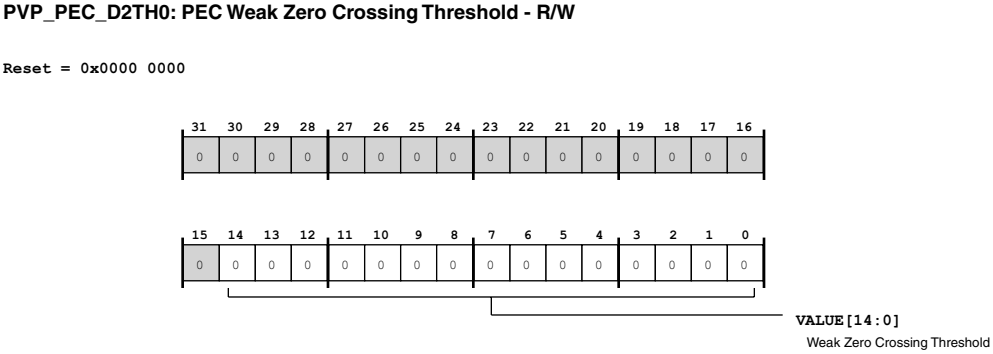


Figure 30-84: PVP_PEC_D2TH0 Register Diagram

Table 30-63: PVP_PEC_D2TH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/W)	VALUE	Weak Zero Crossing Threshold.

PEC Strong Zero Crossing Threshold

The PVP_PEC_D2TH1 provides an upper threshold that the PEC uses to remove the noise during strong zero crossing detection. Zero crossings are considered valid only if there is a sign change and id the difference between the 2nd derivative values of these two pixels is greater than the threshold.

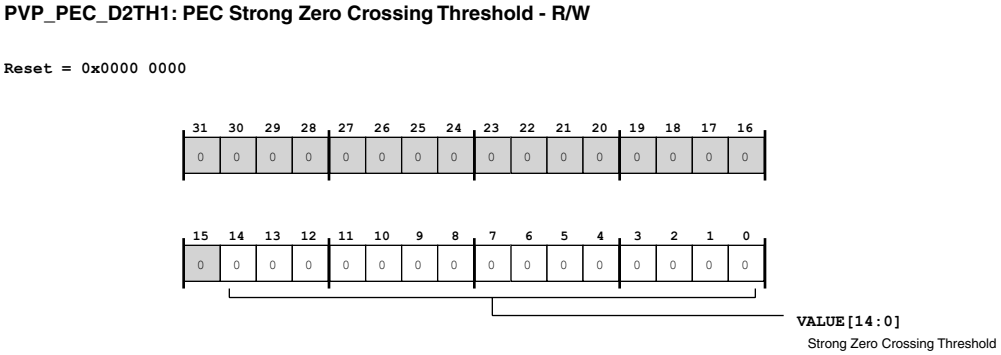


Figure 30-85: PVP_PEC_D2TH1 Register Diagram

Table 30-64: PVP_PEC_D2TH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/W)	VALUE	Strong Zero Crossing Threshold.

IIMn Configuration

The integral image computation (IIM) block computes the rectangular integral image algorithm. The PVP_IIMn_CFG register enables the block's FIFO and configures blocks' connections within thePVP.

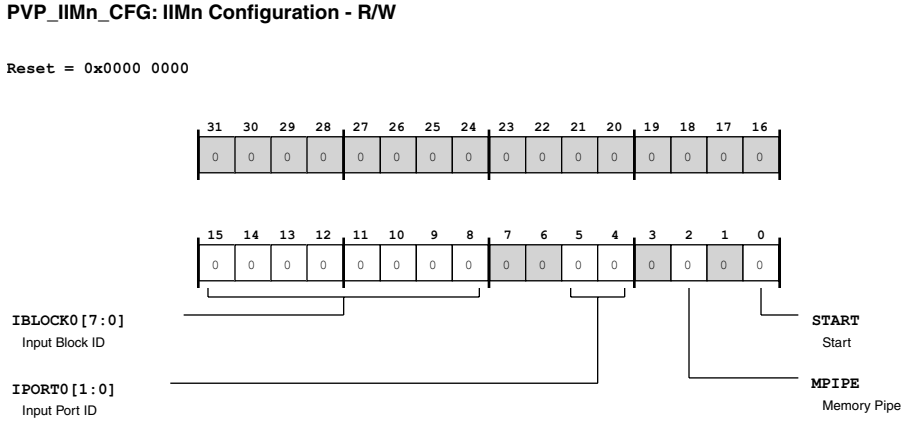


Figure 30-86: PVP_IIMn_CFG Register Diagram

Table 30-65: PVP_IIMn_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:8 (R/W)	IBLOCK0	Input Block ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_IIMn_CFG.IBLOCK0 and PVP_IIMn_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s).	
		5	PEC
		8	ACU
		12	IPF0
		14	IPF1
		16	CNV0
		20	CNV1
		24	CNV2
		28	CNV3
		32	THC0
		40	THC1
		48	PMA
5:4 (R/W)	IPORT0	Input Port ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_IIMn_CFG.IBLOCK0 and PVP_IIMn_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s). For more information, see the connectivity tables for each input block in the architectural concepts section.	
2 (R/W)	MPIPE	Memory Pipe. The PVP_IIMn_CFG.MPIPE bit selects memory pipe or camera pipe.	
		0	Camera Pipe
		1	Memory Pipe
0 (R/W)	START	Start. The PVP_IIMn_CFG.START enables the PVP. This bit reads 0 after the pipe is drained.	
		1	Enable (W1A)

IIMn Control

The PVP_IIMn_CTL register selects operation modes for the integral image computation (IIM) block, including rectangular mode, diagonal mode, and row mode.

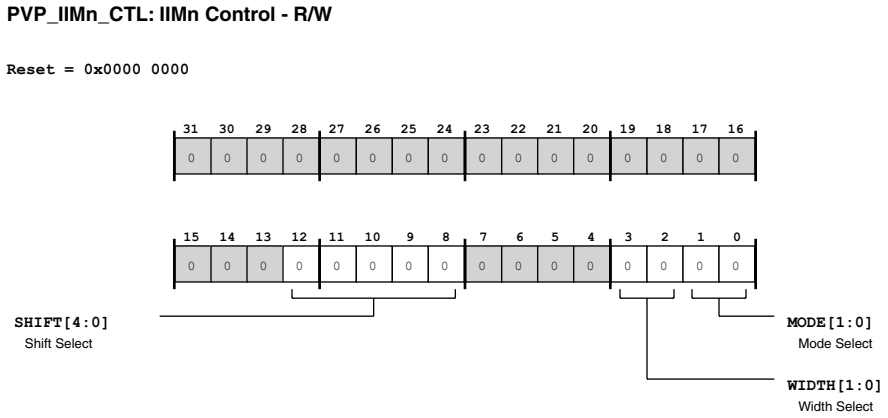


Figure 30-87: PVP_IIMn_CTL Register Diagram

Table 30-66: PVP_IIMn_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
12:8 (R/W)	SHIFT	Shift Select. The PVP_IIMn_CTL.SHIFT selects the amount of arithmetic right shift on input from 0 to 31 bits.	
3:2 (R/W)	WIDTH	Width Select. The PVP_IIMn_CTL.WIDTH selects the input data width for rectangular mode (SAT). This field is applicable only when PVP_IIMn_CTL.MODE =00 and is invalid for other IIM modes.	
		0	Single 32 Bit
		1	Dual 16 Bit
		2	Reserved
		3	Quad 8 Bit
1:0 (R/W)	MODE	Mode Select. The PVP_IIMn_CTL.MODE selects the IIM mode.	
		0	Rectangular Mode (SAT)
		1	Diagonal Mode (RSAT -45)
		2	Row Mode
		3	Reserved

IIMn Scaling Values

The PVP_IIMn_SCALE holds the independently controlled scaling factors in each dimension for the output's programmable downscaler.

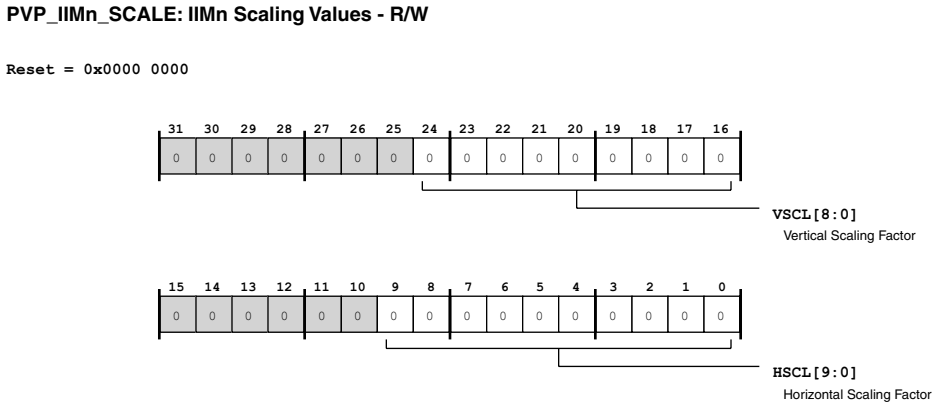


Figure 30-88: PVP_IIMn_SCALE Register Diagram

Table 30-67: PVP_IIMn_SCALE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:16 (R/W)	VSCL	Vertical Scaling Factor. The PVP_IIMn_SCALE.VSCL holds the vertical scaling factor, which can be selected to be any power of 2 from 1 to 512.
9:0 (R/W)	HSCL	Horizontal Scaling Factor. The PVP_IIMn_SCALE.HSCL holds the horizontal scaling factor, which can be selected to be any power of 2 from 1 to 1024.

IIMn Signed Overflow Status

The PVP_IIMn_SOVF_STAT holds the signed overflow status, which is determined considering addition of 32-bit signed numbers.

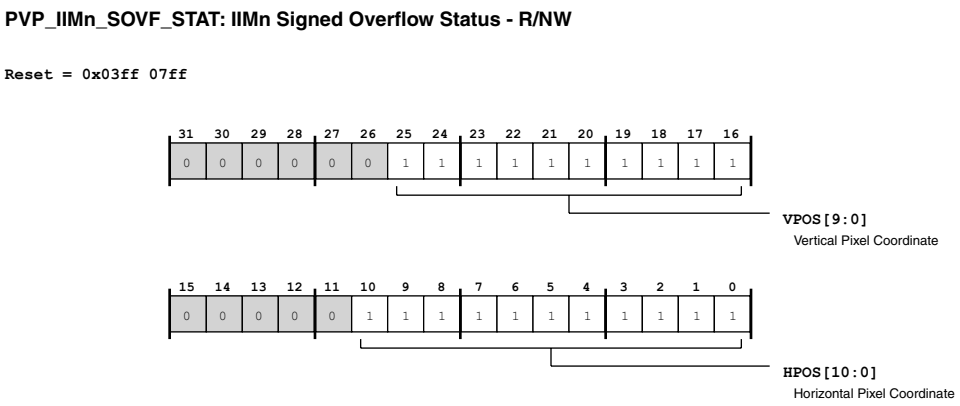


Figure 30-89: PVP_IIMn_SOVF_STAT Register Diagram

Table 30-68: PVP_IIMn_SOVF_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:16 (R/NW)	VPOS	Vertical Pixel Coordinate. The PVP_IIMn_SOVF_STAT.VPOS holds the vertical coordinate of the pixel which resulted in signed saturation.
10:0 (R/NW)	HPOS	Horizontal Pixel Coordinate. The PVP_IIMn_SOVF_STAT.HPOS holds the horizontal coordinate of the pixel which resulted in signed saturation.

IIMn Unsigned Overflow Status

The PVP_IIMn_UOVF_STAT holds the unsigned overflow status, which is determined considering addition of 32-bit unsigned numbers.

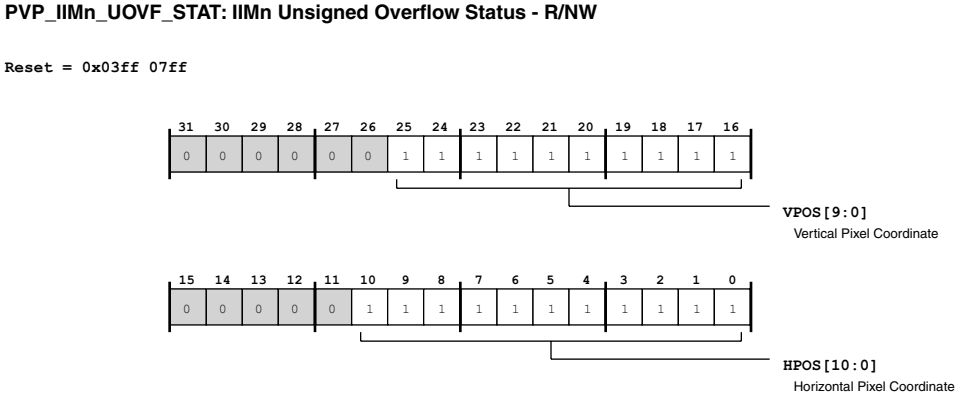


Figure 30-90: PVP_IIMn_UOVF_STAT Register Diagram

Table 30-69: PVP_IIMn_UOVF_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:16 (R/NW)	VPOS	Vertical Pixel Coordinate. The PVP_IIMn_UOVF_STAT.VPOS holds the vertical coordinate of the pixel which resulted in unsigned saturation.
10:0 (R/NW)	HPOS	Horizontal Pixel Coordinate. The PVP_IIMn_UOVF_STAT.HPOS holds the horizontal coordinate of the pixel which resulted in unsigned saturation.

ACU Configuration

The arithmetic computation unit (ACU) block provides basic 32-bit math such as addition, subtraction, multiplication, and division. This block also features a 48-bit accumulator and can normalize and saturate on the output. The PVP_ACU_CFG register enables the block's FIFO and configures blocks' connections within thePVP.

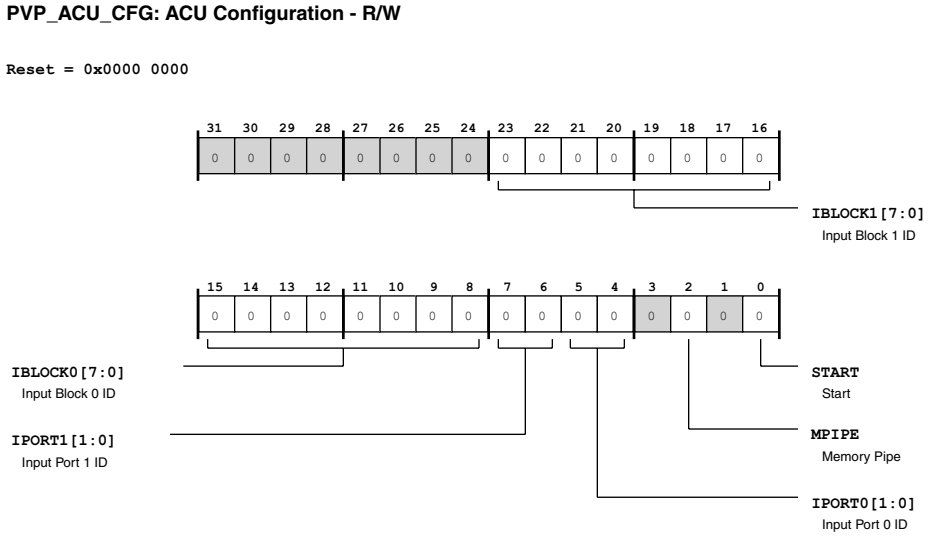


Figure 30-91: PVP_ACU_CFG Register Diagram

Table 30-70: PVP_ACU_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:16 (R/W)	IBLOCK1	Input Block 1 ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_ACU_CFG.IBLOCK1 and PVP_ACU_CFG.IPORT1 fields determine which output port(s) are connected to the input port(s).	
		12	IPF0
		14	IPF1
		16	CNV0
		20	CNV1
		24	CNV2
		28	CNV3
15:8 (R/W)	IBLOCK0	Input Block 0 ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_ACU_CFG.IBLOCK0 and PVP_ACU_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s).	
		12	IPF0
		14	IPF1
		16	CNV0
		20	CNV1
		24	CNV2
		28	CNV3
7:6 (R/W)	IPORT1	Input Port 1 ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_ACU_CFG.IBLOCK1 and PVP_ACU_CFG.IPORT1 fields determine which output port(s) are connected to the input port(s). For more information, see the connectivity tables for each input block in the architectural concepts section.	
5:4 (R/W)	IPORT0	Input Port 0 ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_ACU_CFG.IBLOCK0 and PVP_ACU_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s). For more information, see the connectivity tables for each input block in the architectural concepts section.	

Table 30-70: PVP_ACU_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	MPIPE	Memory Pipe. The PVP_ACU_CFG.MPIPE bit selects memory pipe or camera pipe.	
		0	Camera Pipe
		1	Memory Pipe
0 (R/W)	START	Start. The PVP_ACU_CFG.START enables the PVP. This bit reads 0 after the pipe is drained.	
		1	Write to Enable the Module Enable (W1A)

ACU Control

The PVP_ACU_CTL register selects operation modes for the arithmetic computation unit (ACU), including modes for sum, product, accumulate, and shift operations.

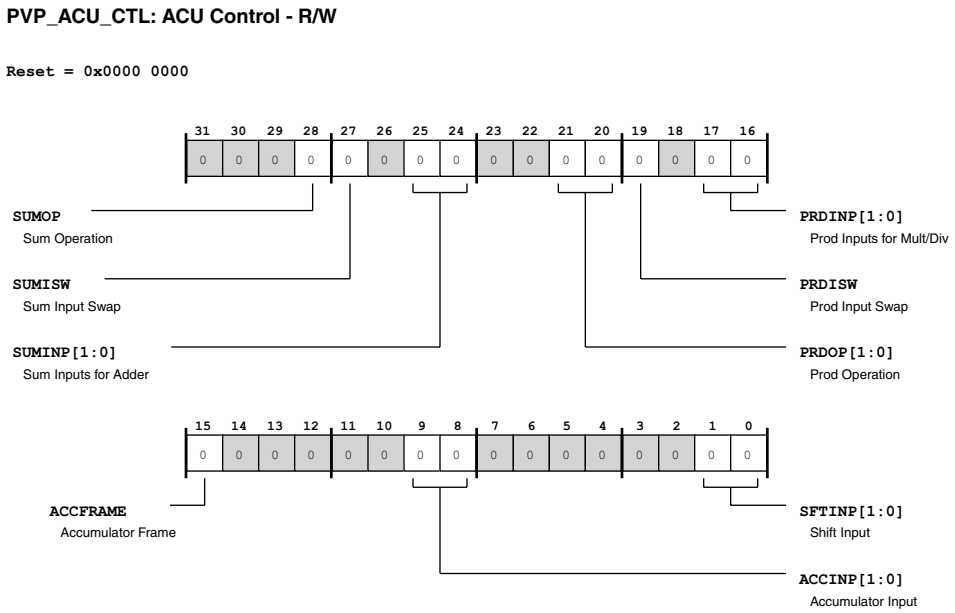


Figure 30-92: PVP_ACU_CTL Register Diagram

Table 30-71: PVP_ACU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
28 (R/W)	SUMOP	Sum Operation. The PVP_ACU_CTL.SUMOP selects add or subtract as the SUM function.	
		0	Add
		1	Subtract
27 (R/W)	SUMISW	Sum Input Swap. The PVP_ACU_CTL.SUMISW selects swapping of operands for the SUM function. The swap is meaningful for subtract operations.	
		0	Do Not Swap Operands
		1	Swap Operands
25:24 (R/W)	SUMINP	Sum Inputs for Adder. The PVP_ACU_CTL.SUMINP selects the inputs to the adder for the SUM function.	
		0	X,Y Inputs
		1	X,OFFSET Inputs
		2	Y,OFFSET Inputs
		3	Reserved
21:20 (R/W)	PRDOP	Prod Operation. The PVP_ACU_CTL.PRDOP selects multiply or divide as the PROD operation.	
		0	Multiply
		1	Divide with Quotient
		2	Divide with Modulus
		3	Reserved
19 (R/W)	PRDISW	Prod Input Swap. The PVP_ACU_CTL.PRDISW selects swapping of operands for the PROD function. The swap is meaningful for division operations.	
		0	Do Not Swap Operands
		1	Swap Operands

Table 30-71: PVP_ACU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17:16 (R/W)	PRDINP	Prod Inputs for Mult/Div. The PVP_ACU_CTL.PRDINP selects the inputs to the multiplier/ divider for the PROD function.	
		0	X,Y Inputs
		1	X,FACTOR Inputs
		2	Y,FACTOR Inputs
		3	SUM,FACTOR Inputs
15 (R/W)	ACCFRAME	Accumulator Frame. The PVP_ACU_CTL.ACCFRAME selects whether or not the accumulator is output and cleared after each row or frame.	
		0	Clear ACC After Row
		1	Clear ACC After Frame
9:8 (R/W)	ACCINP	Accumulator Input. The PVP_ACU_CTL.ACCINP selects the input to the accumulator for accumulate operations.	
		0	X Input
		1	SUM Input
		2	PROD Input
		3	Reserved
1:0 (R/W)	SFTINP	Shift Input. The PVP_ACU_CTL.SFTINP selects the input to the shifter for shift operations.	
		0	X Input
		1	SUM Result Input
		2	PROD Result Input
		3	ACC Result Input

ACU SUM Constant

The PVP_ACU_OFFSET provides the constant value (offset) input for the ACU SUM operation when the PVP_ACU_CTL.SUMINP field selects the offset as one of the inputs for the operation.

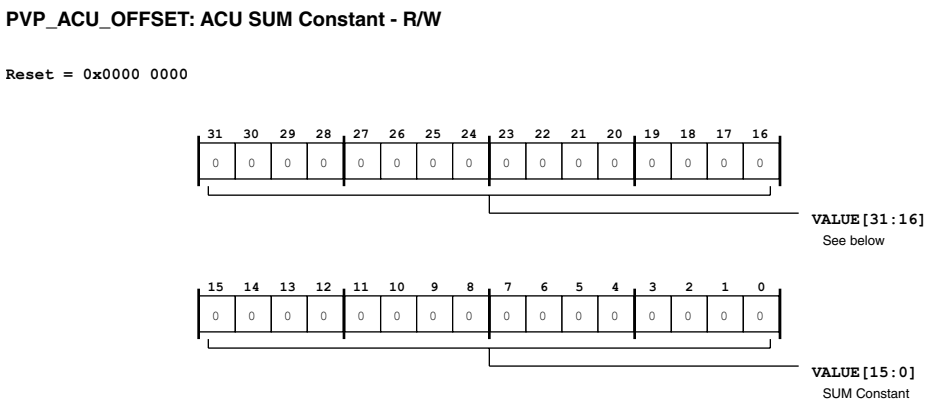


Figure 30-93: PVP_ACU_OFFSET Register Diagram

Table 30-72: PVP_ACU_OFFSET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SUM Constant.

ACU PROD Constant

The PVP_ACU_FACTOR provides the constant value (factor) input for the ACU PROD operation when the PVP_ACU_CTL.PRDINP field selects the factor as one of the inputs for the operation.

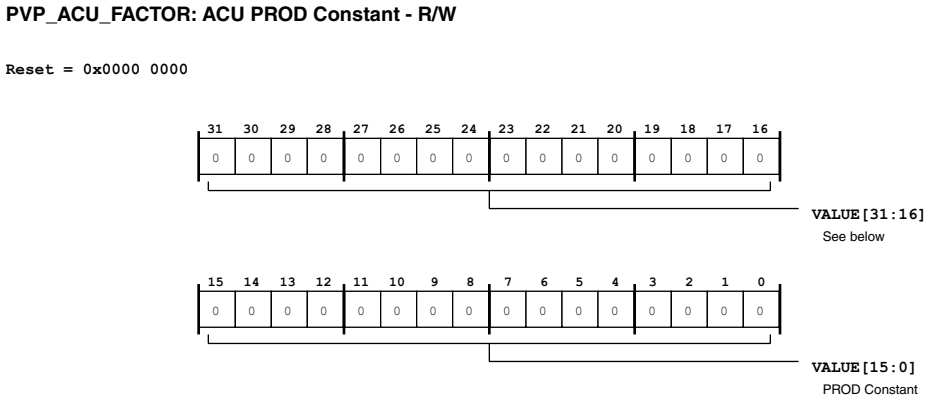


Figure 30-94: PVP_ACU_FACTOR Register Diagram

Table 30-73: PVP_ACU_FACTOR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	PROD Constant.

ACU Shift Constant

The PVP_ACU_SHIFT provides the shift constant value (arithmetic right shift size in bits) for ACC SUM or PROD operations or for 48-bit output from the accumulator. The shift size is from 0 to 31 bits and can be bypassed by programming a value of 0 for the shift constant.

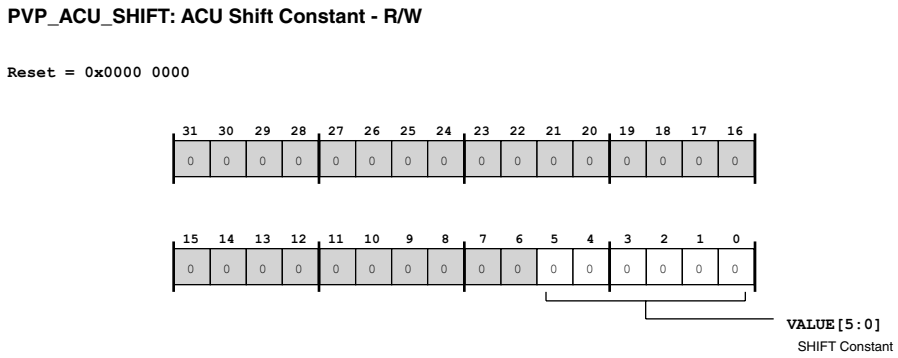


Figure 30-95: PVP_ACU_SHIFT Register Diagram

Table 30-74: PVP_ACU_SHIFT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5:0 (R/W)	VALUE	SHIFT Constant.

ACU Lower Sat Threshold Min

The PVP_ACU_MIN holds the lower threshold value for saturation of ACU shifter output.

PVP_ACU_MIN: ACU Lower Sat Threshold Min - R/W

Reset = 0x8000 0000

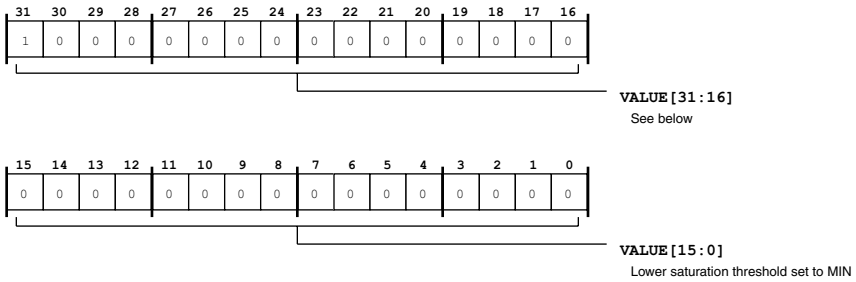


Figure 30-96: PVP_ACU_MIN Register Diagram

Table 30-75: PVP_ACU_MIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Lower saturation threshold set to MIN.

ACU Upper Sat Threshold Max

The PVP_ACU_MAX holds the upper threshold value for saturation of ACU shifter output.

PVP_ACU_MAX: ACU Upper Sat Threshold Max - R/W

Reset = 0x7fff ffff

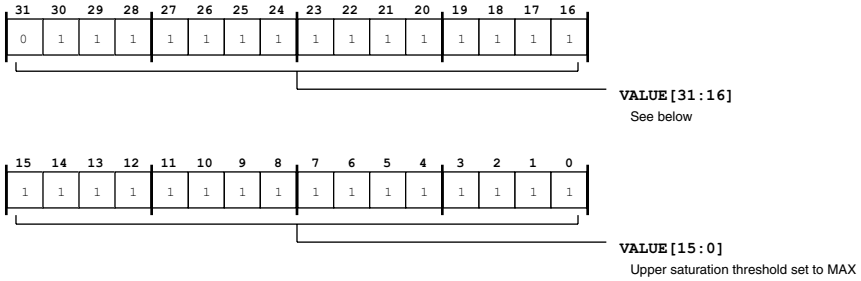


Figure 30-97: PVP_ACU_MAX Register Diagram

Table 30-76: PVP_ACU_MAX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Upper saturation threshold set to MAX.

UDS Configuration

The up-down scaler (UDS) block provides image resizing. The PVP_UDS_CFG register enables the block's FIFO and configures blocks' connections within thePVP.

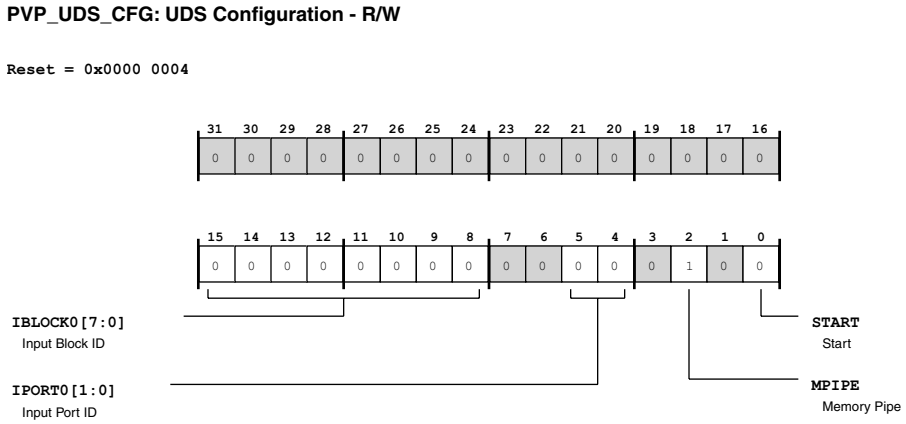


Figure 30-98: PVP_UDS_CFG Register Diagram

Table 30-77: PVP_UDS_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:8 (R/W)	IBLOCK0	Input Block ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_UDS_CFG.IBLOCK0 and PVP_UDS_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s).	
		14	IPF1
5:4 (R/W)	IPORT0	Input Port ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_UDS_CFG.IBLOCK0 and PVP_UDS_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s). For more information, see the connectivity tables for each input block in the architectural concepts section.	
2 (R/NW)	MPIPE	Memory Pipe. The PVP_UDS_CFG.MPIPE bit always reads as 1 (memory pipe).	
0 (R/W)	START	Start. The PVP_UDS_CFG.START enables the PVP. This bit reads 0 after the pipe is drained.	
		1	Enable (W1A)

UDS Control

The PVP_UDS_CTL register selects operation modes for the up-down scaler (UDS) block, including filter tap selection.

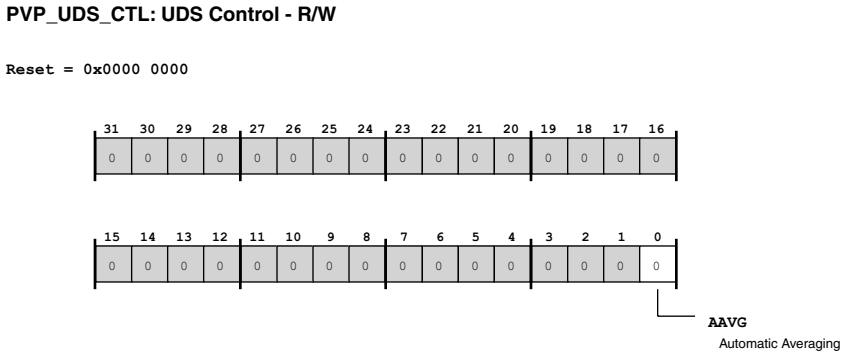


Figure 30-99: PVP_UDS_CTL Register Diagram

Table 30-78: PVP_UDS_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	AAVG	Automatic Averaging. The PVP_UDS_CTL.AAVG selects automatic or manual filter tap selection for the anti-aliasing filter.	
		0	Manual Filter Tap Selection
		1	Auto Filter Tap Selection

UDS Output HCNT

The PVP_UDS_OHCNT selects the horizontal dimension for the output frame. Legal values for the target frame are: 16, 32, 64, 80, 96, 112, and 128.

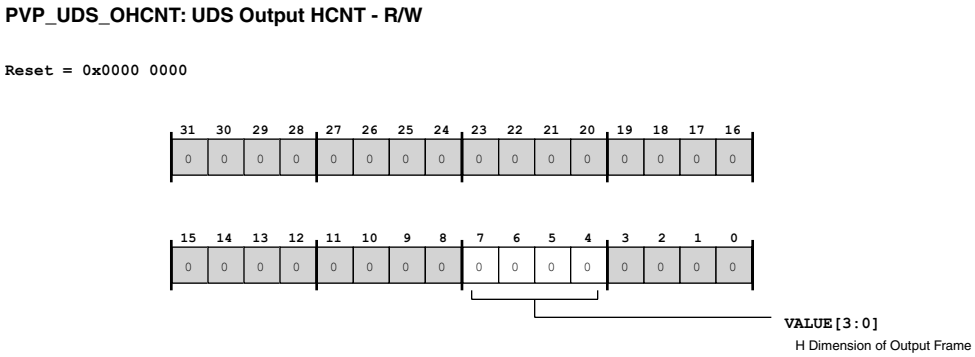


Figure 30-100: PVP_UDS_OHCNT Register Diagram

Table 30-79: PVP_UDS_OHCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/W)	VALUE	H Dimension of Output Frame.

UDS Output VCNT

The PVP_UDS_OVCNT selects the vertical dimension for the output frame. Legal values for the target frame are: 16, 32, 64, 80, 96, 112, and 128.

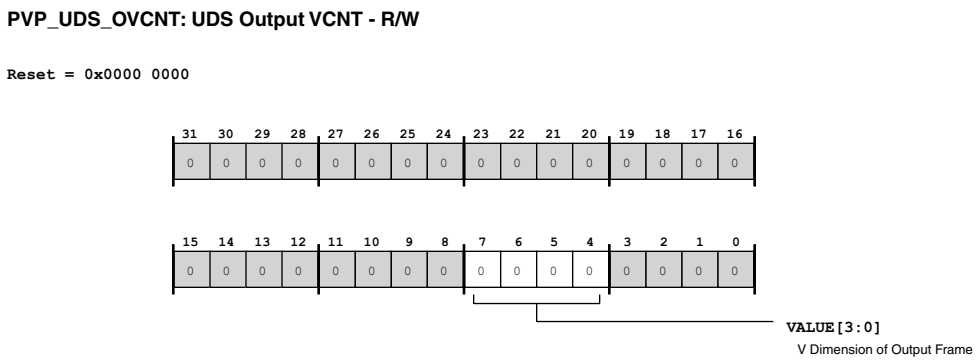


Figure 30-101: PVP_UDS_OVCNT Register Diagram

Table 30-80: PVP_UDS_OVCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/W)	VALUE	V Dimension of Output Frame.

UDS HAVG

The PVP_UDS_HAVG selects the UDS H taps. Legal values for PVP_UDS_HAVG are between 1 and 128.

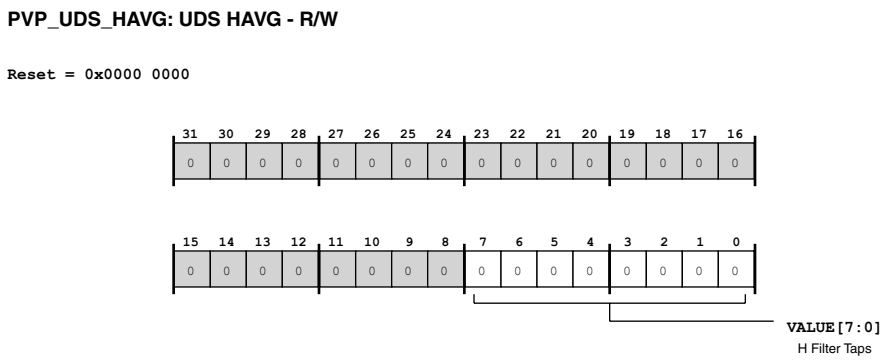


Figure 30-102: PVP_UDS_HAVG Register Diagram

Table 30-81: PVP_UDS_HAVG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	H Filter Taps.

UDS VAVG

The PVP_UDS_VAVG selects the UDS V taps. Legal values for PVP_UDS_VAVG are between 1 and 64.

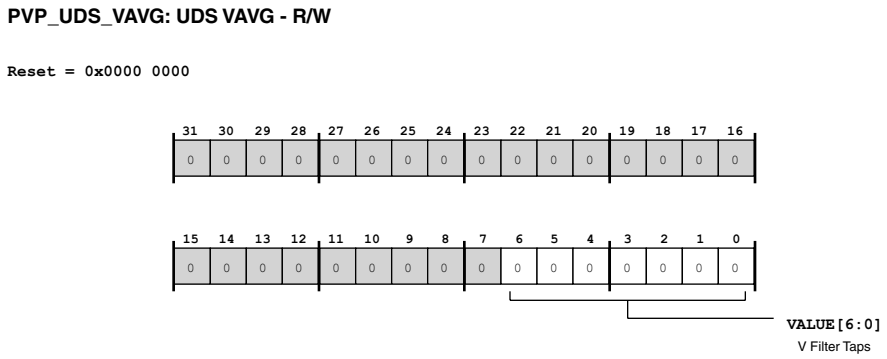


Figure 30-103: PVP_UDS_VAVG Register Diagram

Table 30-82: PVP_UDS_VAVG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	V Filter Taps.

IPF0 (Camera Pipe) Configuration

The input data formatter (IPF) blocks house FIFOs that buffer data for a sustained peak bandwidth output. The PVP_IPF0_CFG register enables the block's FIFO and configures blocks' connections within thePVP.

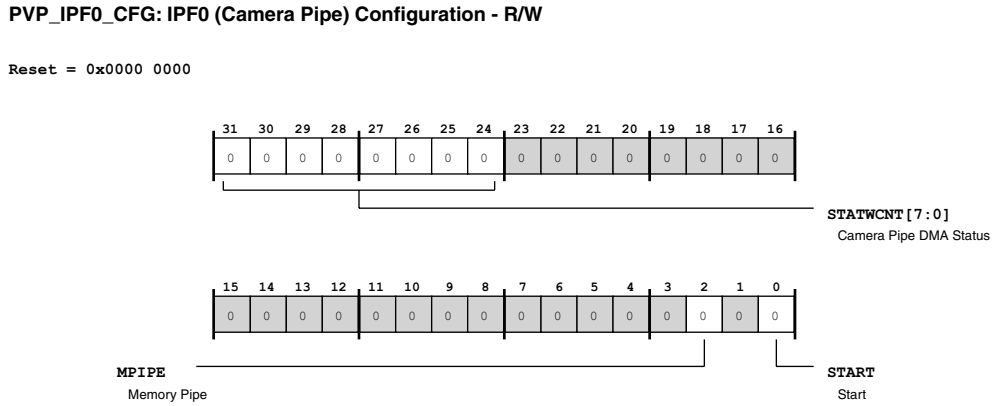


Figure 30-104: PVP_IPF0_CFG Register Diagram

Table 30-83: PVP_IPF0_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:24 (R/W)	STATWCNT	Camera Pipe DMA Status. The PVP_IPF0_CFG.STATWCNT selects whether or not tag status is output on camera pipe status DMA channel. Note that all other values are reserved.	
		0	No Status Output
		1	Tag Status Output
2 (R/NW)	MPIPE	Memory Pipe. The PVP_IPF0_CFG.MPIPE bit always reads as 0 (camera pipe).	
0 (R/W1C)	START	Start. The PVP_IPF0_CFG.START enables the PVP. This bit reads 0 after the pipe is drained.	
		1	Enable (W1A)

IPFn (Camera/Memory Pipe) Pipe Control

The PVP_IPFn_PIPECTL register controls the IPFn block's pipeline features, including status transmission and drain operation.

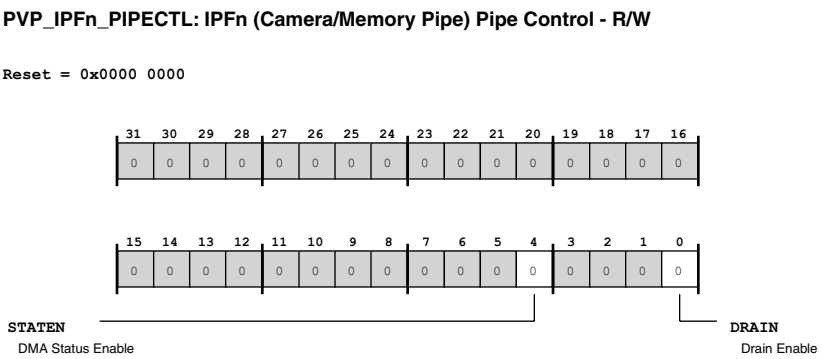


Figure 30-105: PVP_IPFn_PIPECTL Register Diagram

Table 30-84: PVP_IPFn_PIPECTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	STATEN	DMA Status Enable. The PVP_IPFn_PIPECTL.STATEN enables the IPF to send status DMA when PVP_IPFn_FCNT expires or every frame where PVP_IPFn_FCNT equals zero.	
		0	Disable
		1	Enable
0 (R/W)	DRAIN	Drain Enable. The PVP_IPFn_PIPECTL.DRAIN enables the IPF to flush frames automatically in conjunction with PVP_IPFn_FCNT.	
		0	Disable
		1	Enable

IPFn (Camera/Memory Pipe) Control

The PVP_IPFn_CTL register controls the IPFn block's pipeline features, including color format, format conversion, and output port selection.

PVP_IPFn_CTL: IPFn (Camera/Memory Pipe) Control - R/W

Reset = 0x0000 0000

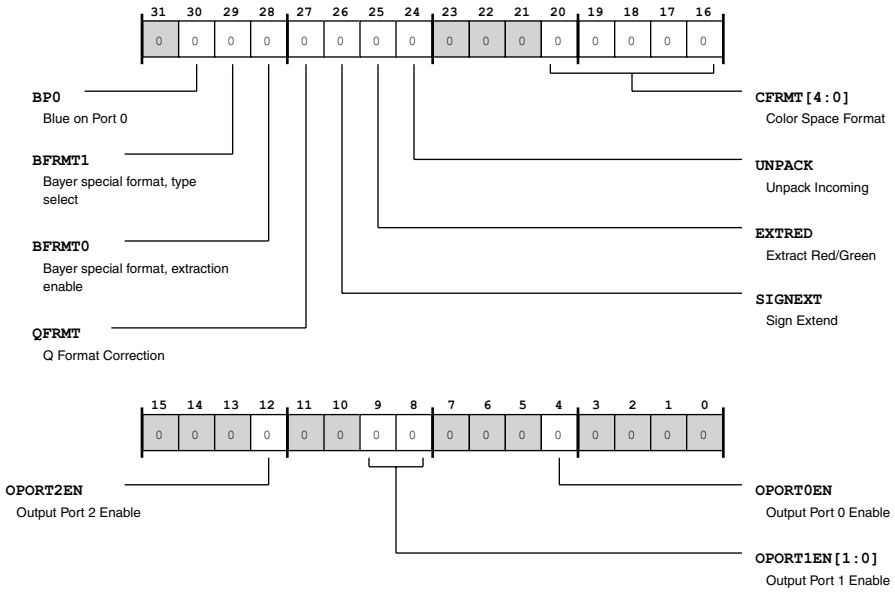


Figure 30-106: PVP_IPFn_CTL Register Diagram

Table 30-85: PVP_IPFn_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	BP0	Blue on Port 0. The PVP_IPFn_CTL.BP0 bit enables B (blue) pixel extraction on Port 0. When enabled and PVP_IPFn_CTL.BFRMT1 =1, only blue pixels are output on output PORT 0. When disabled, the full frame is output on output PORT 0. **Note:** The PVP_IPFn_CTL.BP0 bit influences operation of Port 0 only when the PVP_IPFn_CTL.CFRMT field is set to 0x18 or 0x19 (special Bayer mode) for the PVP_IPF0_CTL register. This bit does not exist in other PVP_IPFn_CTL registers,	
		0	Disable
		1	Enable

Table 30-85: PVP_IPFn_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	BFRMT1	Bayer special format, type select. The PVP_IPFn_CTL.BFRMT1 bit selects whether Port 0, 1, and 2 use Type-1 or Type-2 Bayer format. **Note:** The PVP_IPFn_CTL.BFRMT1 bit influences operation of Port 1 and 2 only when the PVP_IPFn_CTL.CFRMT field is set to 0x18 or 0x19 (special Bayer mode) for the PVP_IPF0_CTL register. This bit does not exist in other PVP_IPFn_CTL registers,	
		0	Type-1 Bayer format
		1	Type-2 Bayer format
28 (R/W)	BFRMT0	Bayer special format, extraction enable. The PVP_IPFn_CTL.BFRMT0 bit enables special Bayer extraction format on Port 1 and 2. When enabled, the R (red) pixels are sent out on output Port 2, and the G (green) pixels (next to R) are sent out on output Port 1. Depending on the value of the PVP_IPFn_CTL.BP0 bit, either the full frame or only extracted B (blue) pixels are output on output Port 0. **Note:** The PVP_IPFn_CTL.BFRMT0 bit influences operation of Port 1 and 2 only when the PVP_IPFn_CTL.CFRMT field is set to 0x18 or 0x19 (special Bayer mode) for the PVP_IPF0_CTL register. This bit does not exist in other PVP_IPFn_CTL registers,	
		0	Disable
		1	Enable
27 (R/W)	QFRMT	Q Format Correction. The PVP_IPFn_CTL.QFRMT enables Q format correction, converting data to unsigned by 1 right shift.	
		0	Disable Q Format Correction
		1	Enable Q Format Correction

Table 30-85: PVP_IPFn_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/W)	SIGNEXT	Sign Extend. The PVP_IPFn_CTL.SIGNEXT selects sign or zero extension for the extracted 5-, 6-, 8-, 16-, or 24-bit value to 32-bit value.	
		0	Zero Extend
		1	Sign Extend
25 (R/W)	EXTRED	Extract Red/Green. The PVP_IPFn_CTL.EXTRED selects red or green extraction from RGB format. This selection does not apply for YUV format.	
		0	Extract Green
		1	Extract Red
24 (R/W)	UNPACK	Unpack Incoming. The PVP_IPFn_CTL.UNPACK enables unpacking for incoming data.	
		0	No Unpacking
		1	Unpack Data

Table 30-85: PVP_IPFn_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
20:16 (R/W)	CFRMT	Color Space Format. The PVP_IPFn_CTL.CFRMT field selects the Color Space Format. Note that all values other than those listed are reserved.	
		0	RGB 8-Bit
		1	RGB 888
		2	RGB 565
		3	RGB 666
		4	RGB 16-Bit
		5	RGB Bayer Format Type-1
		6	RGB Bayer Format Type-2
		16	YUV 4:2:2 8-Bit Type 1
		17	YUV 4:2:2 8-Bit Type 2 Includes with Color Split; this format is only supported in packed mode.
		18	YUV 4:2:2 8-Bit Type 3 Includes Color Split and Sub-Split; this format is only supported in packed mode.
		19	YUV 4:2:2 8-Bit Pair 16-Bit
		20	YUV 4:2:2 16-Bit Type 1
		21	YUV 4:2:2 16-Bit Type 2 Includes Color Split; this format is only supported in packed mode.
		22	YUV 4:2:2 16-Bit Type 3 Includes Color Split and Sub-Split; this format is only supported in packed mode.
		24	Y Alone 8-Bit
		25	Y Alone 16-Bit
		26	Y Alone 24-Bit
		27	32 Bit

Table 30-85: PVP_IPFn_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	OPORT2EN	Output Port 2 Enable. The PVP_IPFn_CTL.OPORT2EN bit enables output Port 2. **Note:** The PVP_IPFn_CTL.BPO, PVP_IPFn_CTL.BFRMT0, and PVP_IPFn_CTL.BFRMT1 bits influence operation of this port only when the PVP_IPFn_CTL.CFRMT field is set to 0x18 or 0x19 (special Bayer mode) for the PVP_IPF0_CTL register. These bits do not exist in the other PVP_IPFn_CTL registers. See the PVP_IPFn_CTL.BPO, PVP_IPFn_CTL.BFRMT0, and PVP_IPFn_CTL.BFRMT1 bit descriptions for more information.	
		0	Disable OPORT2
		1	Enable OPORT2 (full resolution)
9:8 (R/W)	OPORT1EN	Output Port 1 Enable. The PVP_IPFn_CTL.OPORT1EN bits enables output Port 1 and select the port's resolution. **Note:** The PVP_IPFn_CTL.BPO, PVP_IPFn_CTL.BFRMT0, and PVP_IPFn_CTL.BFRMT1 bits influence operation of this port only when the PVP_IPFn_CTL.CFRMT field is set to 0x18 or 0x19 (special Bayer mode) for the PVP_IPF0_CTL register. These bits do not exist in the other PVP_IPFn_CTL registers. See the PVP_IPFn_CTL.BPO, PVP_IPFn_CTL.BFRMT0, and PVP_IPFn_CTL.BFRMT1 bit descriptions for more information.	
		0	Disable OPORT1
		1	Enable OPORT1 (full resolution)
		2	Enable OPORT1 (windowed resolution)

Table 30-85: PVP_IPFn_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	OPORT0EN	Output Port 0 Enable. The PVP_IPFn_CTL.OPORT0EN bit enables output Port 0. **Note:** The PVP_IPFn_CTL.BPO, PVP_IPFn_CTL.BFRMT0, and PVP_IPFn_CTL.BFRMT1 bits influence operation of this port only when the PVP_IPFn_CTL.CFRMT field is set to 0x18 or 0x19 (special Bayer mode) for the PVP_IPF0_CTL register. These bits do not exist in the other PVP_IPFn_CTL registers. See the PVP_IPFn_CTL.BPO, PVP_IPFn_CTL.BFRMT0, and PVP_IPFn_CTL.BFRMT1 bit descriptions for more information.	
		0	Disable OPORT0
		1	Enable OPORT0

IPFn (Camera/Memory Pipe) TAG Value

The PVP_IPFn_TAG holds a unique, user-programmable tag value that indicates to which frame the values were applied. If there are situations where the software does not need information about whether or not a new control word was applied for a new frame, the tag need not be programmed.

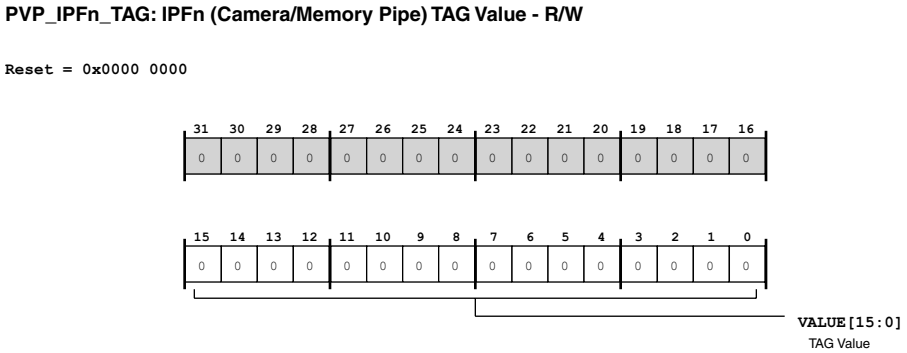


Figure 30-107: PVP_IPFn_TAG Register Diagram

Table 30-86: PVP_IPFn_TAG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	TAG Value.

IPFn (Camera/Memory Pipe) Frame Count

The PVP_IPFn_FCNT whether a given block control structure list (BCL) applies to a set of frames or applies for every new frame.

PVP_IPFn_FCNT: IPFn (Camera/Memory Pipe) Frame Count - R/W

Reset = 0x0000 0000

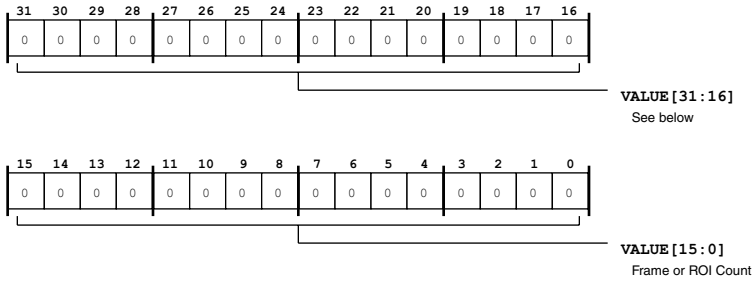


Figure 30-108: PVP_IPFn_FCNT Register Diagram

Table 30-87: PVP_IPFn_FCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Frame or ROI Count.

IPFn (Camera/Memory Pipe) Horizontal Count

The PVP_IPFn_HCNT contains the horizontal count (in pixels) for the region of interest (ROI) starting at the PVP_IPFO_HPOS position.

PVP_IPFn_HCNT: IPFn (Camera/Memory Pipe) Horizontal Count - R/W

Reset = 0x0000 0000

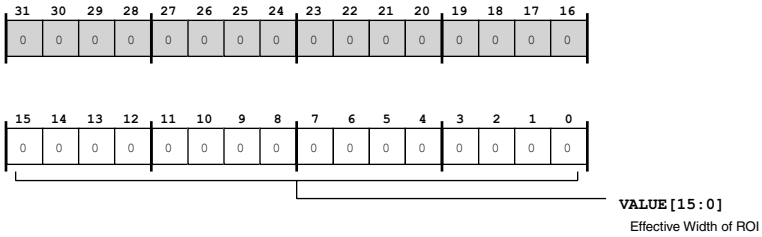


Figure 30-109: PVP_IPFn_HCNT Register Diagram

Table 30-88: PVP_IPFn_HCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Effective Width of ROI.

IPFn (Camera/Memory Pipe) Vertical Count

The PVP_IPFn_VCNT contains the vertical count (in pixels) for the region of interest (ROI) starting at the PVP_IPF0_VPOS position.

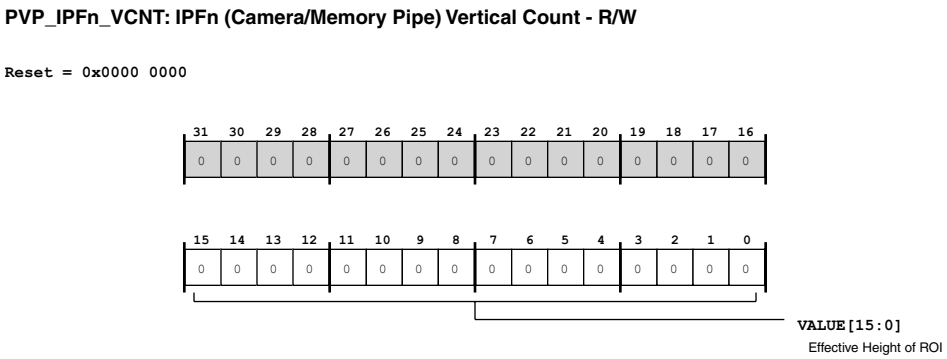


Figure 30-110: PVP_IPFn_VCNT Register Diagram

Table 30-89: PVP_IPFn_VCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Effective Height of ROI.

IPF0 (Camera Pipe) Horizontal Position

The PVP_IPF0_HPOS contains the horizontal starting position (in pixels) for the region of interest (ROI).

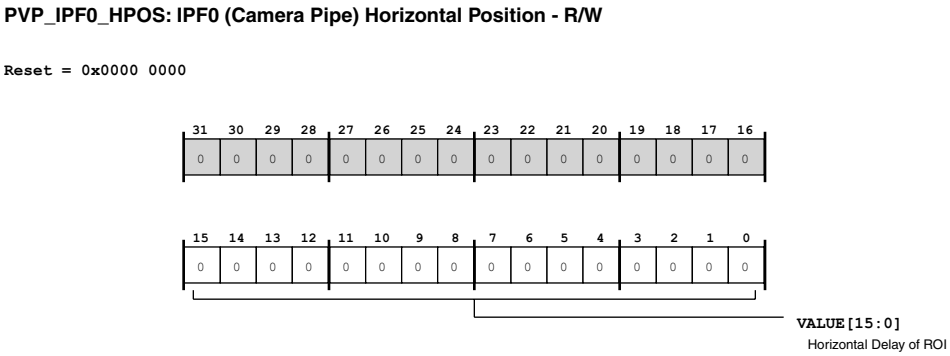


Figure 30-111: PVP_IPF0_HPOS Register Diagram

Table 30-90: PVP_IPF0_HPOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Horizontal Delay of ROI.

IPF0 (Camera Pipe) Vertical Position

The PVP_IPF0_VPOS contains the vertical starting position (in pixels) for the region of interest (ROI).

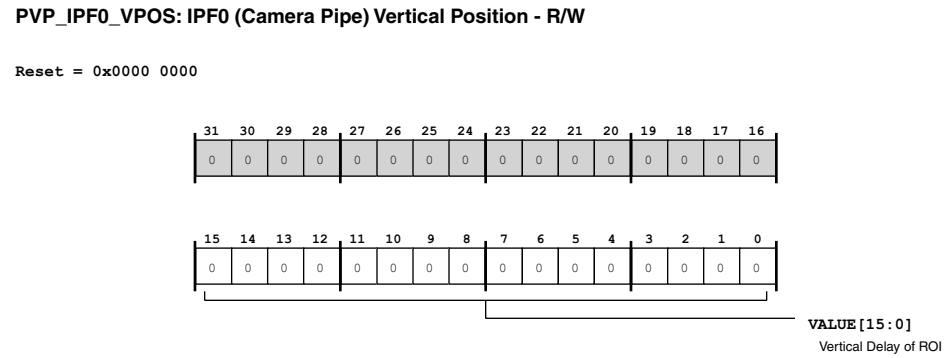


Figure 30-112: PVP_IPF0_VPOS Register Diagram

Table 30-91: PVP_IPF0_VPOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Vertical Delay of ROI.

IPFn (Camera/Memory Pipe) TAG Status

After daisy chain load is completed, the value in PVP_IPFn_TAG is loaded to the PVP_IPFn_TAG_STAT. This register is read-only register and may also be read through status read DMA. The PVP_IPFn_TAG_STAT register is double buffered internally with the PVP_IPFn_TAG_STAT corresponding to previous PVP_IPFn_FCNT entity is sent on status DMA.

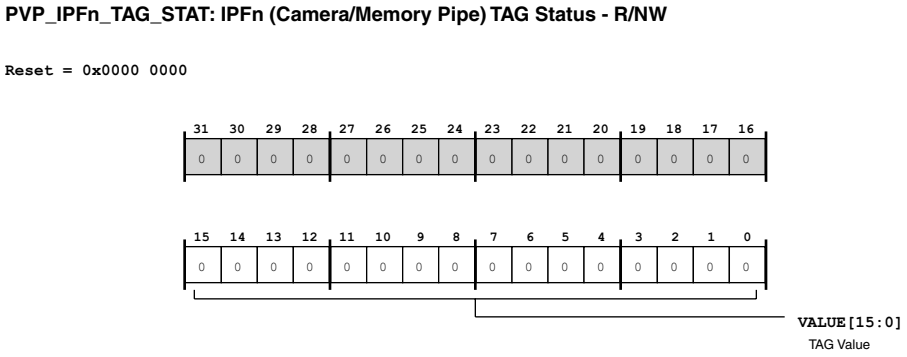


Figure 30-113: PVP_IPFn_TAG_STAT Register Diagram

Table 30-92: PVP_IPFn_TAG_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	TAG Value.

IPF1 (Memory Pipe) Configuration

The PVP_IPF1_CFG register controls the IPF1 block's pipeline features, including status transmission and drain operation.

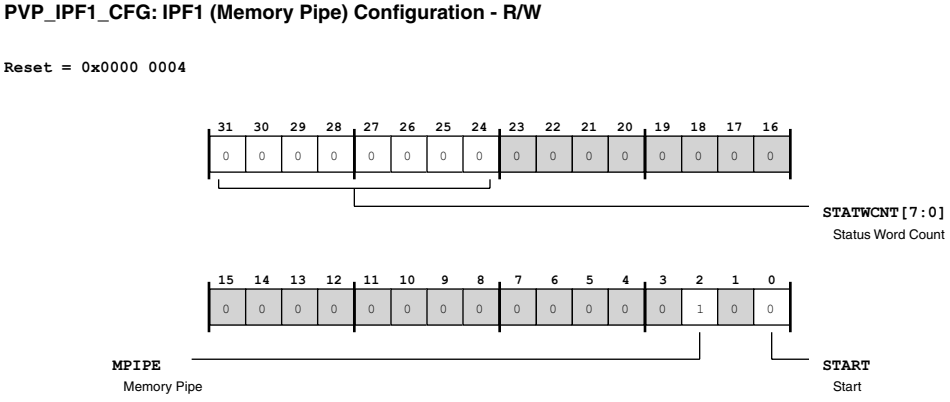


Figure 30-114: PVP_IPF1_CFG Register Diagram

Table 30-93: PVP_IPF1_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:24 (R/W)	STATWCNT	Status Word Count. The PVP_IPF1_CFG.STATWCNT select the camera pipe DMA status information to be output on the memory pipe DMA channel. Note that all other values are reserved.	
		0	No Status Output
		1	Tag Status Output
2 (R/NW)	MPIPE	Memory Pipe. The PVP_IPF1_CFG.MPIPE bit always reads as 1 (memory pipe).	
0 (R/W1C)	START	Start. The PVP_IPF1_CFG.START enables the PVP. This bit reads 0 after the pipe is drained.	
		1	Enable (W1A)

CNVn Configuration

The PVP_CNVn_CFG register controls the convolution/down-scaler engine (CNV) block's pipeline features, including status transmission and drain operation.

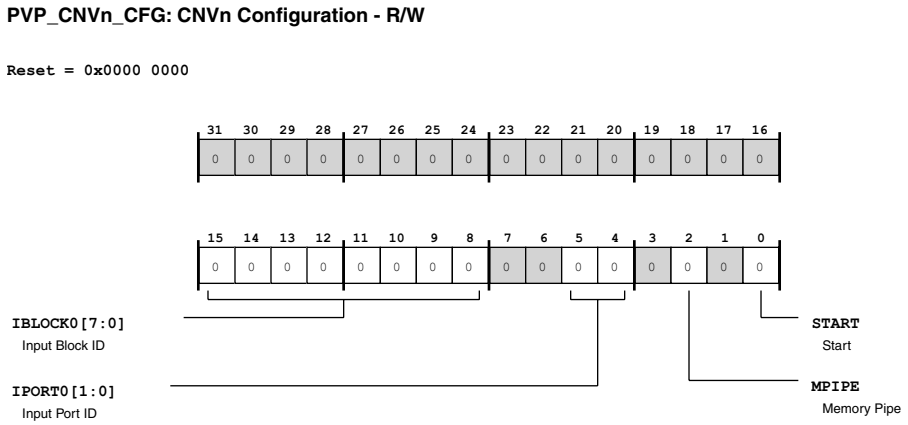


Figure 30-115: PVP_CNVn_CFG Register Diagram

Table 30-94: PVP_CNVn_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:8 (R/W)	IBLOCK0	Input Block ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_CNVn_CFG.IBLOCK0 and PVP_CNVn_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s).	
		12	IPF0
		14	IPF1
		16	CNV0
		20	CNV1
		24	CNV2
		28	CNV3
5:4 (R/W)	IPORT0	Input Port ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_CNVn_CFG.IBLOCK0 and PVP_CNVn_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s). For more information, see the connectivity tables for each input block in the architectural concepts section.	
2 (R/W)	MPIPE	Memory Pipe. The PVP_CNVn_CFG.MPIPE bit selects memory pipe or camera pipe.	
		0	Camera Pipe
		1	Memory Pipe
0 (R/W)	START	Start. The PVP_CNVn_CFG.START enables the PVP. This bit reads 0 after the pipe is drained.	
		1	Enable (W1A)

CNVn Control

The PVP_CNVn_CTL register controls the CNVn block's convolution features, including output saturation, output right shift, and zero/data fill selection.

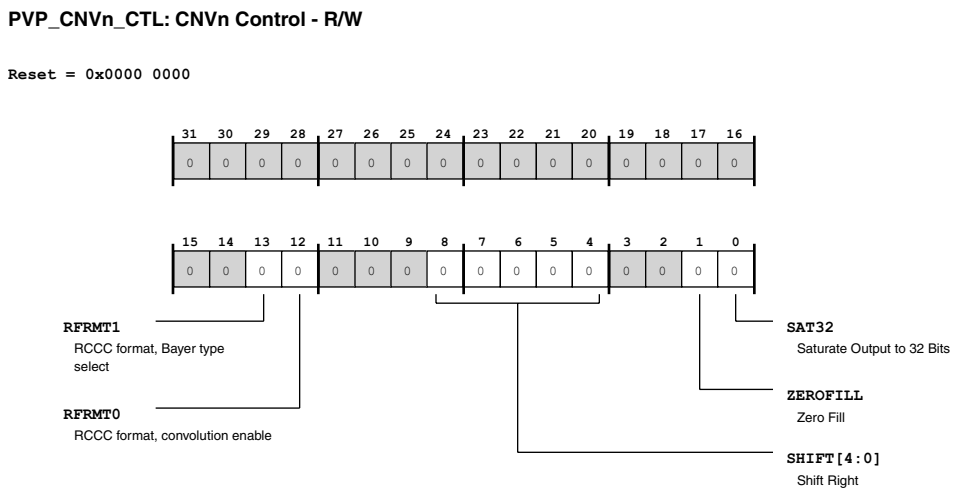


Figure 30-116: PVP_CNVn_CTL Register Diagram

Table 30-95: PVP_CNVn_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	RFRMT1	RCCC format, Bayer type select. The PVP_CNVn_CTL.RFRMT1 bit selects Bayer type-1 or Bayer type-2 specific convolution for the CNV1 block. **Note:** The PVP_CNVn_CTL.RFRMT1 bit influences operation of the CNV1 block for the PVP_CNV1_CTL register. This bit does not exist in other PVP_CNVn_CTL registers,	
		0	Type-1 Bayer format
		1	Type-2 Bayer format

Table 30-95: PVP_CNVn_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	RFRMT0	RCCC format, convolution enable. The PVP_CNVn_CTL.RFRMT0 bit enables applying specific convolution only on R (red) pixels of RCCC data. SHIFT bits do not apply to the bypassed pixels and apply only to the convolved R pixels. Scaling has to be disabled on CNV1 when this mode is enabled. When PVP_CNVn_CTL.RFRMT0 is enabled, the PVP_CNVn_CTL.SAT32 bit must =0. The bypassed input pixels are sign extended to 32 bits, while the convolved pixels are saturated to 16 bits and sign extended to 32 bits. **Note:** The PVP_CNVn_CTL.RFRMT0 bit influences operation of the CNV1 block for the PVP_CNV1_CTL register. This bit does not exist in other PVP_CNVn_CTL registers,	
		0	Disable (CONV on all pixels)
		1	Enable (CONV only on R pixels of RCCC)
8:4 (R/W)	SHIFT	Shift Right. The PVP_CNVn_CTL.SHIFT selects arithmetic right shift of output from 0 to 31 bits.	
1 (R/W)	ZEROFILL	Zero Fill. The PVP_CNVn_CTL.ZEROFILL select whether to fill edge data with zeroes or with duplicated pixels.	
		0	Duplicated Data Fill
		1	Zero Fill
0 (R/W)	SAT32	Saturate Output to 32 Bits. The PVP_CNVn_CTL.SAT32 selects whether to saturate output to highest 32-bit value or highest 16-bit value. For 16-bit saturate of output, the value is driven on the lower 16 bits, and the upper 16 bits are sign-extended.	
		0	16-Bit Saturate of Output
		1	32-Bit Saturate of Output

CNVn Coefficients 0,0 and 0,1

The PVP_CNVn_C00C01 provides coefficients 0,0 and 0,1 for the convolution matrix.

PVP_CNVn_C00C01: CNVn Coefficients 0,0 and 0,1 - R/W

Reset = 0x0000 0000

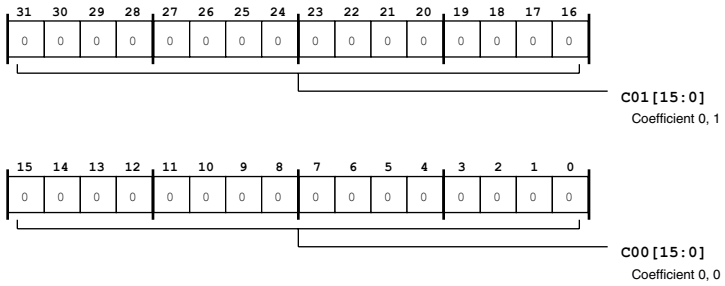


Figure 30-117: PVP_CNVn_C00C01 Register Diagram

Table 30-96: PVP_CNVn_C00C01 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	C01	Coefficient 0, 1.
15:0 (R/W)	C00	Coefficient 0, 0.

CNVn Coefficients 0,2 and 0,3

The PVP_CNVn_C02C03 provides coefficients 0,2 and 0,3 for the convolution matrix.

PVP_CNVn_C02C03: CNVn Coefficients 0,2 and 0,3 - R/W

Reset = 0x0000 0000

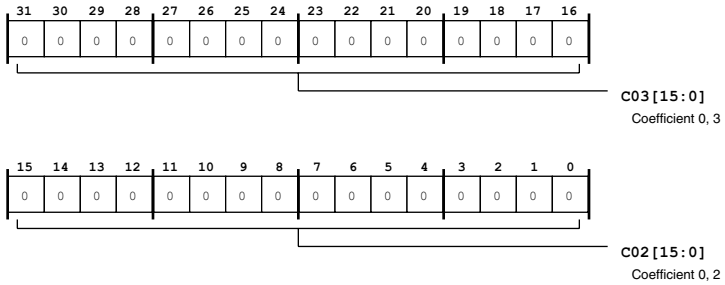


Figure 30-118: PVP_CNVn_C02C03 Register Diagram

Table 30-97: PVP_CNVn_C02C03 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	C03	Coefficient 0, 3.
15:0 (R/W)	C02	Coefficient 0, 2.

CNVn Coefficient 0,4

The `PVP_CNVn_C04` provides coefficients 0,4 for the convolution matrix.

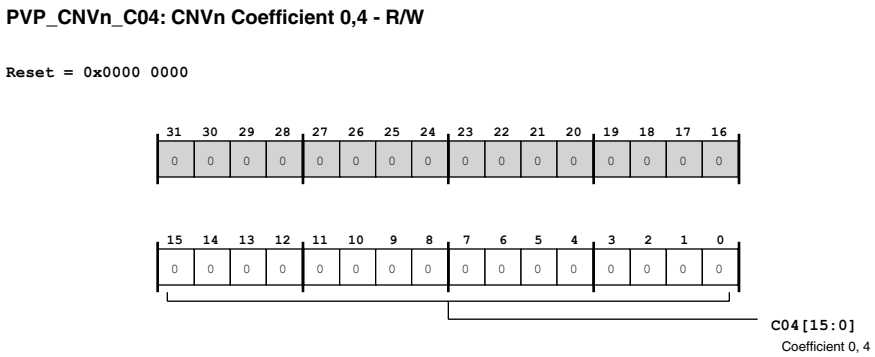


Figure 30-119: PVP_CNVn_C04 Register Diagram

Table 30-98: PVP_CNVn_C04 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	C04	Coefficient 0, 4.

CNVn Coefficients 1,0 and 1,1

The `PVP_CNVn_C10C11` provides coefficients 1,0 and 1,1 for the convolution matrix.

PVP_CNVn_C10C11: CNVn Coefficients 1,0 and 1,1 - R/W

Reset = 0x0000 0000

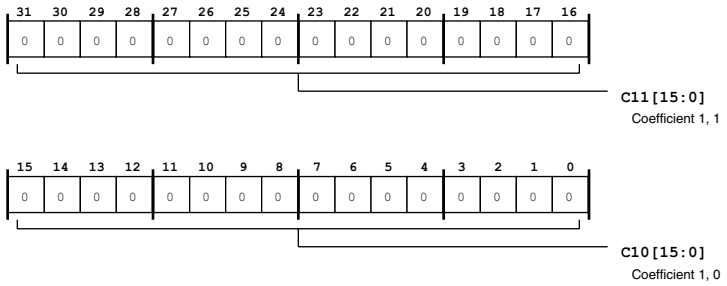


Figure 30-120: PVP_CNVn_C10C11 Register Diagram

Table 30-99: PVP_CNVn_C10C11 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	C11	Coefficient 1, 1.
15:0 (R/W)	C10	Coefficient 1, 0.

CNVn Coefficients 1,2 and 1,3

The `PVP_CNVn_C12C13` provides coefficients 1,2 and 1,3 for the convolution matrix.

PVP_CNVn_C12C13: CNVn Coefficients 1,2 and 1,3 - R/W

Reset = 0x0000 0000

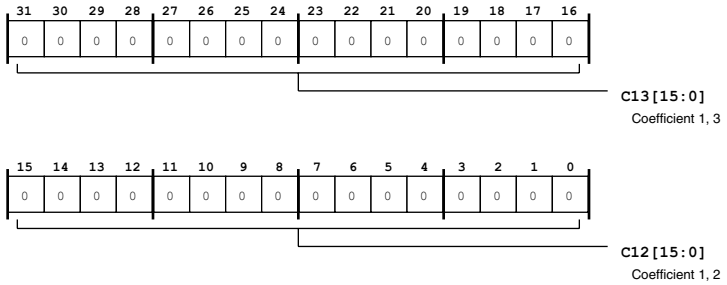


Figure 30-121: PVP_CNVn_C12C13 Register Diagram

Table 30-100: PVP_CNVn_C12C13 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	C13	Coefficient 1, 3.
15:0 (R/W)	C12	Coefficient 1, 2.

CNVn Coefficient 1,4

The PVP_CNVn_C14 provides coefficients 1,4 for the convolution matrix.

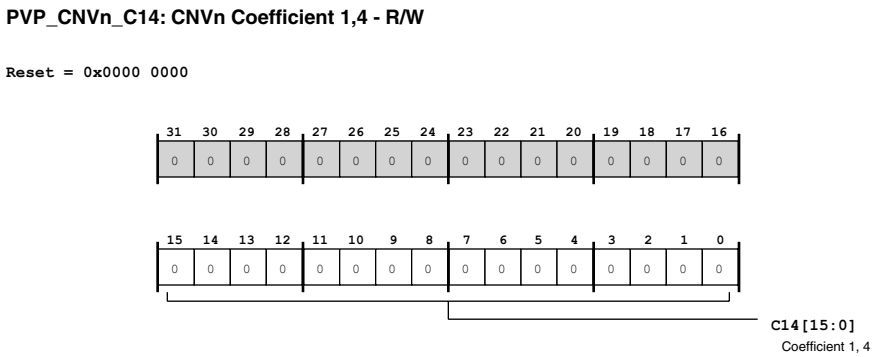


Figure 30-122: PVP_CNVn_C14 Register Diagram

Table 30-101: PVP_CNVn_C14 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	C14	Coefficient 1, 4.

CNVn Coefficients 2,0 and 2,1

The PVP_CNVn_C20C21 provides coefficients 2,0 and 2,1 for the convolution matrix.

PVP_CNVn_C20C21: CNVn Coefficients 2,0 and 2,1 - R/W

Reset = 0x0000 0000

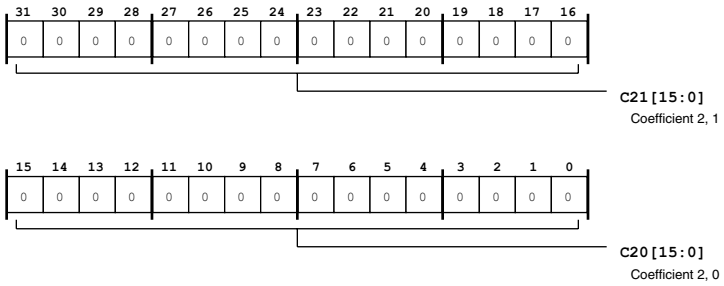


Figure 30-123: PVP_CNVn_C20C21 Register Diagram

Table 30-102: PVP_CNVn_C20C21 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	C21	Coefficient 2, 1.
15:0 (R/W)	C20	Coefficient 2, 0.

CNVn Coefficients 2,2 and 2,3

The PVP_CNVn_C22C23 provides coefficients 2,2 and 2,3 for the convolution matrix.

PVP_CNVn_C22C23: CNVn Coefficients 2,2 and 2,3 - R/W

Reset = 0x0000 0000

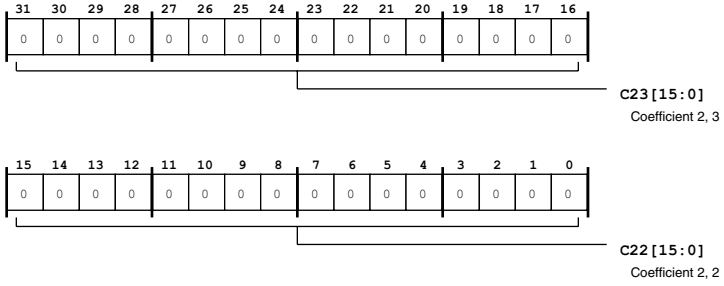


Figure 30-124: PVP_CNVn_C22C23 Register Diagram

Table 30-103: PVP_CNVn_C22C23 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	C23	Coefficient 2, 3.
15:0 (R/W)	C22	Coefficient 2, 2.

CNVn Coefficient 2,4

The PVP_CNVn_C24 provides coefficients 2,4 for the convolution matrix.

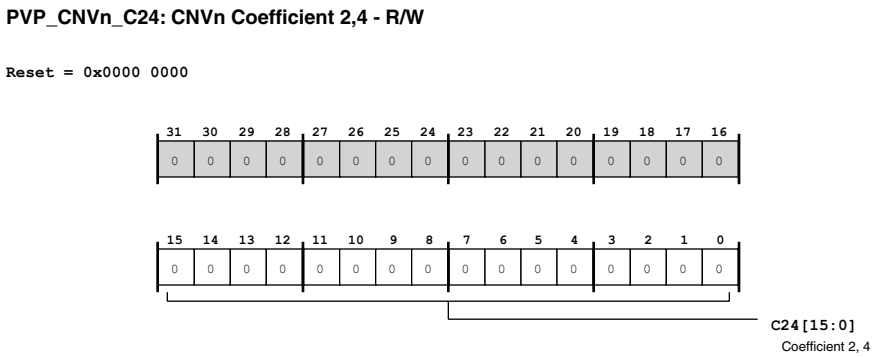


Figure 30-125: PVP_CNVn_C24 Register Diagram

Table 30-104: PVP_CNVn_C24 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	C24	Coefficient 2, 4.

CNVn Coefficients 3,0 and 3,1

The PVP_CNVn_C30C31 provides coefficients 3,0 and 3,1 for the convolution matrix.

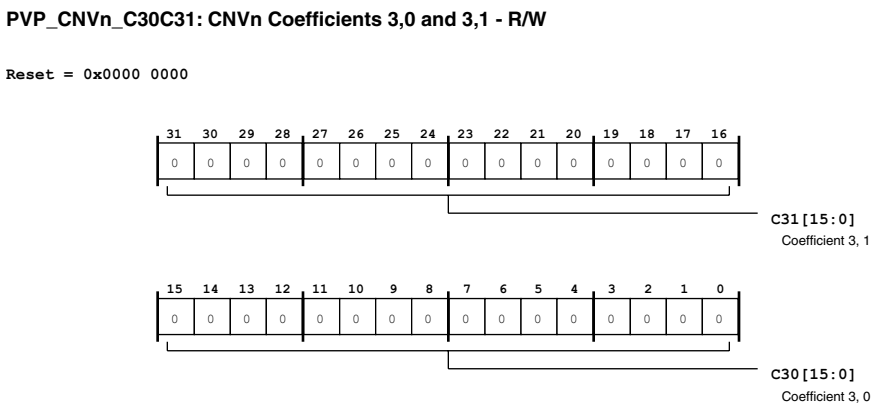


Figure 30-126: PVP_CNVn_C30C31 Register Diagram

Table 30-105: PVP_CNVn_C30C31 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	C31	Coefficient 3, 1.
15:0 (R/W)	C30	Coefficient 3, 0.

CNVn Coefficients 3,2 and 3,3

The `PVP_CNVn_C32C33` provides coefficients 3,2 and 3,3 for the convolution matrix.

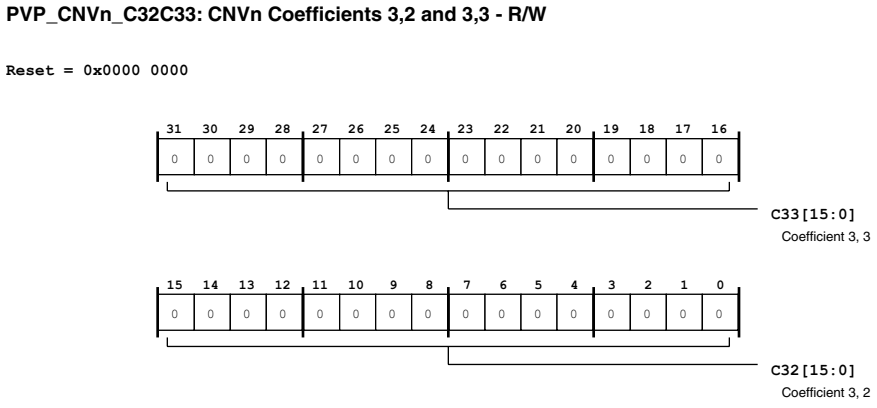


Figure 30-127: PVP_CNVn_C32C33 Register Diagram

Table 30-106: PVP_CNVn_C32C33 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	C33	Coefficient 3, 3.
15:0 (R/W)	C32	Coefficient 3, 2.

CNVn Coefficient 3,4

The PVP_CNVn_C34 provides coefficients 3,4 for the convolution matrix.

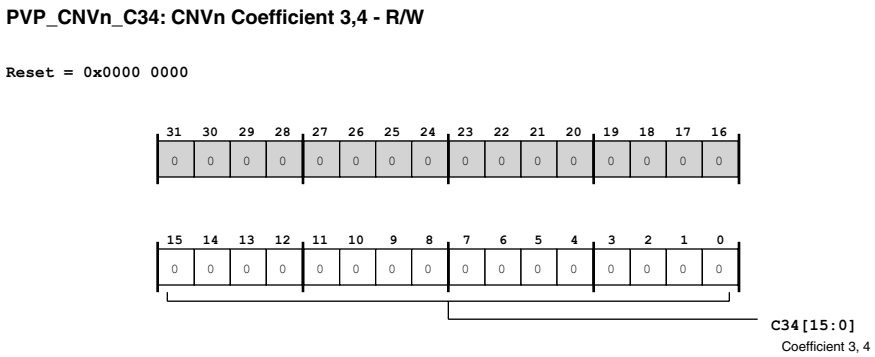


Figure 30-128: PVP_CNVn_C34 Register Diagram

Table 30-107: PVP_CNVn_C34 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	C34	Coefficient 3, 4.

CNVn Coefficients 4,0 and 4,1

The PVP_CNVn_C40C41 provides coefficients 4,0 and 4,1 for the convolution matrix.

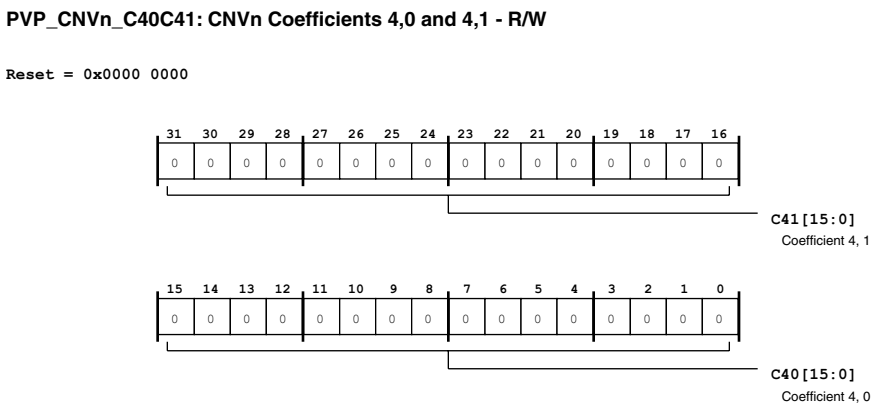


Figure 30-129: PVP_CNVn_C40C41 Register Diagram

Table 30-108: PVP_CNVn_C40C41 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	C41	Coefficient 4, 1.
15:0 (R/W)	C40	Coefficient 4, 0.

CNVn Coefficients 4,2 and 4,3

The PVP_CNVn_C42C43 provides coefficients 4,2 and 4,3 for the convolution matrix.

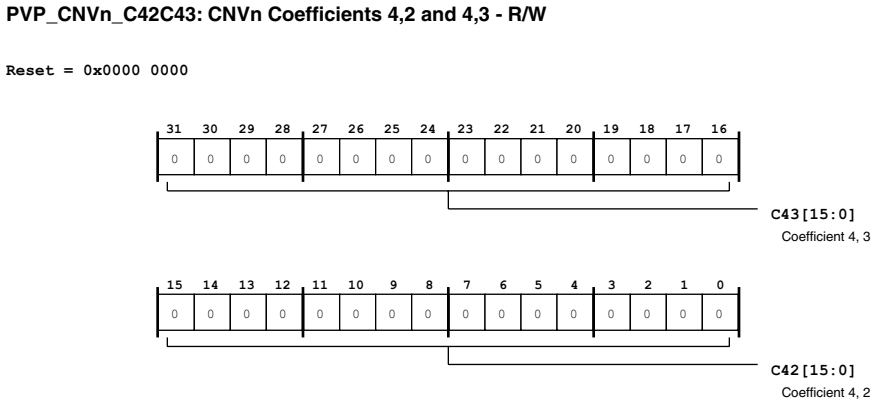


Figure 30-130: PVP_CNVn_C42C43 Register Diagram

Table 30-109: PVP_CNVn_C42C43 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	C43	Coefficient 4, 3.
15:0 (R/W)	C42	Coefficient 4, 2.

CNVn Coefficient 4,4

The PVP_CNVn_C44 provides coefficients 4,4 for the convolution matrix.

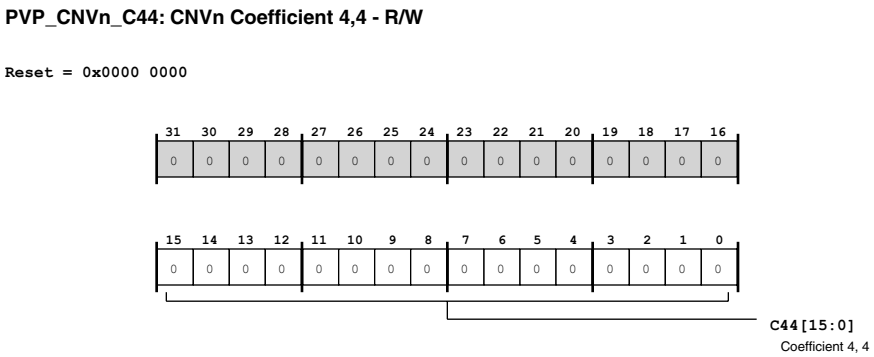


Figure 30-131: PVP_CNVn_C44 Register Diagram

Table 30-110: PVP_CNVn_C44 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	C44	Coefficient 4, 4.

CNVn Scaling Factor

The PVP_CNVn_SCALE holds the vertical and horizontal scaling factors for down scaling incoming frames with decimation ratios of up to 256 (for example, 1, 2, 4, 8, 16, 32, 64, 128, or 256) in either direction.

PVP_CNVn_SCALE: CNVn Scaling Factor - R/W

Reset = 0x0000 0000

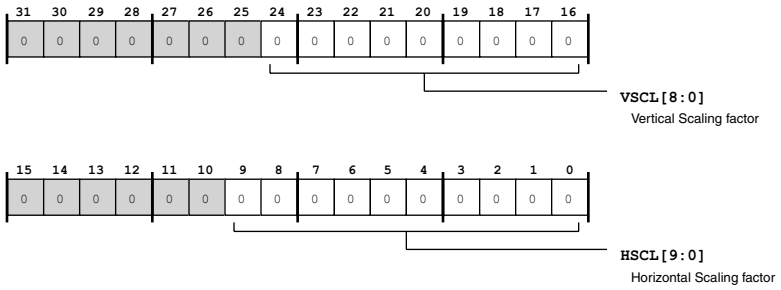


Figure 30-132: PVP_CNVn_SCALE Register Diagram

Table 30-111: PVP_CNVn_SCALE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24:16 (R/W)	VSCL	Vertical Scaling factor.
9:0 (R/W)	HSCL	Horizontal Scaling factor.

THCn Configuration

The PVP_THCn_CFG register controls the threshold/histogram and compression engine (THC) block's pipeline features, including status transmission, port selection, and block enable.

PVP_THCn_CFG: THCn Configuration - R/W

Reset = 0x0000 0000

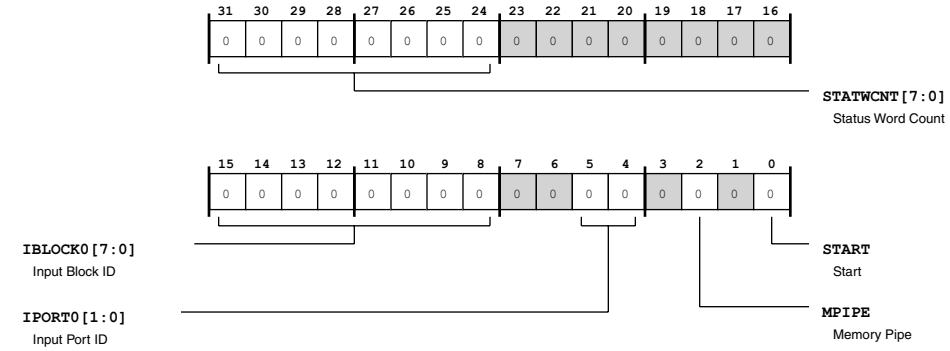


Figure 30-133: PVP_THCn_CFG Register Diagram

Table 30-112: PVP_THCn_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:24 (R/W)	STATWCNT	Status Word Count. The PVP_THCn_CFG.STATWCNT select the camera pipe status information to be output on the memory pipe DMA channel. Note that all other values are reserved.	
15:8 (R/W)	IBLOCK0	Input Block ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_THCn_CFG.IBLOCK0 and PVP_THCn_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s).	
		5	PEC
		8	ACU
		12	IPF0
		14	IPF1
		16	CNV0
		20	CNV1
		24	CNV2
		28	CNV3
		48	PMA
5:4 (R/W)	IPORT0	Input Port ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_THCn_CFG.IBLOCK0 and PVP_THCn_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s). For more information, see the connectivity tables for each input block in the architectural concepts section.	
2 (R/W)	MPIPE	Memory Pipe. The PVP_THCn_CFG.MPIPE bit selects memory pipe or camera pipe.	
		0	Camera Pipe
		1	Memory Pipe
0 (R/W)	START	Start. The PVP_THCn_CFG.START enables the PVP. This bit reads 0 after the pipe is drained.	
		1	Enable (W1A)

THCn Control

The PVP_THCn_CTL register controls the THCn block's threshold/histogram features, including histogram window and counters, output format, and threshold mode selection.

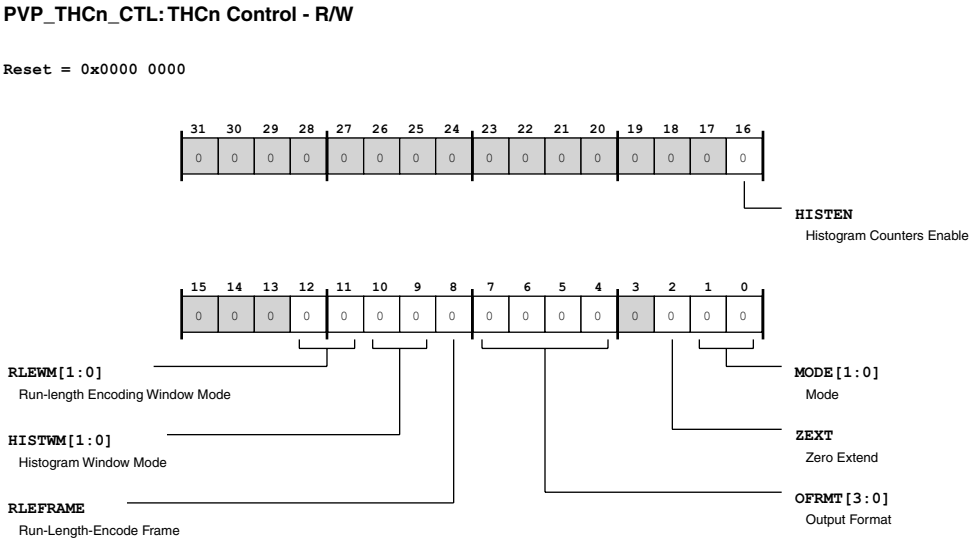


Figure 30-134: PVP_THCn_CTL Register Diagram

Table 30-113: PVP_THCn_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	HISTEN	Histogram Counters Enable. The PVP_THCn_CTL.HISTEN enables histogram counters, enabling the histogram operation.	
		0	Disable
		1	Enable
12:11 (R/W)	RLEWM	Run-length Encoding Window Mode. The PVP_THCn_CTL.RLEWM selects the window mode for the compression block, choosing either compression for pixels inside the window or compression for pixels in the entire frame. Note that values other than shown are invalid.	
		0	Frame Compression
		1	Window Compression

Table 30-113: PVP_THCn_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10:9 (R/W)	HISTWM	Histogram Window Mode. The PVP_THCn_CTL.HISTWM selects window mode for the histogram block, choosing either histogram for pixels inside the window, for pixels outside the window, or for pixels in the entire frame.	
		0	Frame Histogram
		1	Inside-Window Histogram
		2	Outside-Window Histogram
		3	Reserved
8 (R/W)	RLEFRAME	Run-Length-Encode Frame. The PVP_THCn_CTL.RLEFRAME selects the scope for run-length compression as row or frame. When row is selected, the compression block generates a report at the end of a row and the run-length counter is reset. When frame is selected, compression works across rows in the frame. Note that, due to the nature of compression, a variable number of reports are generated per row or per frame.	
		0	Row (Line) Compression
		1	Frame Compression

Table 30-113: PVP_THCn_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7:4 (R/W)	OFRMT	Output Format. The PVP_THCn_CTL.OFRMT selects the THC block's output format, which controls how the compression value and run-length are grouped into a single word. Note that values other than those shown are invalid. Also note that, depending on the PVP_THCn_CTL.OFRMT value , the appropriate number of MSBs are driven with zero value to complete the 32 bit output.	
		0	32-Bit Word (No Compression)
		1	Reserved
		2	4-Bit Index (No Compression)
		3	4-Bit Index / 4-Bit Run Length
		4	4-Bit Index / 4-Bit angle (No Compression)
		5	3-Bit Index / 5-Bit Run Length
		6	4-Bit Index / 12-Bit Run Length
		7	3-Bit Index / 13-Bit Run Length
		8	4-Bit Index / 21-Bit Run Length
		9	16-Bit Word / 16-Bit Run Length
		10	Disable Output/RLE No data is driven on the output port, and the RLE block is switched off.
2 (R/W)	ZEXT	Zero Extend. The PVP_THCn_CTL.ZEXT directs the THC to zero extend the lower 16 bits of the input port to 32 bits for magnitude and to use the upper 16 bits as angle. If PVP_THCn_CTL.ZEXT is disabled, the THC uses the entire 32 bits of the input port for magnitude and treats the angle as all zeroes.	
		0	No Zero Extension
		1	Zero Extend

Table 30-113: PVP_THCn_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	MODE	Mode. The PVP_THCn_CTL.MODE selects the THC mode as clipping mode, quantization mode, or hysteresis mode. For more information about these modes, see the PVP architectural concepts section.	
		0	Clipping/Saturation Mode
		1	Quantization Mode
		2	Hysteresis Mode
		3	Reserved

THCn Histogram Frame Count

The PVP_THCn_HFCNT holds the number of frames for which the histogram counters accumulate. The PVP_THCn_HFCNT_STAT register counts frames up to the value set in PVP_THCn_HFCNT for histogram operation.

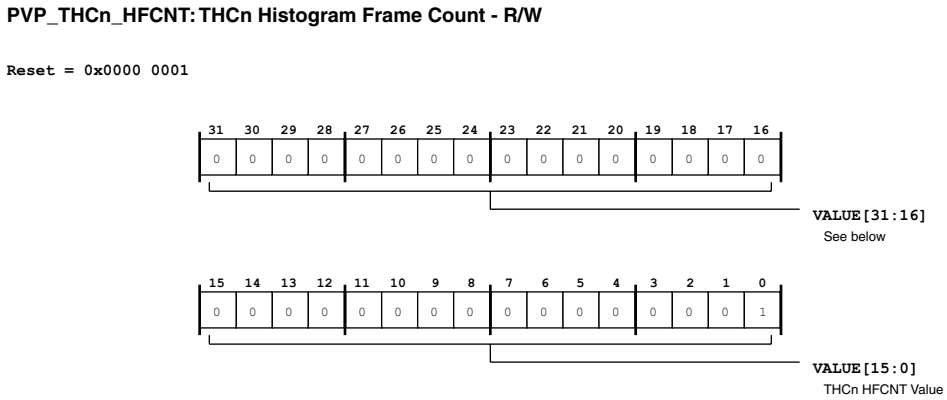


Figure 30-135: PVP_THCn_HFCNT Register Diagram

Table 30-114: PVP_THCn_HFCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	THCn HFCNT Value.

THCn Max RLE Reports

This PVP_THCn_RMAXREP controls the maximum number of RLE reports per line or per frame.

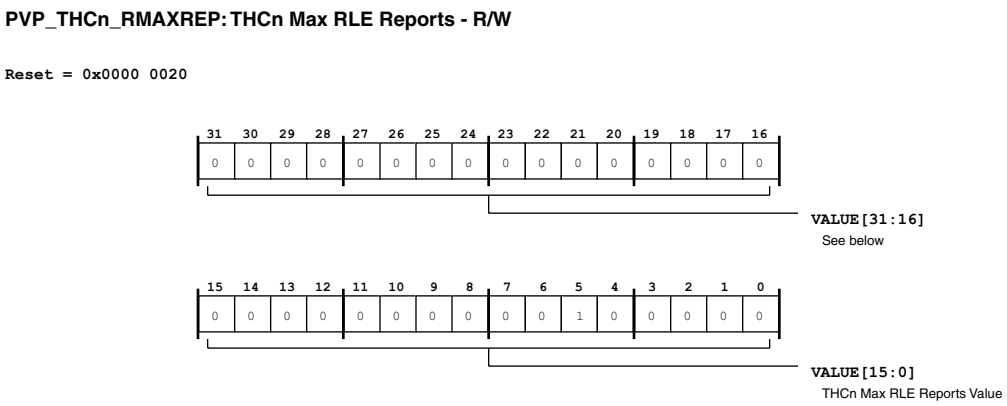


Figure 30-136: PVP_THCn_RMAXREP Register Diagram

Table 30-115: PVP_THCn_RMAXREP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	THCn Max RLE Reports Value.

THCn Min Clip Value

The PVP_THCn_CMINVAL holds the minimum clip value, which the THC uses in clipping/saturation mode. For more information about this mode and the registers involved in its operation (PVP_THCn_CMINVAL, PVP_THCn_CMINTH, PVP_THCn_CMAXTH,PVP_THCn_CMAXVAL), see the PVP functional description.

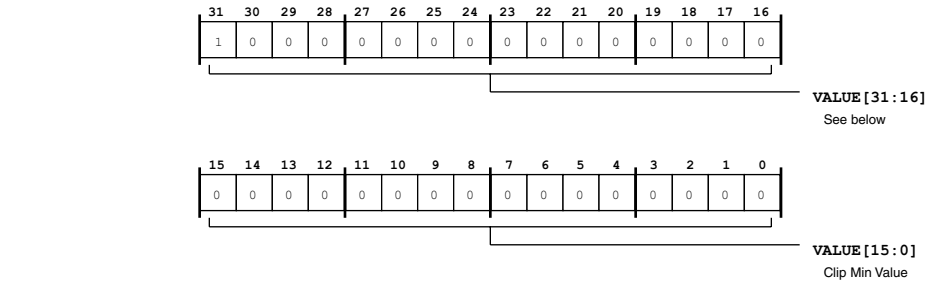


Figure 30-137: PVP_THCn_CMINVAL Register Diagram

Table 30-116: PVP_THCn_CMINVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Clip Min Value.

THCn Clip Min Threshold

The PVP_THCn_CMINTH holds the minimum clip value, which the THC uses in clipping/saturation mode. For more information about this mode and the registers involved in its operation (PVP_THCn_CMINVAL, PVP_THCn_CMINTH, PVP_THCn_CMAXTH,PVP_THCn_CMAXVAL), see the PVP functional description.

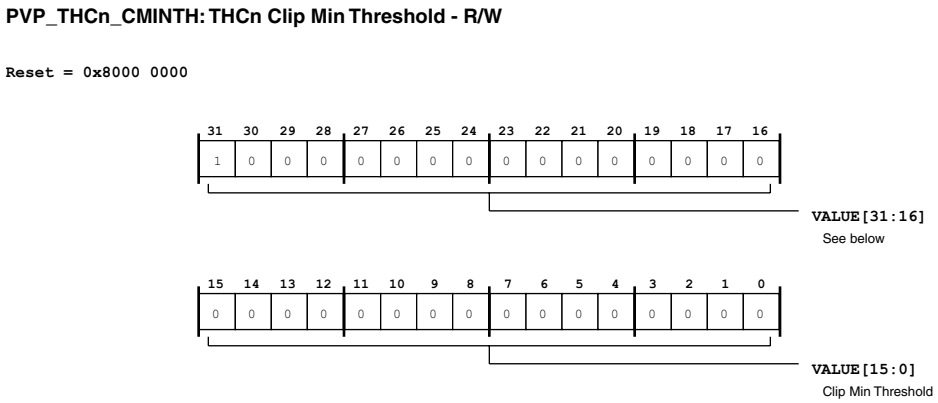


Figure 30-138: PVP_THCn_CMINTH Register Diagram

Table 30-117: PVP_THCn_CMINTH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Clip Min Threshold.

THCn Clip Max Threshold

The PVP_THCn_CMAXTH holds the minimum clip value, which the THC uses in clipping/saturation mode. For more information about this mode and the registers involved in its operation (PVP_THCn_CMINVAL, PVP_THCn_CMINTH, PVP_THCn_CMAXTH,PVP_THCn_CMAXVAL), see the PVP functional description.

PVP_THCn_CMAXTH: THCn Clip Max Threshold - R/W

Reset = 0x7fff ffff

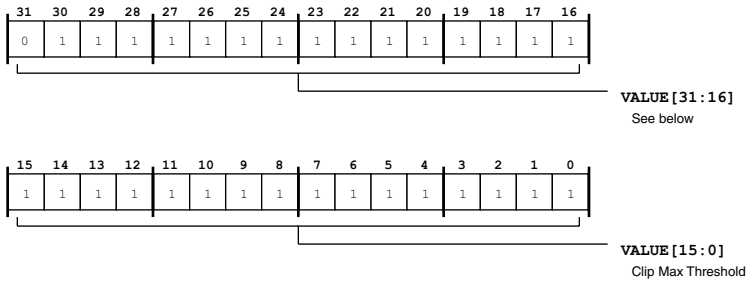


Figure 30-139: PVP_THCn_CMAXTH Register Diagram

Table 30-118: PVP_THCn_CMAXTH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Clip Max Threshold.

THCn Max Clip Value

The PVP_THCn_CMAXVAL holds the minimum clip value, which the THC uses in clipping/saturation mode. For more information about this mode and the registers involved in its operation (PVP_THCn_CMINVAL, PVP_THCn_CMINTH, PVP_THCn_CMAXTH,PVP_THCn_CMAXVAL), see the PVP functional description.

PVP_THCn_CMAXVAL: THCn Max Clip Value - R/W

Reset = 0x7fff ffff

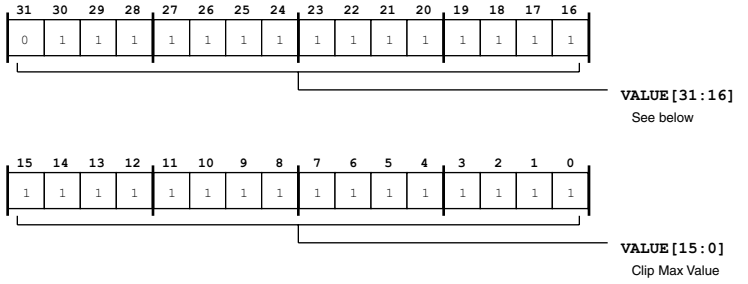


Figure 30-140: PVP_THCn_CMAXVAL Register Diagram

Table 30-119: PVP_THCn_CMAXVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Clip Max Value.

THCn Threshold Value 0

The PVP_THCn_TH0 holds one of up to 16 threshold values, which the THC uses in quantization mode and hysteresis mode. For more information about these modes and the registers involved in their operation, see the PVP functional description.

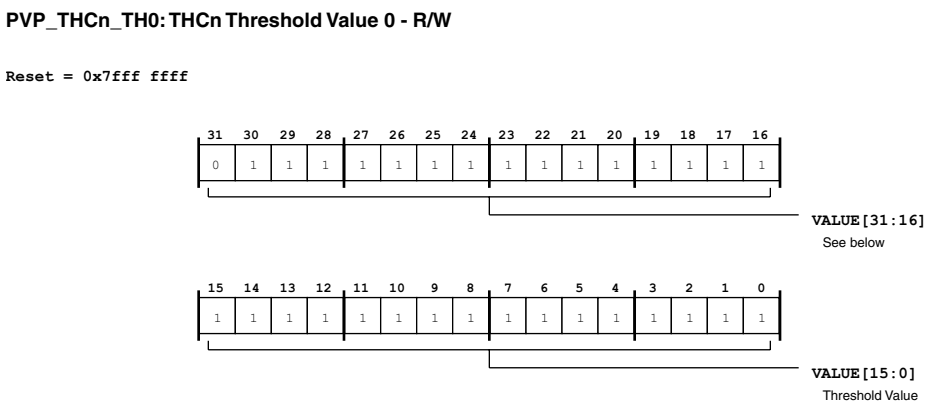


Figure 30-141: PVP_THCn_TH0 Register Diagram

Table 30-120: PVP_THCn_TH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Threshold Value.

THCn Threshold Value 1

The PVP_THCn_TH1 holds one of up to 16 threshold values, which the THC uses in quantization mode and hysteresis mode. For more information about these modes and the registers involved in their operation, see the PVP functional description.

PVP_THCn_TH1: THCn Threshold Value 1 - R/W

Reset = 0x7fff ffff

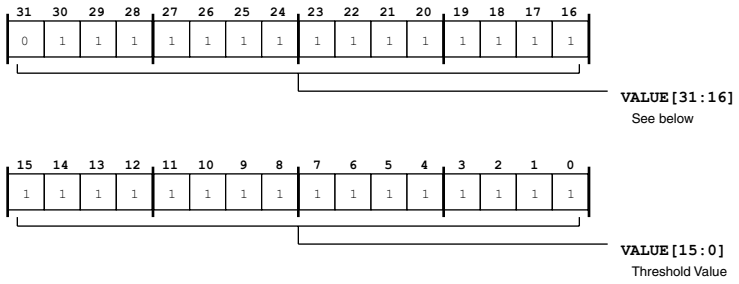


Figure 30-142: PVP_THCn_TH1 Register Diagram

Table 30-121: PVP_THCn_TH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Threshold Value.

THCn Threshold Value 2

The PVP_THCn_TH2 holds one of up to 16 threshold values, which the THC uses in quantization mode and hysteresis mode. For more information about these modes and the registers involved in their operation, see the PVP functional description.

PVP_THCn_TH2: THCn Threshold Value 2 - R/W

Reset = 0x7fff ffff

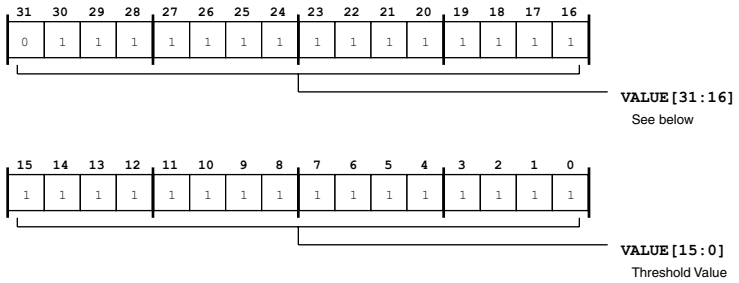


Figure 30-143: PVP_THCn_TH2 Register Diagram

Table 30-122: PVP_THCn_TH2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Threshold Value.

THCn Threshold Value 3

The PVP_THCn_TH3 holds one of up to 16 threshold values, which the THC uses in quantization mode and hysteresis mode. For more information about these modes and the registers involved in their operation, see the PVP functional description.

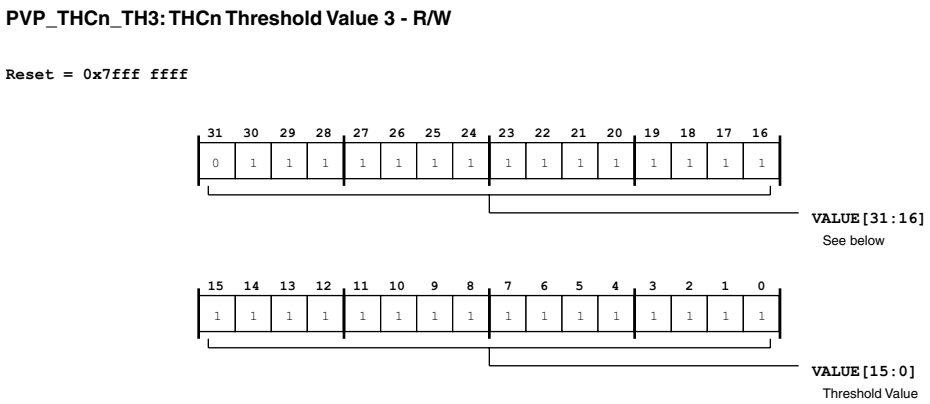


Figure 30-144: PVP_THCn_TH3 Register Diagram

Table 30-123: PVP_THCn_TH3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Threshold Value.

THCn Threshold Value 4

The PVP_THCn_TH4 holds one of up to 16 threshold values, which the THC uses in quantization mode and hysteresis mode. For more information about these modes and the registers involved in their operation, see the PVP functional description.

PVP_THCn_TH4: THCn Threshold Value 4 - R/W

Reset = 0x7fff ffff

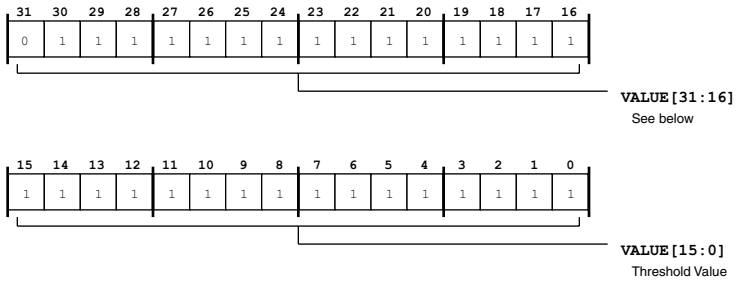


Figure 30-145: PVP_THCn_TH4 Register Diagram

Table 30-124: PVP_THCn_TH4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Threshold Value.

THCn Threshold Value 5

The PVP_THCn_TH5 holds one of up to 16 threshold values, which the THC uses in quantization mode and hysteresis mode. For more information about these modes and the registers involved in their operation, see the PVP functional description.

PVP_THCn_TH5: THCn Threshold Value 5 - R/W

Reset = 0x7fff ffff

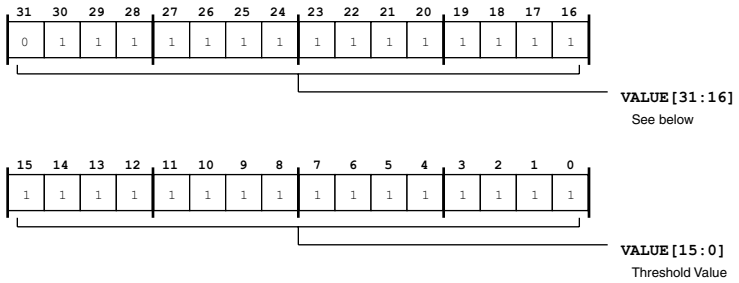


Figure 30-146: PVP_THCn_TH5 Register Diagram

Table 30-125: PVP_THCn_TH5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Threshold Value.

THCn Threshold Value 6

The PVP_THCn_TH6 holds one of up to 16 threshold values, which the THC uses in quantization mode and hysteresis mode. For more information about these modes and the registers involved in their operation, see the PVP functional description.

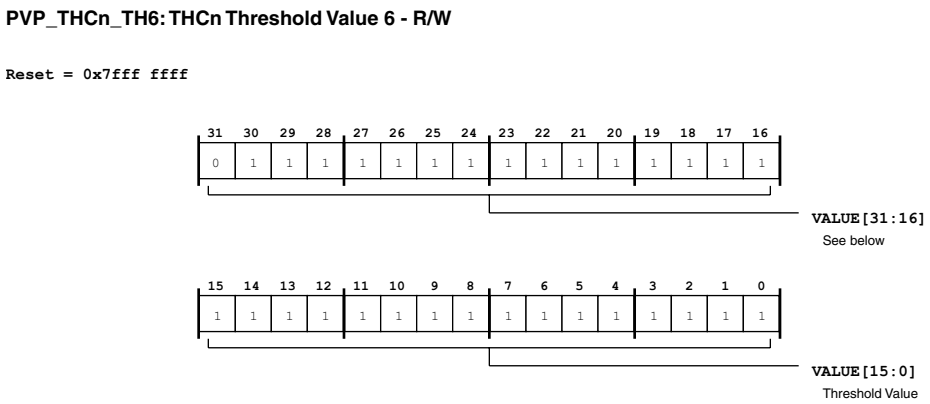


Figure 30-147: PVP_THCn_TH6 Register Diagram

Table 30-126: PVP_THCn_TH6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Threshold Value.

THCn Threshold Value 7

The PVP_THCn_TH7 holds one of up to 16 threshold values, which the THC uses in quantization mode and hysteresis mode. For more information about these modes and the registers involved in their operation, see the PVP functional description.

PVP_THCn_TH7: THCn Threshold Value 7 - R/W

Reset = 0x7fff ffff

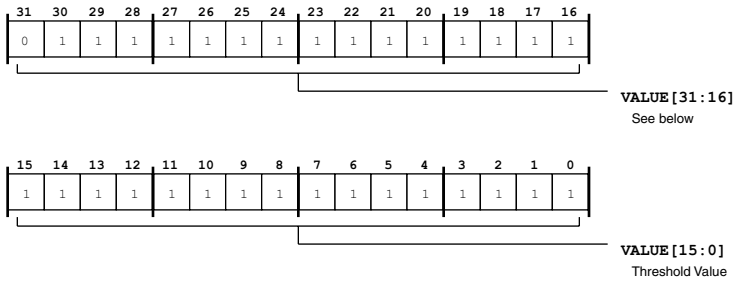


Figure 30-148: PVP_THCn_TH7 Register Diagram

Table 30-127: PVP_THCn_TH7 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Threshold Value.

THCn Threshold Value 8

The PVP_THCn_TH8 holds one of up to 16 threshold values, which the THC uses in quantization mode and hysteresis mode. For more information about these modes and the registers involved in their operation, see the PVP functional description.

PVP_THCn_TH8: THCn Threshold Value 8 - R/W

Reset = 0x7fff ffff

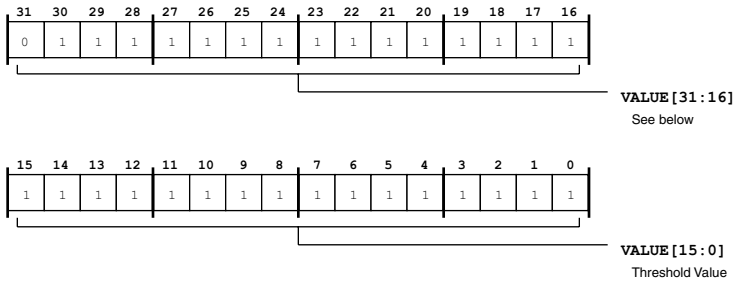


Figure 30-149: PVP_THCn_TH8 Register Diagram

Table 30-128: PVP_THCn_TH8 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Threshold Value.

THCn Threshold Value 9

The PVP_THCn_TH9 holds one of up to 16 threshold values, which the THC uses in quantization mode and hysteresis mode. For more information about these modes and the registers involved in their operation, see the PVP functional description.

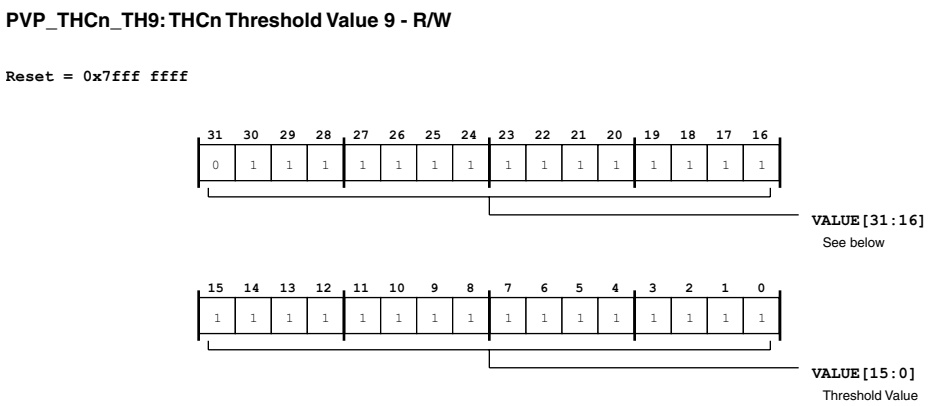


Figure 30-150: PVP_THCn_TH9 Register Diagram

Table 30-129: PVP_THCn_TH9 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Threshold Value.

THCn Threshold Value 10

The PVP_THCn_TH10 holds one of up to 16 threshold values, which the THC uses in quantization mode and hysteresis mode. For more information about these modes and the registers involved in their operation, see the PVP functional description.

PVP_THCn_TH10: THCn Threshold Value 10 - R/W

Reset = 0x7fff ffff

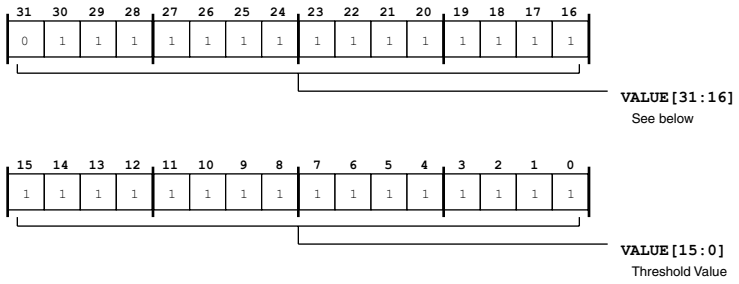


Figure 30-151: PVP_THCn_TH10 Register Diagram

Table 30-130: PVP_THCn_TH10 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Threshold Value.

THCn Threshold Value 11

The PVP_THCn_TH11 holds one of up to 16 threshold values, which the THC uses in quantization mode and hysteresis mode. For more information about these modes and the registers involved in their operation, see the PVP functional description.

PVP_THCn_TH11: THCn Threshold Value 11 - R/W

Reset = 0x7fff ffff

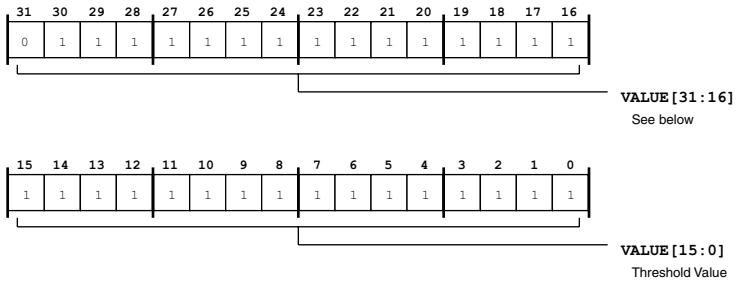


Figure 30-152: PVP_THCn_TH11 Register Diagram

Table 30-131: PVP_THCn_TH11 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Threshold Value.

THCn Threshold Value 12

The PVP_THCn_TH12 holds one of up to 16 threshold values, which the THC uses in quantization mode and hysteresis mode. For more information about these modes and the registers involved in their operation, see the PVP functional description.

PVP_THCn_TH12: THCn Threshold Value 12 - R/W

Reset = 0x7fff ffff

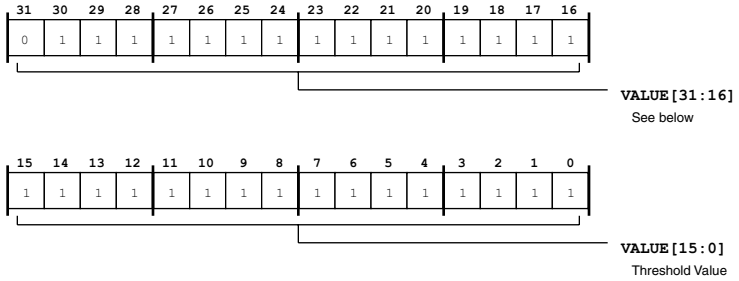


Figure 30-153: PVP_THCn_TH12 Register Diagram

Table 30-132: PVP_THCn_TH12 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Threshold Value.

THCn Threshold Value 13

The PVP_THCn_TH13 holds one of up to 16 threshold values, which the THC uses in quantization mode and hysteresis mode. For more information about these modes and the registers involved in their operation, see the PVP functional description.

PVP_THCn_TH13: THCn Threshold Value 13 - R/W

Reset = 0x7fff ffff

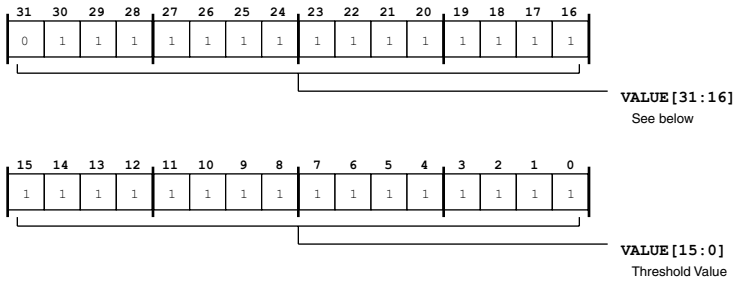


Figure 30-154: PVP_THCn_TH13 Register Diagram

Table 30-133: PVP_THCn_TH13 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Threshold Value.

THCn Threshold Value 14

The PVP_THCn_TH14 holds one of up to 16 threshold values, which the THC uses in quantization mode and hysteresis mode. For more information about these modes and the registers involved in their operation, see the PVP functional description.

PVP_THCn_TH14: THCn Threshold Value 14 - R/W

Reset = 0x7fff ffff

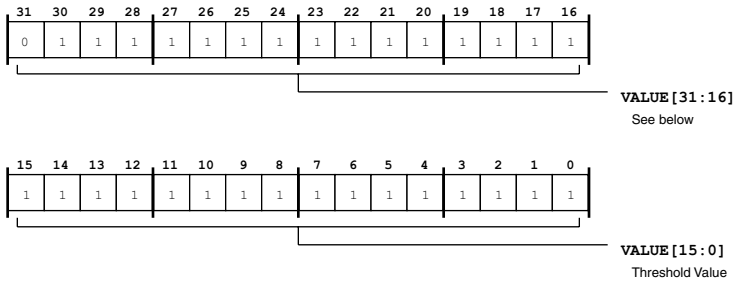


Figure 30-155: PVP_THCn_TH14 Register Diagram

Table 30-134: PVP_THCn_TH14 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Threshold Value.

THCn Threshold Value 15

The PVP_THCn_TH15 holds one of up to 16 threshold values, which the THC uses in quantization mode and hysteresis mode. For more information about these modes and the registers involved in their operation, see the PVP functional description.

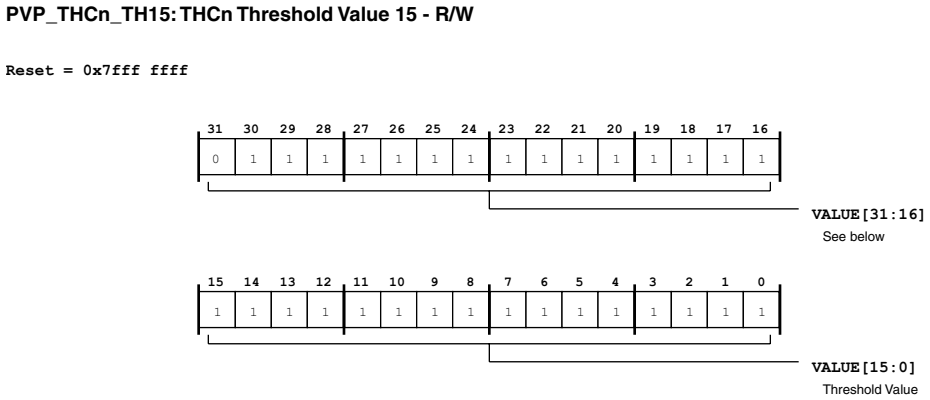


Figure 30-156: PVP_THCn_TH15 Register Diagram

Table 30-135: PVP_THCn_TH15 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Threshold Value.

THCn Histogram Horizontal Position

The PVP_THCn_HHPOS provides RLE encoder window information that the PVP uses for RLE compression operations.

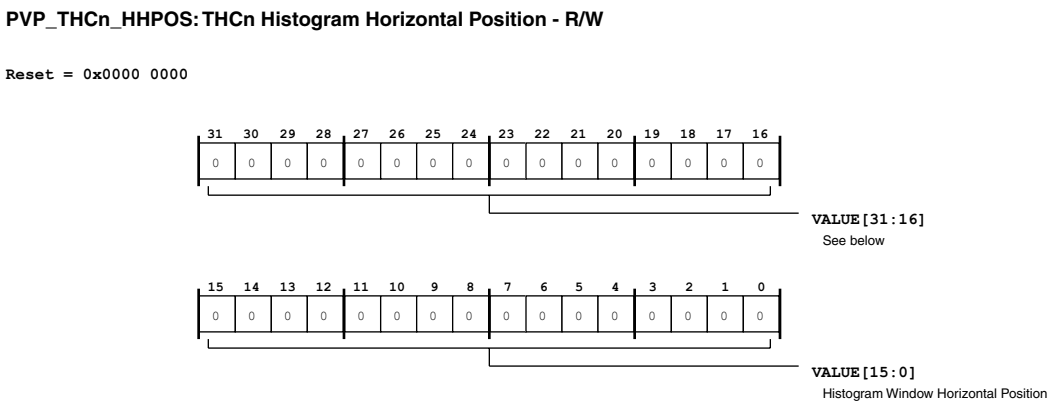


Figure 30-157: PVP_THCn_HHPOS Register Diagram

Table 30-136: PVP_THCn_HHPOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Histogram Window Horizontal Position. The PVP_THCn_HHPOS.VALUE provides the histogram window start X-coordinate position.

THCn Histogram Vertical Position

The PVP_THCn_HVPOS provides RLE encoder window information that the PVP uses for RLE compression operations.

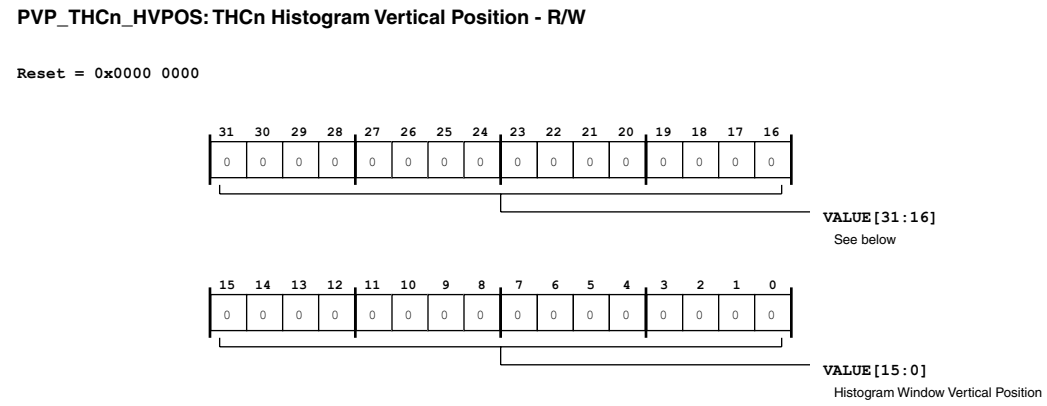


Figure 30-158: PVP_THCn_HVPOS Register Diagram

Table 30-137: PVP_THCn_HVPOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Histogram Window Vertical Position. The PVP_THCn_HVPOS.VALUE provides the histogram window start Y-coordinate position.

THCn Histogram Horizontal Count

The PVP_THCn_HHCNT provides RLE encoder window information that the PVP uses for RLE compression operations.

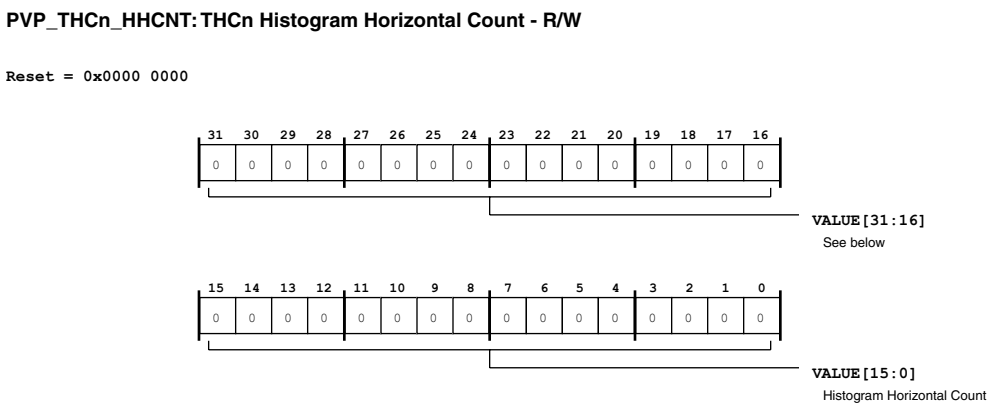


Figure 30-159: PVP_THCn_HHCNT Register Diagram

Table 30-138: PVP_THCn_HHCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Histogram Horizontal Count. The PVP_THCn_HHCNT.VALUE provides the histogram window width (count) in the X dimension.

THCn Histogram Vertical Count

The PVP_THCn_HVCNT provides RLE encoder window information that the PVP uses for RLE compression operations.

PVP_THCn_HVCNT: THCn Histogram Vertical Count - R/W

Reset = 0x0000 0000

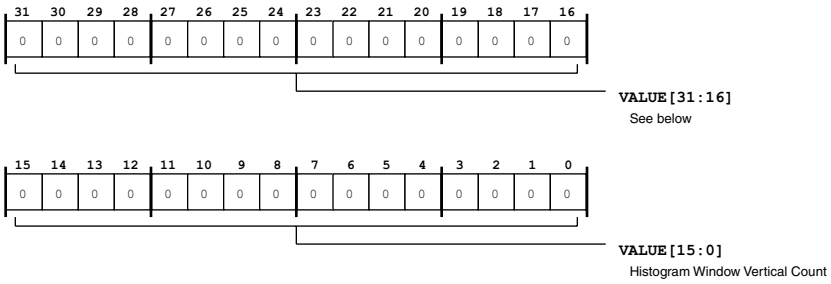


Figure 30-160: PVP_THCn_HVCNT Register Diagram

Table 30-139: PVP_THCn_HVCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Histogram Window Vertical Count. The PVP_THCn_HVCNT.VALUE provides the histogram window width in the Y dimension.

THCn RLE Horizontal Position

The PVP_THCn_RHPOS provides RLE encoder window information that the PVP uses for RLE compression operations.

PVP_THCn_RHPOS: THCn RLE Horizontal Position - R/W

Reset = 0x0000 0000

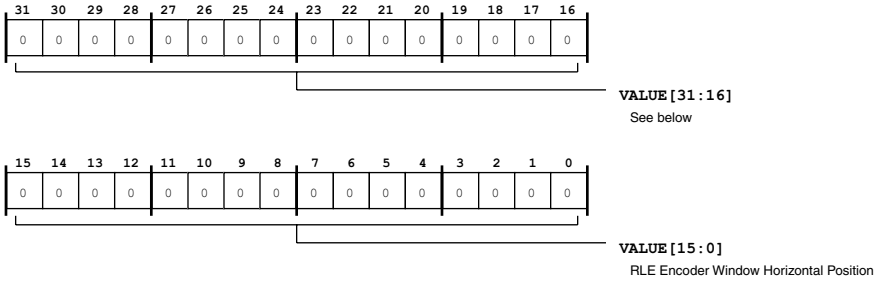


Figure 30-161: PVP_THCn_RHPOS Register Diagram

Table 30-140: PVP_THCn_RHPOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	RLE Encoder Window Horizontal Position. The PVP_THCn_RHPOS.VALUE provides the RLE encoder window start X-coordinate position.

THCn RLE Vertical Position

The PVP_THCn_RVPOS provides RLE encoder window information that the PVP uses for RLE compression operations.

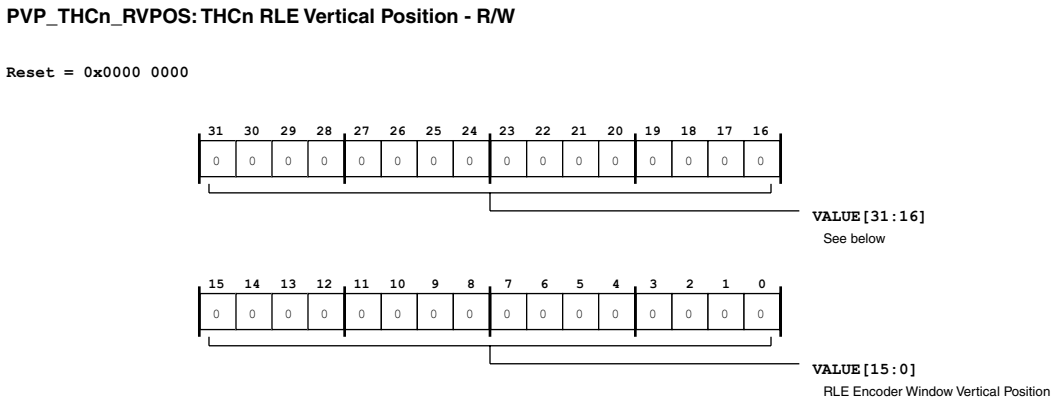


Figure 30-162: PVP_THCn_RVPOS Register Diagram

Table 30-141: PVP_THCn_RVPOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	RLE Encoder Window Vertical Position. The PVP_THCn_RVPOS.VALUE provides the RLE encoder window start Y-coordinate position.

THCn RLE Horizontal Count

The PVP_THCn_RHCNT provides RLE encoder window information that the PVP uses for RLE compression operations.

PVP_THCn_RHCNT: THCn RLE Horizontal Count - R/W

Reset = 0x0000 0000

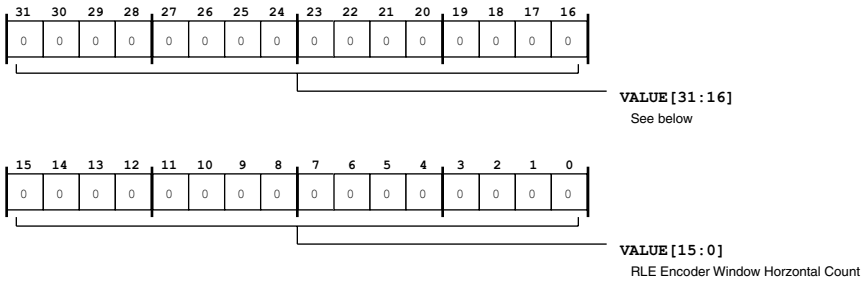


Figure 30-163: PVP_THCn_RHCNT Register Diagram

Table 30-142: PVP_THCn_RHCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	RLE Encoder Window Horzontal Count. The PVP_THCn_RHCNT.VALUE provides the RLE encoder window width (count) in the X dimension.

THCn RLE Vertical Count

The PVP_THCn_RVCNT provides RLE encoder window information that the PVP uses for RLE compression operations.

PVP_THCn_RVCNT: THCn RLE Vertical Count - R/W

Reset = 0x0000 0000

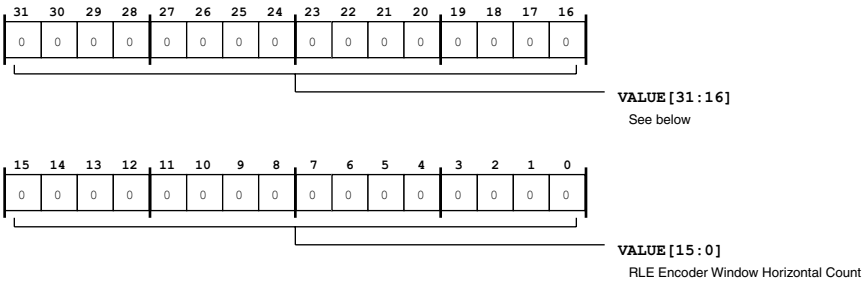


Figure 30-164: PVP_THCn_RVCNT Register Diagram

Table 30-143: PVP_THCn_RVCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	RLE Encoder Window Horizontal Count. The PVP_THCn_RVCNT.VALUE provides the RLE encoder window width (count) in the Y dimension.

THCn Histogram Frame Count Status

The PVP_THCn_HFCNT_STAT indicates the current histogram frame count. The number of frames for which the histogram counters operate is determined by the PVP_THCn_HFCNT register. On reaching this count, the histogram counters are cleared and restart again.

PVP_THCn_HFCNT_STAT: THCn Histogram Frame Count Status - R/NW

Reset = 0x0000 0000

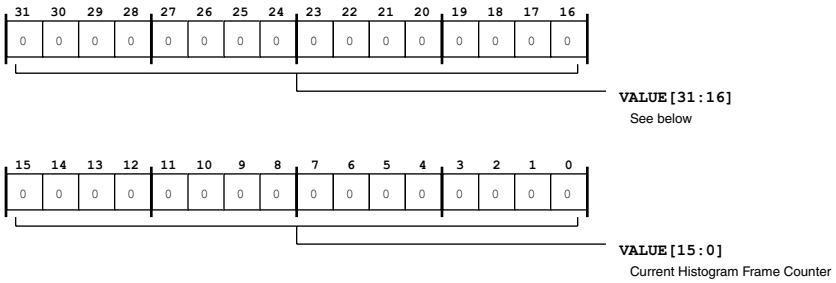


Figure 30-165: PVP_THCn_HFCNT_STAT Register Diagram

Table 30-144: PVP_THCn_HFCNT_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Current Histogram Frame Counter.

THCn Histogram Counter Value 0

The PVP_THCn_HCNT0_STAT holds the current value for one of the histogram counters. If a histogram counter reaches the maximum value (0xFFFFFFFF) before expiry of histogram frame count (PVP_THCn_HFCNT), the histogram counters do not overflow, but saturate at that value.

PVP_THCn_HCNT0_STAT: THCn Histogram Counter Value 0 - R/NW

Reset = 0x0000 0000

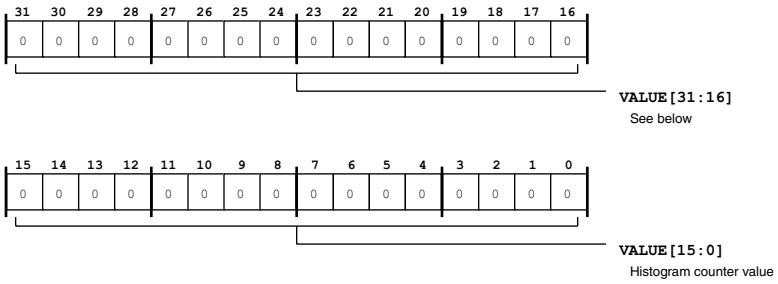


Figure 30-166: PVP_THCn_HCNT0_STAT Register Diagram

Table 30-145: PVP_THCn_HCNT0_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Histogram counter value.

THCn Histogram Counter Value 1

The PVP_THCn_HCNT1_STAT holds the current value for one of the histogram counters. If a histogram counter reaches the maximum value (0xFFFFFFFF) before expiry of histogram frame count (PVP_THCn_HFCNT), the histogram counters do not overflow, but saturate at that value.

PVP_THCn_HCNT1_STAT: THCn Histogram Counter Value 1 - R/NW

Reset = 0x0000 0000

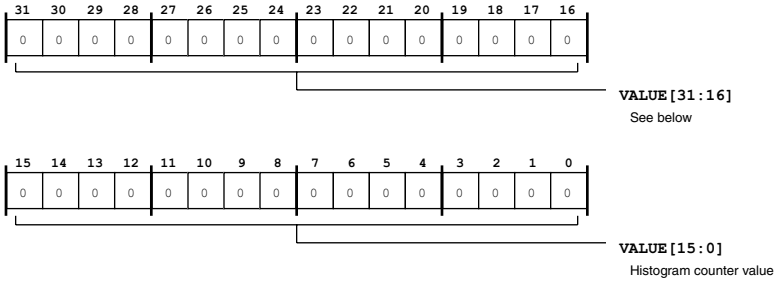


Figure 30-167: PVP_THCn_HCNT1_STAT Register Diagram

Table 30-146: PVP_THCn_HCNT1_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Histogram counter value.

THCn Histogram Counter Value 2

The PVP_THCn_HCNT2_STAT holds the current value for one of the histogram counters. If a histogram counter reaches the maximum value (0xFFFFFFFF) before expiry of histogram frame count (PVP_THCn_HFCNT), the histogram counters do not overflow, but saturate at that value.

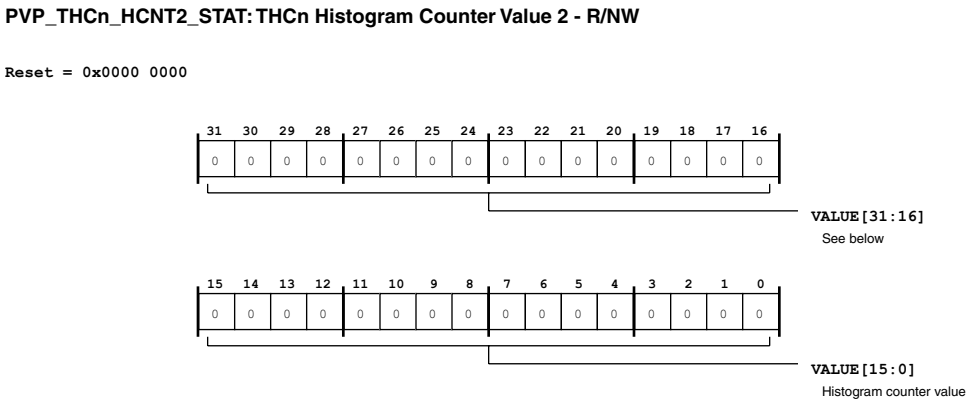


Figure 30-168: PVP_THCn_HCNT2_STAT Register Diagram

Table 30-147: PVP_THCn_HCNT2_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Histogram counter value.

THCn Histogram Counter Value 3

The PVP_THCn_HCNT3_STAT holds the current value for one of the histogram counters. If a histogram counter reaches the maximum value (0xFFFFFFFF) before expiry of histogram frame count (PVP_THCn_HFCNT), the histogram counters do not overflow, but saturate at that value.

PVP_THCn_HCNT3_STAT: THCn Histogram Counter Value 3 - R/NW

Reset = 0x0000 0000

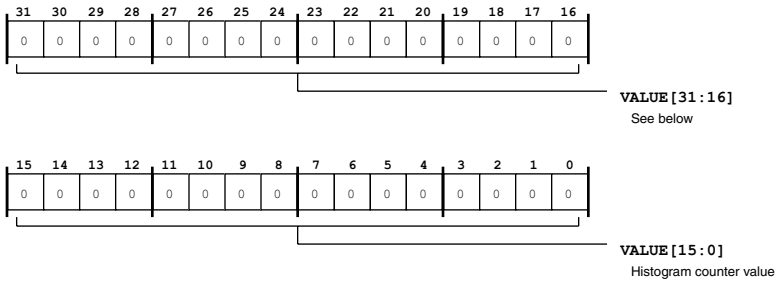


Figure 30-169: PVP_THCn_HCNT3_STAT Register Diagram

Table 30-148: PVP_THCn_HCNT3_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Histogram counter value.

THCn Histogram Counter Value 4

The PVP_THCn_HCNT4_STAT holds the current value for one of the histogram counters. If a histogram counter reaches the maximum value (0xFFFFFFFF) before expiry of histogram frame count (PVP_THCn_HFCNT), the histogram counters do not overflow, but saturate at that value.

PVP_THCn_HCNT4_STAT: THCn Histogram Counter Value 4 - R/NW

Reset = 0x0000 0000

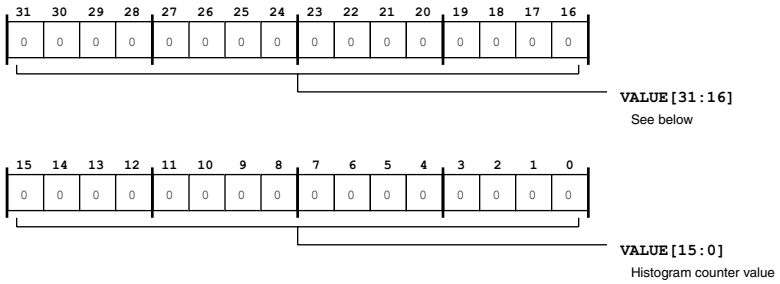


Figure 30-170: PVP_THCn_HCNT4_STAT Register Diagram

Table 30-149: PVP_THCn_HCNT4_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Histogram counter value.

THCn Histogram Counter Value 5

The PVP_THCn_HCNT5_STAT holds the current value for one of the histogram counters. If a histogram counter reaches the maximum value (0xFFFFFFFF) before expiry of histogram frame count (PVP_THCn_HFCNT), the histogram counters do not overflow, but saturate at that value.

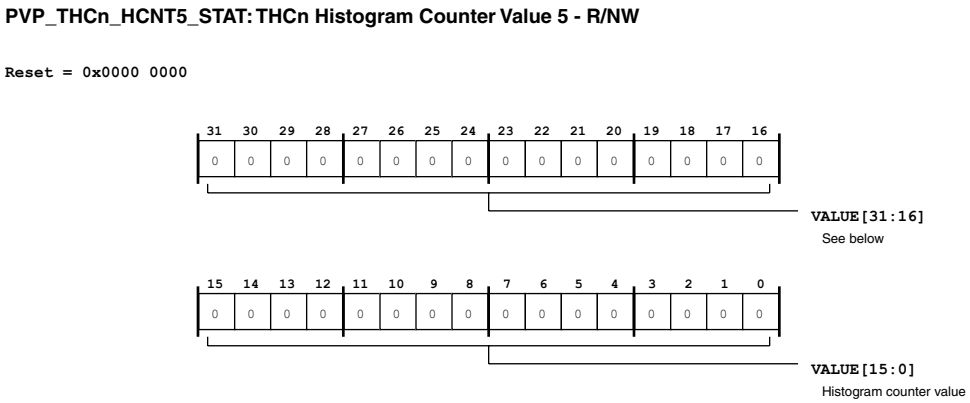


Figure 30-171: PVP_THCn_HCNT5_STAT Register Diagram

Table 30-150: PVP_THCn_HCNT5_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Histogram counter value.

THCn Histogram Counter Value 6

The PVP_THCn_HCNT6_STAT holds the current value for one of the histogram counters. If a histogram counter reaches the maximum value (0xFFFFFFFF) before expiry of histogram frame count (PVP_THCn_HFCNT), the histogram counters do not overflow, but saturate at that value.

PVP_THCn_HCNT6_STAT: THCn Histogram Counter Value 6 - R/NW

Reset = 0x0000 0000

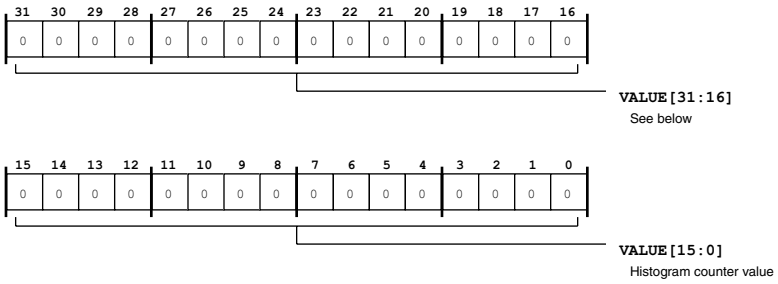


Figure 30-172: PVP_THCn_HCNT6_STAT Register Diagram

Table 30-151: PVP_THCn_HCNT6_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Histogram counter value.

THCn Histogram Counter Value 7

The PVP_THCn_HCNT7_STAT holds the current value for one of the histogram counters. If a histogram counter reaches the maximum value (0xFFFFFFFF) before expiry of histogram frame count (PVP_THCn_HFCNT), the histogram counters do not overflow, but saturate at that value.

PVP_THCn_HCNT7_STAT: THCn Histogram Counter Value 7 - R/NW

Reset = 0x0000 0000

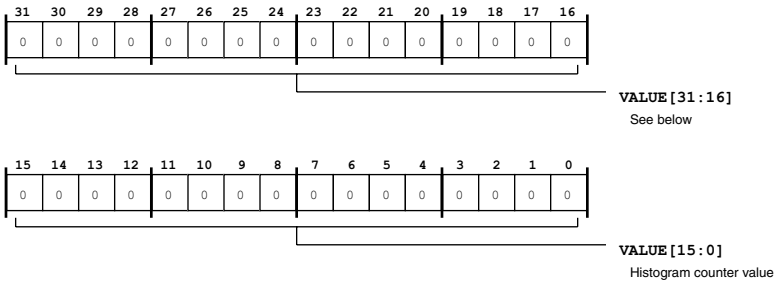


Figure 30-173: PVP_THCn_HCNT7_STAT Register Diagram

Table 30-152: PVP_THCn_HCNT7_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Histogram counter value.

THCn Histogram Counter Value 8

The PVP_THCn_HCNT8_STAT holds the current value for one of the histogram counters. If a histogram counter reaches the maximum value (0xFFFFFFFF) before expiry of histogram frame count (PVP_THCn_HFCNT), the histogram counters do not overflow, but saturate at that value.

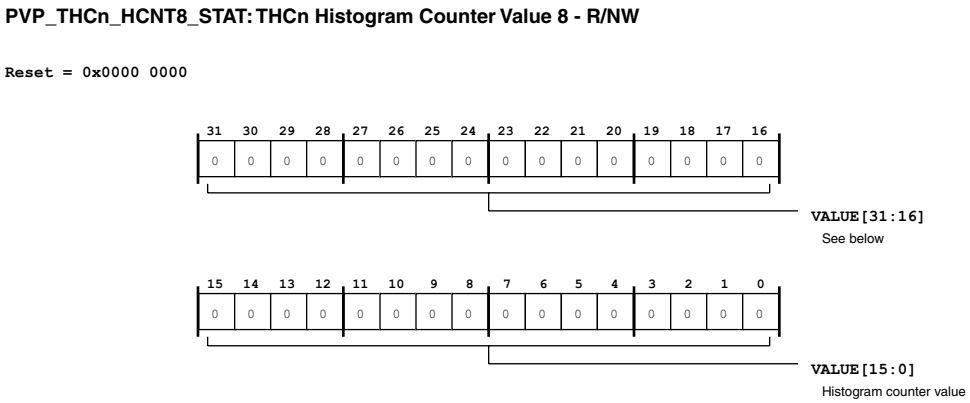


Figure 30-174: PVP_THCn_HCNT8_STAT Register Diagram

Table 30-153: PVP_THCn_HCNT8_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Histogram counter value.

THCn Histogram Counter Value 9

The PVP_THCn_HCNT9_STAT holds the current value for one of the histogram counters. If a histogram counter reaches the maximum value (0xFFFFFFFF) before expiry of histogram frame count (PVP_THCn_HFCNT), the histogram counters do not overflow, but saturate at that value.

PVP_THCn_HCNT9_STAT: THCn Histogram Counter Value 9 - R/NW

Reset = 0x0000 0000

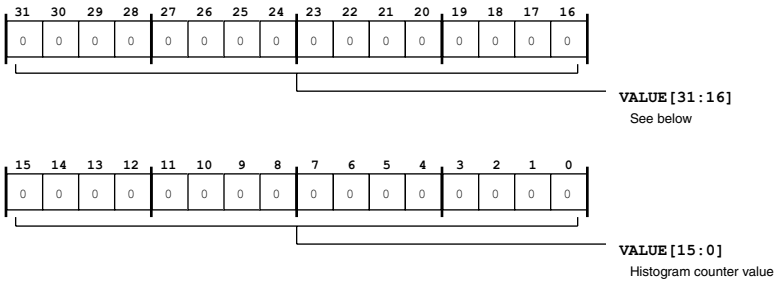


Figure 30-175: PVP_THCn_HCNT9_STAT Register Diagram

Table 30-154: PVP_THCn_HCNT9_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Histogram counter value.

THCn Histogram Counter Value 10

The PVP_THCn_HCNT10_STAT holds the current value for one of the histogram counters. If a histogram counter reaches the maximum value (0xFFFFFFFF) before expiry of histogram frame count (PVP_THCn_HFCNT), the histogram counters do not overflow, but saturate at that value.

PVP_THCn_HCNT10_STAT: THCn Histogram Counter Value 10 - R/NW

Reset = 0x0000 0000

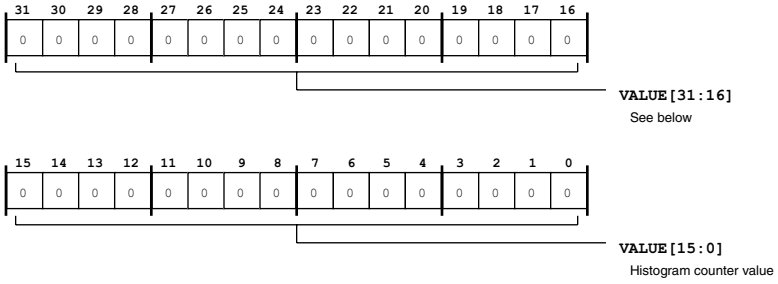


Figure 30-176: PVP_THCn_HCNT10_STAT Register Diagram

Table 30-155: PVP_THCn_HCNT10_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Histogram counter value.

THCn Histogram Counter Value 11

The PVP_THCn_HCNT11_STAT holds the current value for one of the histogram counters. If a histogram counter reaches the maximum value (0xFFFFFFFF) before expiry of histogram frame count (PVP_THCn_HFCNT), the histogram counters do not overflow, but saturate at that value.

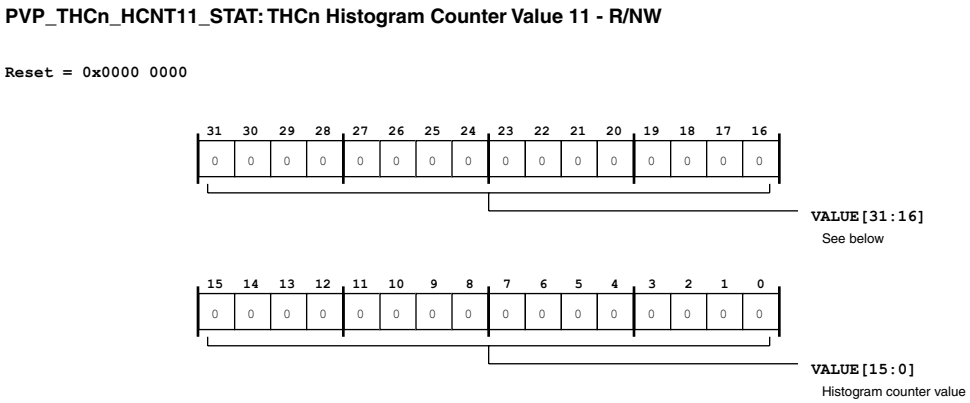


Figure 30-177: PVP_THCn_HCNT11_STAT Register Diagram

Table 30-156: PVP_THCn_HCNT11_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Histogram counter value.

THCn Histogram Counter Value 12

The PVP_THCn_HCNT12_STAT holds the current value for one of the histogram counters. If a histogram counter reaches the maximum value (0xFFFFFFFF) before expiry of histogram frame count (PVP_THCn_HFCNT), the histogram counters do not overflow, but saturate at that value.

PVP_THCn_HCNT12_STAT: THCn Histogram Counter Value 12 - R/NW

Reset = 0x0000 0000

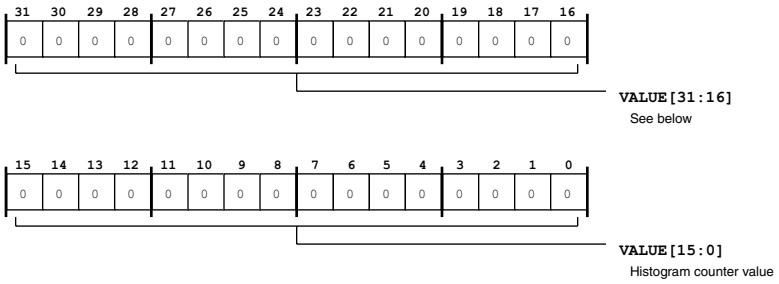


Figure 30-178: PVP_THCn_HCNT12_STAT Register Diagram

Table 30-157: PVP_THCn_HCNT12_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Histogram counter value.

THCn Histogram Counter Value 13

The PVP_THCn_HCNT13_STAT holds the current value for one of the histogram counters. If a histogram counter reaches the maximum value (0xFFFFFFFF) before expiry of histogram frame count (PVP_THCn_HFCNT), the histogram counters do not overflow, but saturate at that value.

PVP_THCn_HCNT13_STAT: THCn Histogram Counter Value 13 - R/NW

Reset = 0x0000 0000

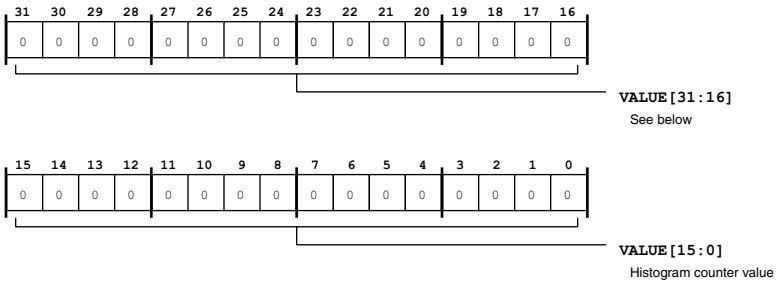


Figure 30-179: PVP_THCn_HCNT13_STAT Register Diagram

Table 30-158: PVP_THCn_HCNT13_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Histogram counter value.

THCn Histogram Counter Value 14

The PVP_THCn_HCNT14_STAT holds the current value for one of the histogram counters. If a histogram counter reaches the maximum value (0xFFFFFFFF) before expiry of histogram frame count (PVP_THCn_HFCNT), the histogram counters do not overflow, but saturate at that value.

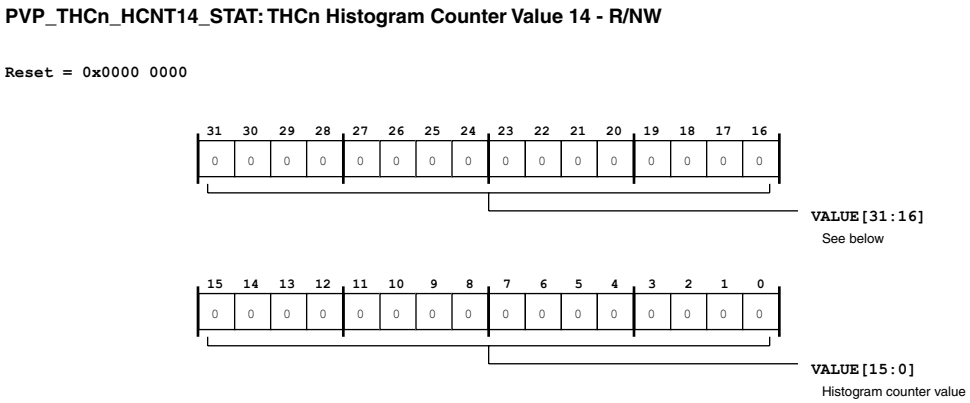


Figure 30-180: PVP_THCn_HCNT14_STAT Register Diagram

Table 30-159: PVP_THCn_HCNT14_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Histogram counter value.

THCn Histogram Counter Value 15

The PVP_THCn_HCNT15_STAT holds the current value for one of the histogram counters. If a histogram counter reaches the maximum value (0xFFFFFFFF) before expiry of histogram frame count (PVP_THCn_HFCNT), the histogram counters do not overflow, but saturate at that value.

PVP_THCn_HCNT15_STAT: THCn Histogram Counter Value 15 - R/NW

Reset = 0x0000 0000

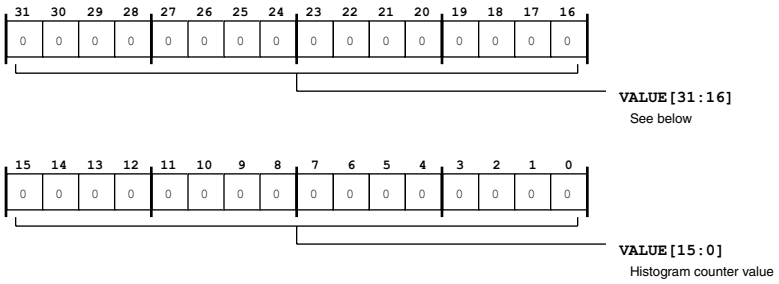


Figure 30-181: PVP_THCn_HCNT15_STAT Register Diagram

Table 30-160: PVP_THCn_HCNT15_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Histogram counter value.

THCn Number of RLE Reports

The PVP_THCn_RREP_STAT holds the number of RLE reports. The PVP generates the run length reports at the output of the module as soon as they are available. Note that there is a variable number of reports generated per line or per frame, depending on the pixel patterns.

PVP_THCn_RREP_STAT: THCn Number of RLE Reports - R/NW

Reset = 0x0000 0000

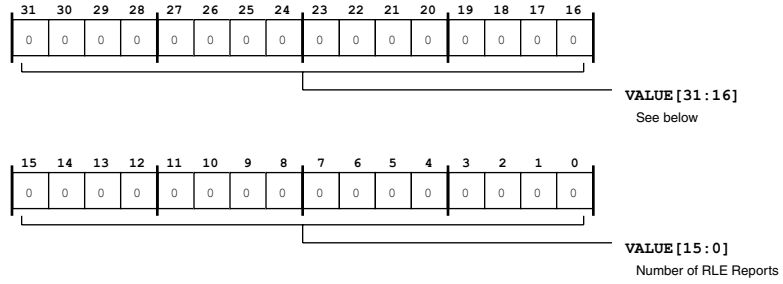


Figure 30-182: PVP_THCn_RREP_STAT Register Diagram

Table 30-161: PVP_THCn_RREP_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Number of RLE Reports.

PMA Configuration

The PVP_PMA_CFG register controls the pixel magnitude and angle computation unit (PMA) block's pipeline features, including port selection and block enable.

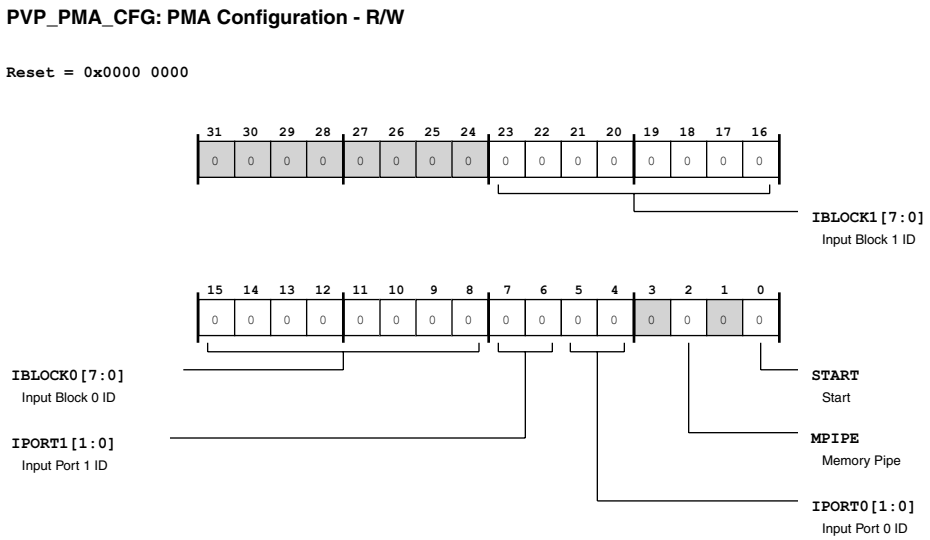


Figure 30-183: PVP_PMA_CFG Register Diagram

Table 30-162: PVP_PMA_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:16 (R/W)	IBLOCK1	Input Block 1 ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_PMA_CFG.IBLOCK1 and PVP_PMA_CFG.IPORT1 fields determine which output port(s) are connected to the input port(s).	
		12	IPF0
		14	IPF1
		16	CNV0
		20	CNV1
		24	CNV2
		28	CNV3
15:8 (R/W)	IBLOCK0	Input Block 0 ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_PMA_CFG.IBLOCK0 and PVP_PMA_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s).	
		12	IPF0
		14	IPF1
		16	CNV0
		20	CNV1
		24	CNV2
		28	CNV3
7:6 (R/W)	IPORT1	Input Port 1 ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_PMA_CFG.IBLOCK1 and PVP_PMA_CFG.IPORT1 fields determine which output port(s) are connected to the input port(s). For more information, see the connectivity tables for each input block in the architectural concepts section.	
5:4 (R/W)	IPORT0	Input Port 0 ID. The blocks comprising the PVP pipe may connect to one or more output ports. The PVP_PMA_CFG.IBLOCK0 and PVP_PMA_CFG.IPORT0 fields determine which output port(s) are connected to the input port(s). For more information, see the connectivity tables for each input block in the architectural concepts section.	

Table 30-162: PVP_PMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	MPIPE	Memory Pipe. The PVP_PMA_CFG.MPIPE bit selects memory pipe or camera pipe.	
		0	Camera Pipe
		1	Memory Pipe
0 (R/W)	START	Start. The PVP_PMA_CFG.START enables the PVP. This bit reads 0 after the pipe is drained.	
		1	Write to Enable the Module

31 Enhanced Parallel Peripheral Interface (EPPI)

The enhanced parallel peripheral interface (EPPI) is a half-duplex, bidirectional port with a dedicated clock pin and three frame sync (FS) pins. It can support direct connections to active TFT LCD, parallel A/D and D/A converters, video encoders and decoders, image sensor modules and other general-purpose peripherals. Each EPPI has two DMA channels associated with it. Moreover, in some modes, a EPPI may use an additional DMA channel.

EPPI Features

The following features are supported in the EPPI module.

- Programmable data length: 8, 10, 12, 14, 16, 18 and 24 bits per clock cycle
- Bidirectional and half-duplex port
- Internal or external clock source
- Clock gating by an external device asserting the clock gating control signal
- Various framed and non-framed operating modes as well as internal or external frame syncs
- Various general-purpose modes with 1, 2, 3 and 0 frame sync modes for both receive and transmit
- Ignores premature external frame syncs for data consistency
- SMPTE274M and SMPTE 296M high definition format support
- ITU-656, SMPTE 296M and SMPTE 274M status word error detection and correction for ITU-656 receive modes
- ITU-656, SMPTE 296M and SMPTE 274M receive modes – active video only, vertical blanking only, and entire field
- ITU-656, SMPTE 296M and SMPTE 274M preamble and status word decode
- Optional packing and unpacking of data to/from 32 bits from/to 8, 16 and 24 bits. If packing/unpacking is enabled, endianness can be altered to change the order of packing/unpacking of bytes/words
- Optional sign extension or zero-fill and alternate even or odd data sample filter for receive modes
- RGB888 to RGB666 or RGB565 conversion for transmit modes
- 4:2:2 YCrCb data Tx/Rx interleaving/de-interleaving modes

- Configurable LCD data enable (DEN) output available on frame sync 3
- Supports delayed start of PPI frame syncs
- Data clipping and mirroring
- Horizontal and vertical windowing for general-purpose 2 and 3 frame sync modes
- Preamble, blanking and stripping support
- Multiplexing dual input

EPPI Functional Description

The following sections provide a functional descriptions of the EPPI.

- *RGB Data Formats*
- *Data Clipping*
- *Data Mirroring*
- *Windowing*
- *Preamble, Blanking and Stripping Support*

ADSP-BF60x EPPI Register List

The EPPI is a half-duplex, bidirectional port accommodating up to 16 bits of data. It has a dedicated clock pin and three multiplexed frame sync pins. The highest system throughput is achieved with 8-bit data, since two 8-bit data samples can be packed as a single 16-bit word. In such a case, the earlier sample is placed in the 8 least significant bits (LSBs). For more information on EPPI functionality, see the EPPI register descriptions.

Table 31-1: ADSP-BF60x EPPI Register List

Name	Description
EPPI_STAT	Status Register
EPPI_HCNT	Horizontal Transfer Count Register
EPPI_HDLY	Horizontal Delay Count Register
EPPI_VCNT	Vertical Transfer Count Register
EPPI_VDLY	Vertical Delay Count Register

Table 31-1: ADSP-BF60x EPPI Register List (Continued)

Name	Description
EPPI_FRAME	Lines Per Frame Register
EPPI_LINE	Samples Per Line Register
EPPI_CLKDIV	Clock Divide Register
EPPI_CTL	Control Register
EPPI_FS1_WLHB	FS1 Width Register / EPPI Horizontal Blanking Samples Per Line Register
EPPI_FS1_PASPL	FS1 Period Register / EPPI Active Samples Per Line Register
EPPI_FS2_WLVB	FS2 Width Register / EPPI Lines Of Vertical Blanking Register
EPPI_FS2_PALPF	FS2 Period Register / EPPI Active Lines Per Field Register
EPPI_IMSK	Interrupt Mask Register
EPPI_ODDCLIP	Clipping Register for ODD (Chroma) Data
EPPI_EVENCLIP	Clipping Register for EVEN (Luma) Data
EPPI_FS1_DLY	Frame Sync 1 Delay Value
EPPI_FS2_DLY	Frame Sync 2 Delay Value
EPPI_CTL2	Control Register 2

ADSP-BF60x EPPI Interrupt List

Table 31-2: ADSP-BF60x EPPI Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
EPPI0 Channel 0 DMA	98	29	LEVEL
EPPI0 Channel 1 DMA	99	30	LEVEL
EPPI0 Status	100		LEVEL
EPPI2 Channel 0 DMA	101	31	LEVEL
EPPI2 Channel 1 DMA	102	32	LEVEL
EPPI2 Status	103		LEVEL

Table 31-2: ADSP-BF60x EPPI Interrupt List Interrupt List (Continued)

Description	Interrupt ID	DMA Channel	Sensitivity
EPPI1 Channel 0 DMA	104	33	LEVEL
EPPI1 Channel 1 DMA	105	34	LEVEL
EPPI1 Status	106		LEVEL

ADSP-BF60x EPPI Trigger List

Table 31-3: ADSP-BF60x EPPI Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
EPPI0 Channel 0 DMA	51	PULSE/EDGE
EPPI0 Channel 1 DMA	52	PULSE/EDGE
EPPI2 Channel 0 DMA	53	PULSE/EDGE
EPPI2 Channel 1 DMA	54	PULSE/EDGE
EPPI1 Channel 0 DMA	55	PULSE/EDGE
EPPI1 Channel 1 DMA	56	PULSE/EDGE

Table 31-4: ADSP-BF60x EPPI Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
EPPI0 Channel 0 DMA	51	
EPPI0 Channel 1 DMA	52	
EPPI2 Channel 0 DMA	53	
EPPI2 Channel 1 DMA	54	
EPPI1 Channel 0 DMA	55	
EPPI1 Channel 1 DMA	56	

ADSP-BF60x EPPI DMA List

Table 31-5: ADSP-BF60x EPPI DMA List DMA Channel List

Description	DMA Channel
EPPI0 Channel 0 DMA	DMA29
EPPI0 Channel 1 DMA	DMA30

Table 31-5: ADSP-BF60x EPPI DMA List DMA Channel List (Continued)

Description	DMA Channel
EPPI2 Channel 0 DMA	DMA31
EPPI2 Channel 1 DMA	DMA32
EPPI1 Channel 0 DMA	DMA33
EPPI1 Channel 1 DMA	DMA34

RGB Data Formats

For transmit modes, the EPPI can convert RGB888 data in memory to RGB666 at the output if the EPPI_CTL.RGBFMTEN bit is set and if the EPPI_CTL.DLENvalue is equal to 18 bits. Similarly, the EPPI can convert RGB888 data in memory to RGB565 at the output if the EPPI_CTL.RGBFMTEN bit is set and if EPPI_CTL.DLENis equal to 16 bits.

This conversion is performed as follows:

- If EPPI_CTL.PACKEN=1, the EPPI first unpacks, according to the EPPI_CTL.SWAPEN bit settings, the three 32-bit data words from the DMA into four 24-bit data words to be transmitted out as described earlier.
- If EPPI_CTL.PACKEN=0, then the EPPI takes the lower 24 bits of the 32-bit DMA as the data to be transmitted. Then the EPPI truncates this 24-bit data word to the required data width by removing the lower 2 bits of G and the lower 2 or 3 bits of R and B.

Data Clipping

The EPPI contains two registers to define the lower and upper limits for the Luma and Chroma components. This is used for clipping data values during 8, 10, 12 or 16-bit transmit modes. All data values for odd samples which are less than the value in the EPPI_ODDCLIP.LOWODD bit field are replaced with the value in the EPPI_ODDCLIP.LOWODD field and all data values for even samples which are less than the value in the EPPI_EVENCLIP.LOWEVEN field are replaced with the value in the EPPI_EVENCLIP.LOWEVEN field.

In the same manner, all data values for odd samples which are more than the value in the EPPI_ODDCLIP.HIGHODD bit field are replaced with the value in the EPPI_ODDCLIP.HIGHODD field and all data values for even samples which are more than the values in the EPPI_EVENCLIP.HIGHEVEN field are replaced with the values in the EPPI_EVENCLIP.HIGHEVEN field.

Depending on the programmed EPPI length, only the corresponding bits (least aligned) are considered for clipping. For example if the EPPI is programmed in 10-bit mode, bits 9–0 and bits 25–16 constitute the clipping thresholds. The higher bits are ignored. The EPPI supports 8, 10, 12, and 16-bit clipping thresholds.

For the 4:2:2 YCrCb color space, Luma and Chroma typically have different lower and upper thresholds, which is why separate thresholds may be required for even and odd data samples. In the case of mono-

chrome (Y only) or some non-video clipping applications, the value in the EPPI_ODDCLIP.LOWODD field should be the same as the value in the EPPI_EVENCLIP.LOWEVEN field, and the value in the EPPI_ODDCLIP.HIGHODD field should be the same as the value in the EPPI_EVENCLIP.HIGHEVEN field.

In GP 0 FS mode with internal blanking generation, clipping is valid only for the active video part of the transmitted data. ITU-R 656 preambles, status words and blanking data bypass the clipping logic.

If the EPPI is programmed in 16, 20 or 24-bit mode with the EPPI_CTL.SPLTWRD bit set, the YDATA (luma data) gets the clipping threshold levels of the EPPI_EVENCLIP register and the CDATA (chroma data) gets the clipping threshold levels of the EPPI_ODDCLIP register.

The clipping registers are ignored when the EPPI_CTL.RGBFMTEN bit is set.

Data Mirroring

To increase the pin multiplexing options with respect to the EPPI data pins, an additional data mirroring feature is available which mirrors the EPPI data bits 15–0. The MMR bit (EPPI_CTL.DMIRR) if set, mirrors the bits (default is EPPI_CTL.DMIRR=0, no mirror). The complete description of this bit is provided in "Register Descriptions" section. This feature is available in both transmit and receive.

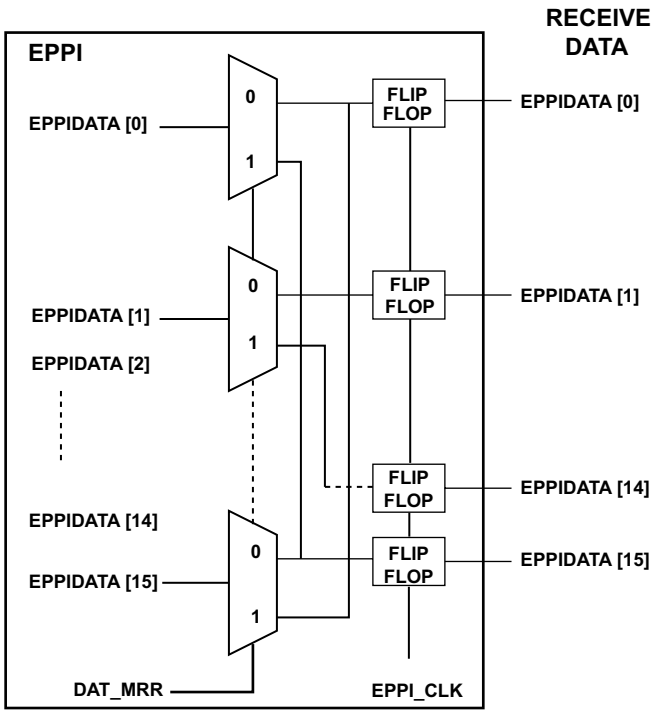


Figure 31-1: Data Mirroring Receive

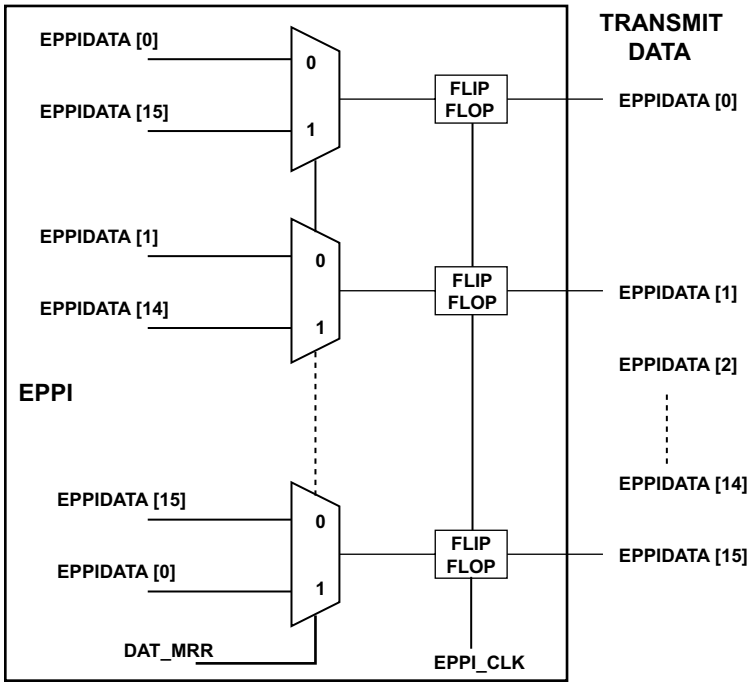


Figure 31-2: Data Mirroring Transmit

Windowing

The EPPI supports windowing for general-purpose input modes. The module can be configured to bring in a region of interest instead of the entire frame of data which helps reduce bandwidth.

Preamble, Blanking and Stripping Support

The EPPI supports embedding blanking information and clipping of active data to be transmitted. This is available for single channel data, interleaved data and parallel data and supports data length (`EPPI_CTL.DLEN`) equal to 16, 20 or 24 bits.

Support of preamble generation/detection and stripping of blanking information is also provided for ITU656 as well as the two HD formats, SMPTE 274M and 296M. The SMPTE standards are shown below in the table as a comparison with the ITU 656 modes. Preambles for SMPTE and ITU modes are identical. Extension of preamble to 12 bits is also supported.

Table 31-6: Video Mode Comparison

Video Mode	Frame Rate	Frame Resolution	Active Video Resolution	Sampling Frequency (MHz)	Remarks
ITU 656 (NTSC)	30	1716x525	720x480	27.2	Y-C interleaved

Table 31-6: Video Mode Comparison (Continued)

Video Mode	Frame Rate	Frame Resolution	Active Video Resolution	Sampling Frequency (MHz)	Remarks
ITU 656 (PAL)	25	1728x625	720x576	26.8	Y-C interleaved
SMPTE 296M	30	3300x750	1280x720	74.25	Y,C separate
	60	1650x750	1280x720	74.25	Y,C separate
SMPTE 274M	30	2200x1125	1920x1080	74.25	Y,C separate
	60	2200x1125	1920x1080	148.5	Y,C separate
	25	2640x1125	1920x1080	74.25	Y,C separate
	50	2640x1125	1920x1080	148.5	Y,C separate
	24	2750x1125	1920x1080	74.25	Y,C separate

The EPPI supports all SMPTE modes shown above which run at a sampling frequency not higher than 74. 25 MHz.

EPPI Definitions

ITU-R BT. 656

Description of a digital video protocol for interfaces and data stream format required to send uncompressed PAL or NTSC standard definition TV (525 or 625 lines) signals.

YUV422

YUV is a color space where pixels are defined by a luminance (Y) component and chrominance (UV) components. The suffix signifies how the chrominance components have been decimated as well as formatting. In this case, the YUV422 format has the chrominance decimated by two, meaning only half of each chrominance component are available. Typical YUV422 formatting interleaves the luminance and chrominance as such as U1Y1V1Y2U2Y3V2Y4.

RGB888

RGB is a color space where pixels are defined by three color values; one red (R), one green (G) and one blue (B). The suffix signifies the bit widths for these color components. In this case, RGB888 means that each red, green and blue value is 8-bits each.

RGB565

RGB is a color space where pixels are defined by three color values; one red (R), one green (G) and one blue (B). The suffix signifies the bit widths for these color components. In this case, RGB565 means that

the red (R) and blue (B) are 5-bits each while the green (G) is 6-bits. When packed together, each RGB565 pixel can be represented in a 16-bit data word. This format is commonly used in LCD display panels.

SMPTE 274M

An HD standard defining the spatial resolution (image sample structure) and frame rates for 1920x1080.

SMPTE 296M

An HD standard for defining the spatial resolution (image sample structure) and frame rates fro 1280x720.

EPPI Block Diagram

The figure shows the functional blocks within the EPPI.

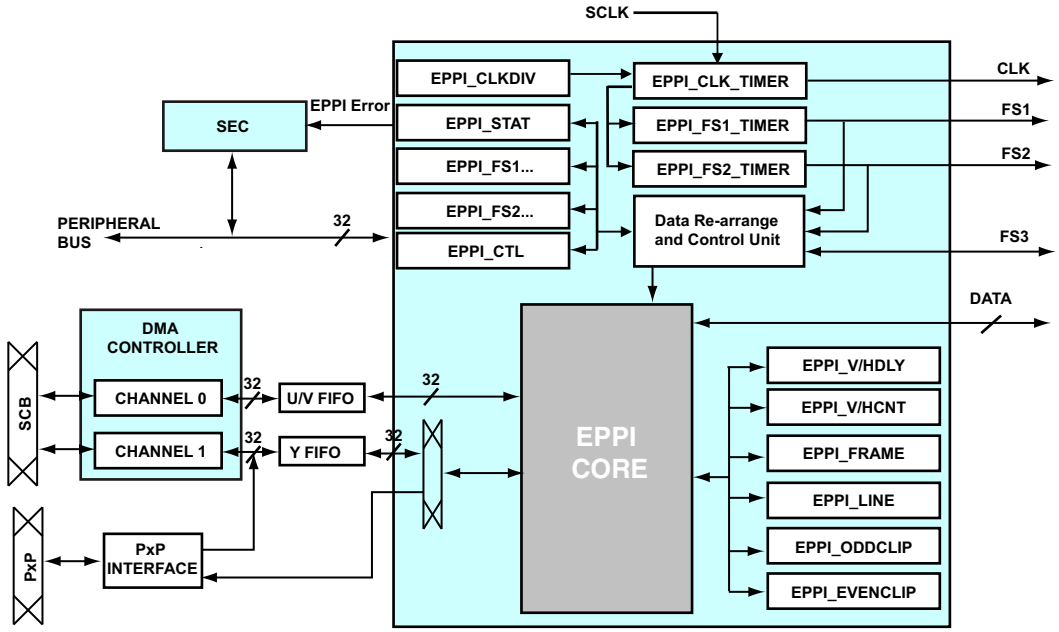


Figure 31-3: EPPI Block Diagram

EPPI Architectural Concepts

The following architectural concepts are described in this section.

- *EPPI Interface*
- *Reset Operation*
- *Frame Sync Polarity and Sampling Edge*
- *Direct Memory Access (DMA)*

- *Pixel Pipe Interface (PxP)*
- *EPPI Clock*

EPPI Interface

The EPPI has a pixel pipe (PxP) interface in addition to the interface to the system crossbar. In transmit mode the PxP interface is in the SCLK domain. In receive mode the PxP interface is in the EPPI clock domain.

A block diagram of the architecture for the EPPI interface is shown below.

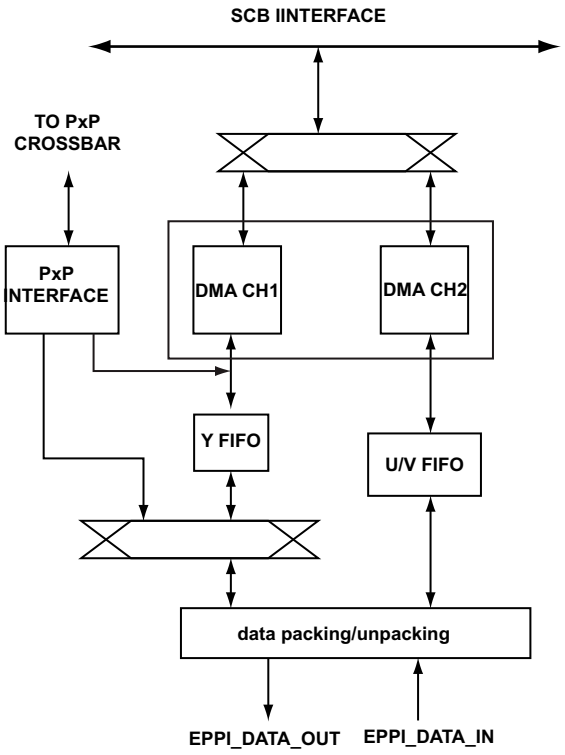


Figure 31-4: EPPI DMA and PxP Interface

In transmit operations the Y channel data path can take data from either the YFIFO or the PxP interface. This option is configured using the EPPI_CTL.DMACFG bit. When configured to take data from the PxP interface, the PxP data is transmitted on the EPPI pins in all modes. Additionally:

- Luma (and chroma) components should be made available only through the PxP interface.
- The video subsystem block informs the EPPI whether the DMA or PxP channel is available.
- When the EPPI is configured for PxP transmit, the incoming data and valid signals from the PxP are sampled on SCLK and the outgoing ready signal is driven on SCLK.

In receive operations the EPPI data can be transferred to either the PxP interface, to the YFIFO or both. When receiving data, only the data going to DMA channel 1 is sent to the PxP. This could be either the Y

data only (in split receive mode with EPPI_CTL.DMACFG = 1) or the full EPPI input pin data in any other mode. Additionally:

- If the EPPI is connected to the pixel compositor (PIXC) via the PxP, the EPPI can be programmed in single DMA mode (EPPI_CTL.DMACFG = 0) so that all of the data from the EPPI pins goes through the YFIFO (DMA channel 1).
- The video subsystem block informs the EPPI whether the DMA channel, the PxP channel, or both channels are available.

Reset Operation

On a hardware reset, the entire EPPI is reset. All MMRs return to their default values. EPPI interrupt and DMA requests become inactive and internally generated PPI_CLK and frame syncs are aborted.

In software, the EPPI can be reset and re-configured by writing 0 to the EPPI_CTL.ENbit. When disabled in this manner, only the EPPI_STATregister is cleared to its reset value. Interrupts and DMA requests become inactive and internally generated clock and frame syncs are aborted.

Frame Sync Polarity and Sampling Edge

The EPPI_CTL.POLSand EPPI_CTL.POLCbits provide a mechanism to select the active level of the frame syncs and the sampling/driving edge of the EPPI clock, respectively. This allows the EPPI to connect to data sources and receivers with a wide array of control signal polarities. Often, the remote data source/receiver also offers configurable signal polarities. In these cases, the EPPI_CTL.POLSand EPPI_CTL.POLCbits add flexibility.

Table 31-7: Frame Sync Polarity Selections and Frame Sync Pin States

Bit Setting	Frame Sync 2	Frame Sync 1
POLS = b#00	Active high/starts out low	Active high/starts out low
POLS = b#01	Active high/starts out low	Active low/starts out high
POLS = b#10	Active low/starts out high	Active high/starts out low
POLS = b#11	Active low/starts out high	Active low/starts out high

Table 31-8: Frame Clock Polarity Selections and Receive/Transmit Pin States

	Receive		Transmit	
Bit Setting	Sample Data	Sample/Drive Syncs	Drive Data	Sample/Drive Syncs
POLS = b#00	Falling edge	Falling edge	Rising edge	Rising edge
POLS = b#01	Falling edge	Rising edge	Rising edge	Falling edge
POLS = b#10	Rising edge	Falling edge	Falling edge	Rising edge
POLS = b#11	Rising edge	Rising edge	Falling edge	Falling edge

Direct Memory Access (DMA)

The EPPI has a native DMA controller with two channels. A local arbiter arbitrates between these channels and requests are forwarded to the system crossbar. The EPPI has one connection to the fabric.

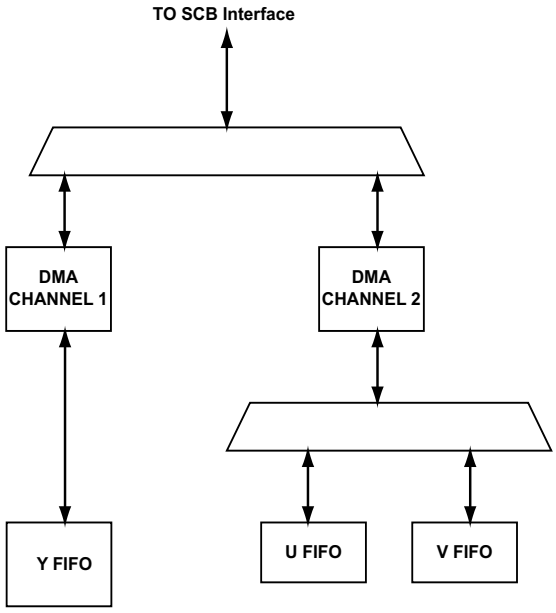


Figure 31-5: EPPI DMA Interface

The EPPI must be used with DMA and configuring the EPPI DMA channels is a necessary step toward using the EPPI interface. The channels can be configured for either transmit or receive operation, and have a maximum throughput of (PPI_CLK) x (32 bits/transfer). In modes where data lengths permit, packing may be possible in order to increase transfer bandwidth. The highest throughput is achieved with 8-bit data and packing enabled.

The DMA engine generates interrupts at the completion of a row, frame, or partial-frame transfer. The DMA engine also coordinates the source or destination point for the data that is transferred through the EPPI.

The 2D DMA capability allows the processor to be interrupted at the end of a line or after a frame of video is transferred, or if a DMA error occurs. The DMA_XCNT and DMA_YCNTregisters allow for flexible data interrupt points. For example, assume the DMA_XMOD = DMA_YMOD = 1. If a data frame contains 320 × 240 bytes (240 rows of 320 bytes each), the following conditions hold.

- Setting DMA_XCNT = 320, DMA_YCNT = 240, and DISEL = 1 (the DISEL bit is located in DMA_CFG register) interrupts on every row transferred, for the entire frame.
- Setting DMA_XCNT = 320, DMA_YCNT = 240, and DISEL = 0 interrupts only on the completion of the frame (when 240 rows of 320 bytes have been transferred).
- Setting DMA_XCNT = 38,400 (320 x 120), DMA_YCNT = 2, and DISEL =1 causes an interrupt when half of the frame is transferred, and again when the whole frame is transferred.

The following is the general procedure for setting up DMA operation with the EPPI.

1. Configure the DMA registers as appropriate for the desired DMA operating mode.
2. Enable the DMA channel for operation.
3. Configure appropriate EPPI registers.
4. Enable the EPPI by writing 1 to the `EPPI_CTL.EN` bit.

Pixel Pipe Interface (PxP)

The EPPI has a pixel pipe (PxP) interface in addition to the SCB interface to the fabric. In transmit, the PxP interface is in the SCLK domain while in receive, the interface is in the EPPI clock domain.

If the EPPI is configured for transmit, the Y channel data path can take data from either the YFIFO or the PxP crossbar. When configured to take the data from PxP crossbar (`DMA_CFG`register = 0), the PxP data is transmitted on the EPPI pins in all modes. Additionally:

- Luma (and Chroma) components should be made available only through PxP.
- The video subsystem blocks (VID) determine the control signals for deciding the channel (DMA or PxP) from which the data is taken. Refer to the “Video Subsystem (VID)” chapter for details.
- When using PxP transmit, the incoming data and valid signal from the PxP are sampled on the `SCLK` and the outgoing ready signal is driven on *SCLK*. This implies the PIXC has to drive out the data in *SCLK*.

The EPPI block with the pixel pipe addition is shown below.

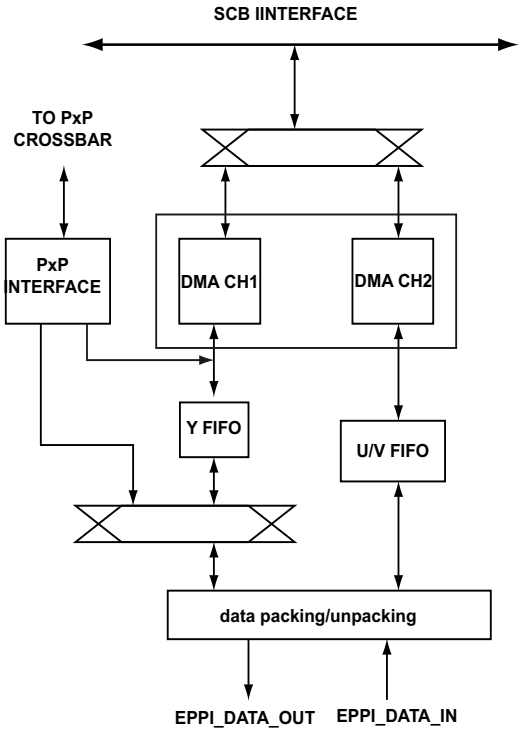


Figure 31-6: EPPI Block with the Pixel Pipe Addition

For receive operations EPPI data is transferred to either the PxP crossbar, the YFIFO, or both and only the data going to DMA channel 1 is sent to the PxP. This is either the Y data only (in split receive mode with `DMA_CFG`register = 1) or the full EPPI input pin data in any other mode. If the EPPI is connected to the PIXC via the PxP, the EPPI can be programmed in single DMA mode (`DMA_CFG`register = 0) so that all the data from the EPPI pins goes through the Y FIFO (DMA channel 1).

The video subsystem block informs the EPPI whether the DMA channel, the PxP channel, or both channels are available.

The following figures describe the design level details of how the PxP interface is multiplexed with the existing data path in transmit and receive case.

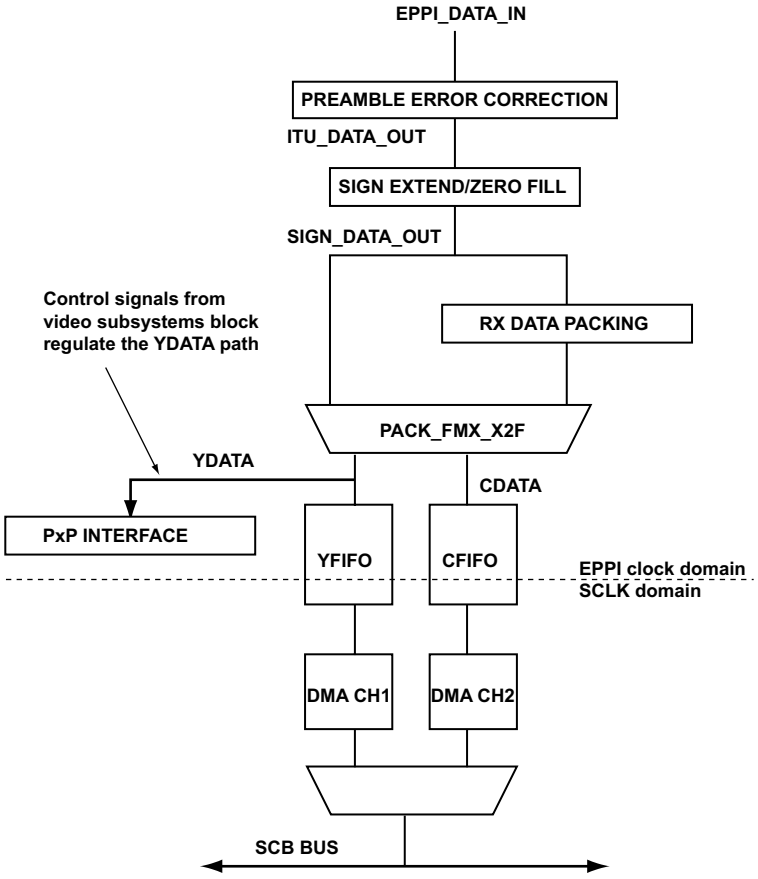


Figure 31-7: Receive Data Path with PxP Interface Muxing

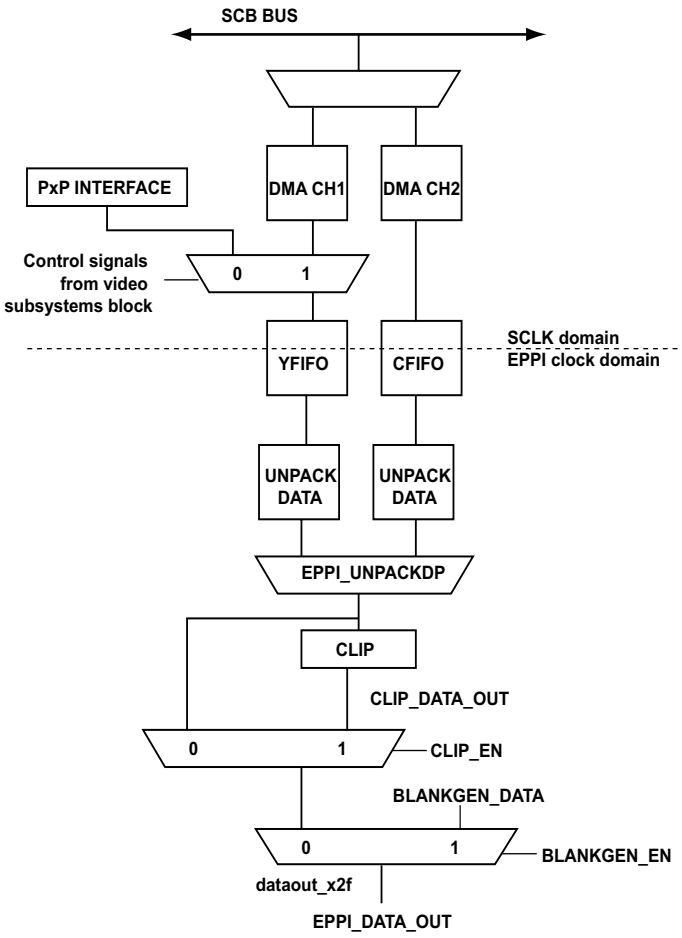


Figure 31-8: Transmit Data Path with PxP Interface Muxing

EPPI Clock

The EPPI can be supplied with an external clock, or the clock can be generated internally and supplied to external devices. For information on the maximum PPI_CLK specification in internal and external clock modes see the product specific data sheet.

When using an external `PPI_CLK`, there may be up to two cycles latency before valid data is received or transmitted.

The internal clock can be generated from `SCLK` if the `EPPI_CTL.ICLKGEN` bit is set. The generated clock frequency is then determined by the value in the `EPPI_CLKDIV`register. The internally generated EPPI clock frequency is:

$f_{PCLK} = f_{SCLK}/(\texttt{EPPI\_CLKDIV} + 1)$

where:

f_{PCLK} – frequency of internally generated EPPI clock

f_{SCLK} – frequency of *SCLK*

EPPI_CLKDIV – Clock division value programmed in the EPPI_CLKDIVregister.

A few examples are given in the table below.

Table 31-9: Relationship Between CLKDIV and the Ratio of SCLK to EPPI Clock

CLKDIV15–0	SCLK:EPPI Clock Ratio
0x0002	1:3
0x0003	1:4
0x0004	1:5
0x0005	1:6
...	...

EPPI Operating Modes

The EPPI supports various receive and transmit modes of operation which include the detection and generation of preamble data. Specifically, the EPPI supports data formats described in the specifications ITU-656, SMPTE 274M and SMPTE 296M. In addition to these modes, the EPPI also supports general purpose receive and transmit using up to three frame syncs (FS).

Most of the bits used for configuring operating modes are located in the control register (EPPI_CTL). Complete descriptions of these bits can be found in the “Register Descriptions” section of this chapter.

ITU-R 656 Modes

The EPPI supports three input modes and one output mode for ITU-R 656 framed data. These modes are described in this section.

ITU-R 656 Background

In ITU-R 656 (formerly known as CCIR-656) mode, the horizontal (H), vertical (V), and field (F) signals are sent as an embedded part of the video data stream in a series of bytes that form a control word.

The letter H is used to distinguish between the *start of active video* (SAV) and *end of active video* (EAV) signals, which indicate the beginning and end of active video data in each line. The SAV occurs on a 1-to-0 transition of H, and EAV occurs on a 0-to-1 transition of H. The space between EAV and SAV is filled with horizontal blanking data. Therefore H = 1 during the horizontal blanking portion of the data stream and H = 0 during the active video portion of the data stream.

The letter V is used to denote the vertical blanking portion of the data stream. A transition in V can occur only in the EAV sequence. When V = 1, the data stream contains vertical blanking data and when V = 0, the data stream contains active video data.

The letter F is used to distinguish Field 1 from Field 2. Interlaced video has two fields in a frame of data. It requires each field to be handled uniquely, and alternate rows of each field combined to create the actual video image.

For interlaced video, F = 0 represents Field 1 (*Odd Field*) and F = 1 represents Field 2 (*Even Field*). Progressive video makes no distinction between Field 1 and Field 2, and F is always 0 for progressive video. Interlaced video requires each field to be handled uniquely, because alternate rows of each field combine to create the actual video image.

An entire field of video is comprised of active video plus horizontal blanking (the space between an EAV and SAV code) and vertical blanking (the space where V = 1). A field of video commences on a transition of the F bit.

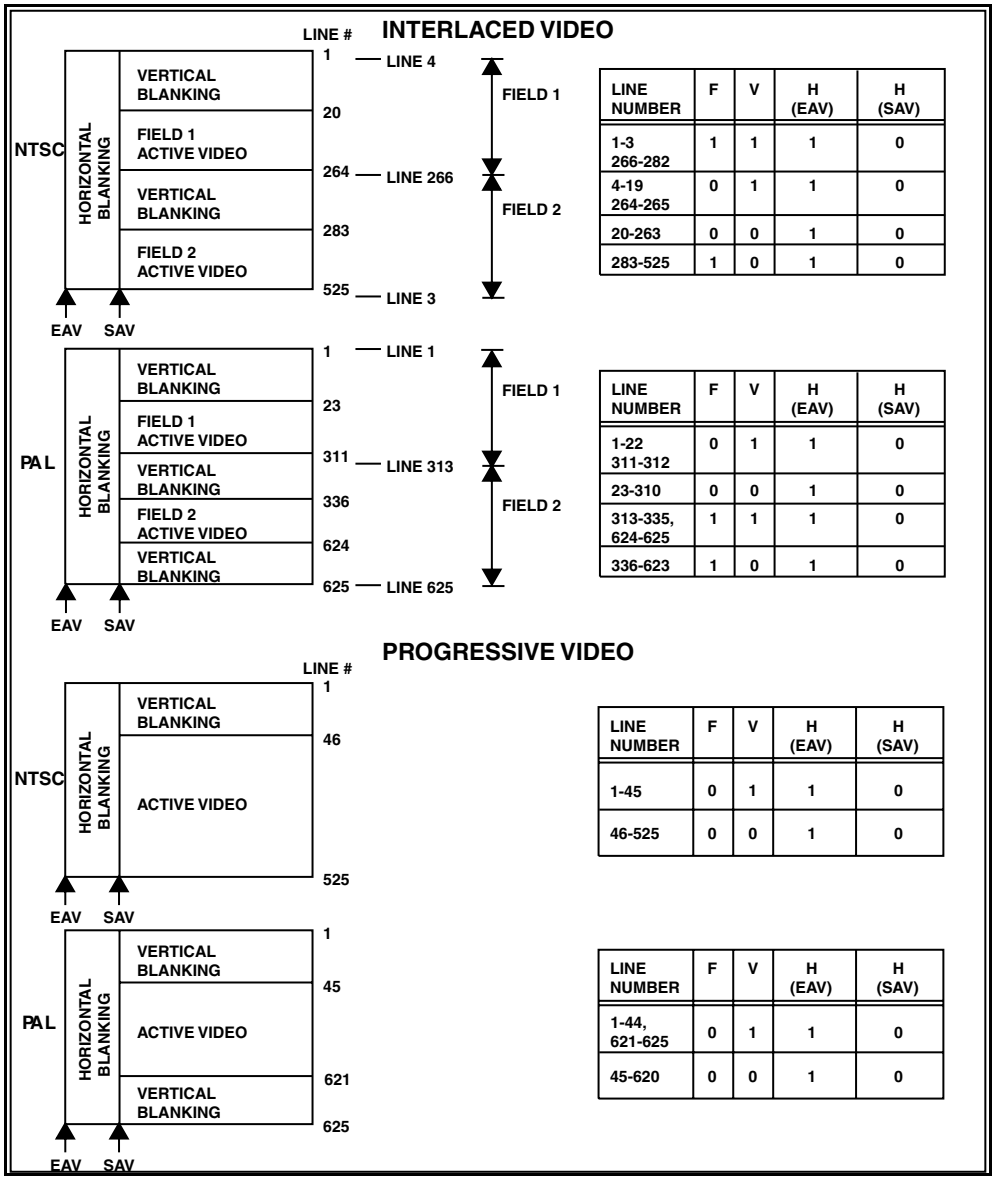


Figure 31-9: Typical Video Frame Partitioning for NTSC/PAL Systems in Interlaced and Progressive ITU-R BT.656 Systems

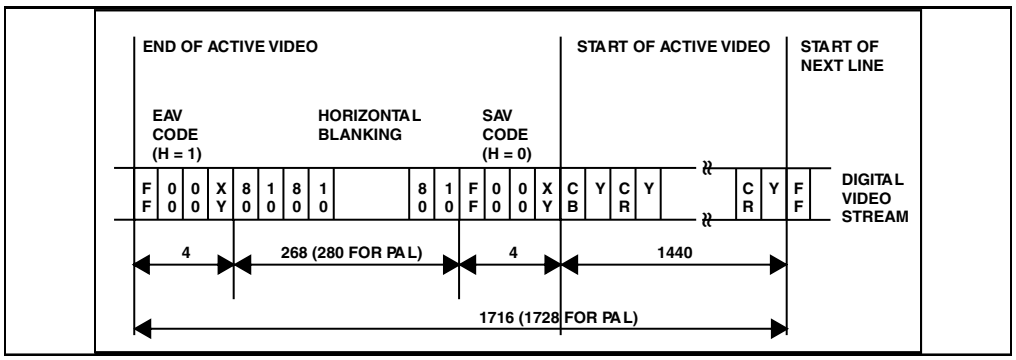


Figure 31-10: ITU-R 656 8-Bit Parallel Data Stream fro NTSC (PAL) Systems

NOTE: As shown in the following table, there is a defined preamble of three data elements (for example, in the case of 8-bit video: 0xFF, 0x00, 0x00), followed by the XY status word, which, aside from the F (field), V (vertical blanking) and H (horizontal blanking) bits, contains four protection bits for error detection and correction. F and V are only allowed to change as part of EAV sequences (that is, transition from H = 0 to H = 1).

The bit definitions are as follows:

- F = 0 for field 1
- F = 1 for field 2
- V = 1 during vertical blanking
- V = 0 when not in vertical blanking
- H = 0 at SAV
- H = 1 at EAV
- P3 = V XOR H
- P2 = F XOR H
- P1 = F XOR V
- P0 = F XOR V XOR H

P3–P0 are protection bits and enable 1 and 2-bit errors to be detected, and 1-bit errors to be corrected, at the receiver. The EPPI performs the correction if it detects 1-bit errors in F, V, or H. Errors in the protection bits themselves are detected but not corrected.

Table 31-10: Control Sequences for 8-Bit and 10-Bit ITU-R 656 Video

	8-Bit Data								10-Bit Data	
	D9 (MSB	D8	D7	D6	D5	D4	D3	D2	D1	D0
Preamble	1	1	1	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
Control Byte	1	F	V	H	P3	P2	P1	P0	0	0

The EPPI_STATregister contains two bits, EPPI_STAT.ERRDET and EPPI_STAT.ERRNCOR, that are used to report the status of error detected and error not corrected, respectively.

The EPPI_STAT.ERRDET bit is set whenever an error is detected in the status word. However, this bit does not generate an interrupt. The EPPI_STAT.ERRNCOR bit is set when more than a 1-bit error is detected in the status word. An interrupt is generated when the EPPI_STAT.ERRNCOR bit is set. It can be cleared by clearing the EPPI_STAT.ERRNCOR and EPPI_STAT.ERRDET bits. Both bits are sticky and W1C.

In many applications, video streams other than the standard NTSC/PAL formats (for example, CIF, QCIF) can be employed. Because of this, the processor interface is flexible enough to accommodate different row and field lengths. In general, as long as the incoming video has the proper EAV/SAV codes, the EPPI can read it in. In other words, a CIF image could be formatted to be 656-compliant, where EAV and SAV values define the range of the image for each line, and the V and F codes are used to delimit fields and frames.

The following sections provide descriptions of EPPI operations.

Table 31-11: Operating Modes and Generic EPPI Operation

		How to configure	Useful for	How to configure in ITU R 656 TX Mode
ITU-R BT.656 RX	Entire field	DIR= 0 XFRTYPE = b#01		
	Active video	DIR = 0 XFRTYPE = b#00		
	Blanking only	DIR = 0 XFRTYPE = b#10		
GP 0 FS	TX	DIR = 1 XFRTYPE = b#11 FSCFG = b#00	Applications where periodic frame syncs are not used to frame the data	BLANKGEN = 1 DLEN = (b#000, b#001 or b#100)
	RX	DIR = 0 XFRTYPE = b#11 FSCFG = b#00		

Table 31-11: Operating Modes and Generic EPPI Operation (Continued)

		How to configure	Useful for	How to configure in ITU R 656 TX Mode
GP 1 FS	TX	DIR = 1 XFRTYPE = b#11 FSCFG =b#01	Interfacing with ADCs, DACs and other general-purpose devices	BLANKGEN = 1 DLEN = (b#000, b#001 or b#100)
	RX	DIR = 0 XFRTYPE = b#11 FSCFG = b#01		
GP 2 FS	TX	DIR = 1 XFRTYPE = b#11 FSCFG = b#10	Video applications that use two hardware synchronization signals, HSYNC and VSYNC	BLANKGEN = 1 DLEN = (b#000, b#001 or b#100)
	RX	DIR = 0 XFRTYPE = b#11 FS_CFG = b#10		
GP 3 FS	TX	DIR = 1 XFRTYPE = b#11 FSCFG = b#11	Video applications that use three hardware sync signals, HSYNC, VSYNC, and FIELD	BLANKGEN = 1 DLEN = (b#000, b#001 or b#100)
	RX	DIR = 0 XFRTYPE = b#11 FSCFG = b#11		

ITU-R 656 Input Modes

In the ITU-R 656 input modes, the clock is either provided by the video source or supplied externally by the system.

As shown in the following figure and described in the following sections, there are three sub-modes supported for ITU-R 656 inputs: entire field, active video only, and vertical blanking interval only.

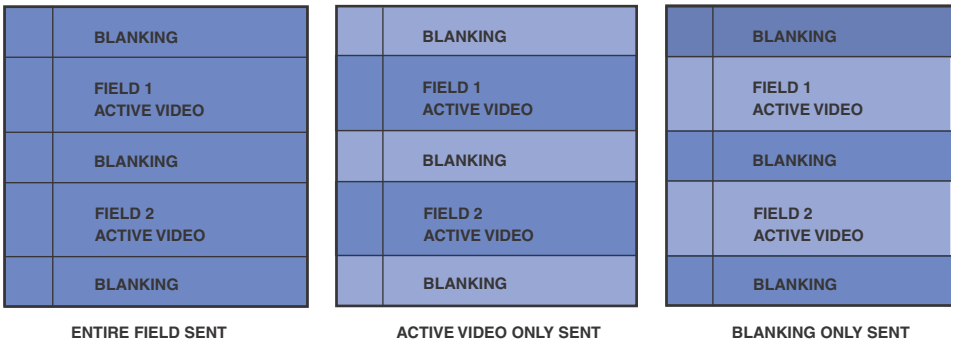


Figure 31-11: ITU-R 656 Input Sub-modes

Entire Field

In this mode, the entire incoming bit stream is read in through the EPPI. This includes active video as well as control byte sequences and ancillary data that may be embedded in horizontal and vertical blanking intervals.

Data transfer starts immediately after Field 1 synchronization occurs, but does not include the first EAV code that contains the F = 0 assignment for interlaced video, or V = 0 assignment for progressive video.

Active Video

This mode is used when only the active video portion of a field is of interest, and not any of the blanking intervals. The EPPI ignores (does not read in) all data between EAV and SAV, as well as all data present when V = 1. Furthermore, the control byte sequences are not stored to memory, they are filtered out by the EPPI. After the start of Field 1 synchronizes, the EPPI ignores incoming samples until it sees an SAV.

In active video mode, programs must specify the number of total (active plus vertical blanking) lines per frame in the `EPPI_FRAME` register, and the number of total (active plus horizontal blanking plus 8) samples per line in the `EPPI_LINE` register.

In this mode (as well as vertical blanking interval mode), any input data sequence that is considered part of the preamble (for example in 8-bit ITU mode, 0xFF or 0x00 appearing in the input data stream are considered part of the preamble) is not sent to memory, even if it appears individually, and is not tagged along with the preamble sequence FF, 00, 00.

Vertical Blanking Interval (VBI)

In this mode, data transfer is only active while V = 1 is in the control byte sequence. This indicates that the video source is in the midst of the vertical blanking interval (VBI), which is sometimes used for ancillary data transmission. The ITU-R 656 recommendation specifies the format for these ancillary data packets, but the EPPI is not equipped to decode the packets themselves. This task must be handled in software. Horizontal blanking data is logged where it coincides with the rows of the VBI.

The VBI is split into two regions within each field. From the EPPI's standpoint, it considers these two separate regions as one contiguous space. However, keep in mind that frame synchronization begins at the start of Field 1, which doesn't necessarily correspond to the start of vertical blanking. For instance, in 525/60 systems, the start of Field 1 (F = 0) corresponds to line 4 of the VBI.

In VBI mode, the program must specify the number of total (active plus vertical blanking) lines per frame in the `EPPI_FRAME` register, and the number of total (active plus horizontal blanking plus 8) samples per line in the `EPPI_LINE` register.

In this mode (as well as active video mode), any input data sequence that is considered as part of the preamble (for example in 8-bit ITU mode, if 0xFF or 0x00 appears in the input data stream these values are considered part of the preamble) is not sent to memory, even if it appears individually, and is not tagged along with the preamble sequence FF, 00, 00.

ITU-R 656 Output in General-Purpose Transmit Modes

In GP transmit mode, the EPPI provides the ability to frame an ITU-R 656 output stream with the proper preambles and blanking intervals. This is done by setting the EPPI_CTL.BLANKGEN bit. The EPPI then only needs to fetch active data from memory through the DMA channel, saving DMA bandwidth. The EPPI_FS1_PASPL, EPPI_FS2_WLVB, EPPI_FS2_PALPF and EPPI_FS1_WLHB registers need to be programmed correctly in order for the EPPI to internally generate and embed the proper preamble, status word (EAV and SAV sequences) and blanking data along with the active video from memory. The EPPI can also drive out the frame syncs based on the EPPI_CTL.FSCFG bit setting.

The figure below shows the bit stream format in 16-bit transmit modes with blanking generation (EPPI_CTL.BLANKGEN enabled). Each 16-bit data sample consists of 8-bit luma (Y) and 8-bit chroma (Cr or Cb) components. During transmission, the chroma data and the 0x80 blanking bytes are on the upper half (MSBs) of the data lines while the luma data and the 0x10 blanking bytes are on the lower half (LSBs) of the data lines.

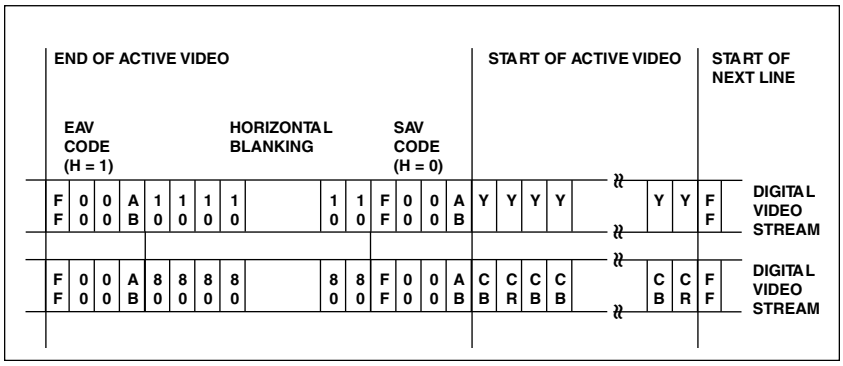


Figure 31-12: 16-Bit Transmit with Internal Blanking Generation

The following figure shows the data transmitted by the EPPI in this mode. After the EPPI is enabled and if the EPPI FIFO is not empty, the transmission starts by sending out a EAV sequence for a vertical blanking line. For interlaced video, F starts at 1. For progressive video, F is always 0.

NOTE: Internal blanking generation functionality is valid only when the data length is 8, 10, or 16 bits and when the EPPI is in GP transmit modes. The EPPI_CTL.BLANKGEN bit generates preambles even in GP 2FS mode.

The ITU-R 656 output mode's internal blanking generation functionality can also be bypassed (for instance, if sending ancillary data in the blanking interval) by clearing the EPPI_CTL.BLANKGEN bit. The EPPI_CTL.BLANKGEN bit generates preambles even in GP 2FS mode.

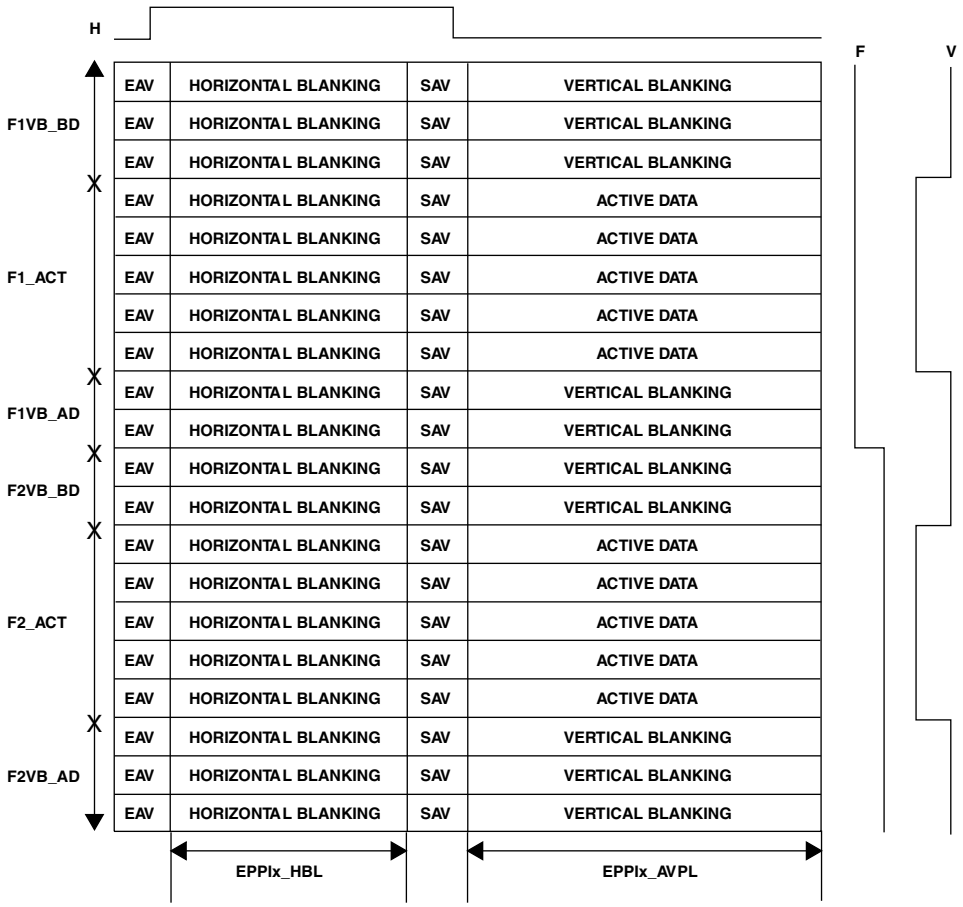


Figure 31-13: Generated Blanking Preamble Sequence

Frame Synchronization in ITU-R 656 Modes

For interlaced video, the start of frame synchronization occurs when a high-to-low transition is detected in F, the field indicator. For progressive video, the start of frame synchronization occurs when a high-to-low transition is detected in V, the vertical blanking indicator. These transitions in F and V can occur only in the EAV sequence. A start of line is detected on a low-to-high transition in H, the horizontal blanking indicator, which occurs in the EAV sequence as well.

For interlaced video, the start of frame corresponds to the start of field 1. Consequently, up to two fields may be ignored (for example, if field 1 started before the EPPI-to-camera channel was established) before data is received into the EPPI. For progressive video, the start of frame corresponds to the start of active video.

Because all H and V signaling is embedded in the data stream in ITU-R 656 modes, the count registers (`EPPI_HCNT`, `EPPI_VCNT`) are ignored. However, the `EPPI_FRAME` register is still used to check for synchronization errors. Therefore, this MMR must be programmed with the number of lines expected in each frame of video.

The EPPI keeps track of the number of EAV-to-SAV transitions that occur from the start of a frame until it decodes the end-of-frame condition (transition from F = 1 to F = 0 in the case of interlaced video and transition from V = 1 to V = 0 in the case of progressive video).

At the end-of-frame condition, the actual number of lines processed is compared against the value in EPPI_FRAME. If there is a mismatch, a frame track error is asserted in the EPPI_STAT register. For instance, if an SAV transition was missed, the current field only has NUM_ROWS – 1 rows, but re synchronization occurs at the start of the next frame. When the entire field is received, the field status bit is toggled in the EPPI_STAT register. This way, an interrupt service routine (ISR) can discern which field was just read in.

General-Purpose EPPI Modes

The general-purpose (GP) EPPI modes are intended to suit a wide variety of data capture and transmission applications.

Each EPPI has three bidirectional frame sync pins. Frame syncs can be generated internally by the EPPI, or by an external device communicating with the EPPI.

GP modes can be distinguished based on the number of frame syncs used and the EPPI supports GP 0 FS—GP 3 FS modes.

All the GP modes, except 0 FS mode, support horizontal windowing. GP modes with 2 and 3 frame syncs also support vertical windowing.

For GP transmit modes with internal clock or internal frame syncs, the EPPI starts generating the clock or frame syncs only when the EPPI FIFO is full for the first time. For GP 0 FS transmit mode, the EPPI only starts transmitting when the EPPI FIFO is full for the first time.

General-Purpose 0 Frame Sync Mode

This mode is useful for applications where periodic frame syncs are not used to frame the data.

After the initial trigger, the EPPI receives/transmits data samples on every clock cycle. However, if the EPPI_CTL.SKIPEN bit is set for receive mode, the EPPI receives only alternate data samples.

The EPPI_LINE, EPPI_FRAME, EPPI_HCNT, EPPI_HDLY, EPPI_VCNT and EPPI_VDLY registers are not valid for GP 0 FS mode. Therefore windowing is not possible in this mode. Also, line and frame track errors are not applicable in this mode.

GP 0 FS receive mode is further divided into two sub-modes; internal trigger (EPPI_CTL.FLDSEL bit=1) and external trigger (EPPI_CTL.FLDSEL bit=0), based on how data transmission/reception is initiated. GP 0 FS transmit mode is always internally triggered. All subsequent data manipulation is handled through DMA.

- **Frame synchronization in GP 0 FS external trigger mode.** When the EPPI is programmed in external trigger mode, it does not generate the PPI_FS1 signal and a trigger must be provided by the external

device. The EPPI starts receiving the data as soon as an PPI_FS1 signal assertion is detected. After that, all subsequent data manipulation is handled by way of DMA and any activity on PPI_FS1 is ignored.

- **Frame synchronization in GP 0 FS internal trigger mode.** When the EPPI is programmed in internal trigger mode, it starts receiving/transmitting data as soon as the EPPI clock is enabled and synchronized. There may be up to two cycles latency before valid data is received or transmitted.

General-Purpose 1 Frame Sync Mode

This mode is useful for interfacing the EPPI with analog-to-digital converters (ADCs), digital-to-analog converters (DACs) and other general-purpose devices. This mode works for both transmit and receive.

The EPPI_FRAME, EPPI_VDLY and EPPI_VCNT registers have no effect in GP 1 FS mode. As a result, frame track errors and vertical windowing are not available.

General-Purpose 2 Frame Sync Mode

This mode is useful for video applications that use two hardware synchronization signals, HSYNC and VSYNC. The HSYNC signal can be connected to PPI_FS1and the VSYNC signal can be connected to PPI_FS2.

Data Enable in General-Purpose 2 Frame Sync Transmit Mode

When configured in GP 2 FS transmit mode and for internal frame sync generation, with EPPI_CTL.MUXSEL and EPPI_CTL.CLKGATEN bits not enabled, the PPI_FS3 pin functions as a data enable (DEN) pin. The functionality of the DEN pin is described in the following two cases.

Case 1 – When blanking generation is configured using the EPPI_CTL.BLANKGEN bit and the EPPI data length (EPPI_CTL.DLEN bit) is configured for 8, 10, or 16-bit transfers, the PPI_FS3 pin asserts during the *active data* regions, aligned with PPI_CLK according to the clock polarity (EPPI_CTL.POLC bit) settings. For this mode PPI_FS3 is driven based on the EPPI_CTL.POLC setting (PPI_FS3 is driven out on the same EPPI clock edge that drives out data). The frame sync polarity (EPPI_CTL.POLS) setting does not apply here—PPI_FS3 is always active high in this mode.

Case 2 – When blanking generation (EPPI_CTL.BLANKGEN=0) is disabled, or it is enabled but the EPPI data length (EPPI_CTL.DLEN bit) is configured for a transfer size different from 8, 10, or 16-bits, the PPI_FS3 pin asserts at the start of the active data region on each line, aligned with PPI_CLK according to the EPPI_CTL.POLC bit settings. For this mode PPI_FS3 is driven based on the EPPI_CTL.POLC setting (the PPI_FS3 signal is driven out on the same EPPI clock edge that drives out data).

The EPPI_CTL.POLS bit setting does not apply for case 2. The PPI_FS3 signal is always active high in this mode. Once asserted, PPI_FS3 stays asserted for the number of clock cycles per line configured in the EPPI_HCNT register, then it deasserts. This behavior on each line continues for the total number of lines programmed in the EPPI_VCNT register per frame, and repeats at the start of subsequent video frames.

Also in case 2, if transmission of valid data is held off due to delays programmed in the EPPI_HDLY and/or EPPI_VDLY registers, the assertion of PPI_FS3 is also held off accordingly, on a per-line and/or per-frame basis.

General-Purpose 3 Frame Sync Mode

This mode is useful for video applications that use three hardware synchronization signals, HSYNC, VSYNC, and FIELD. The HSYNC can be connected to the PPI_FS1 pin, VSYNC can be connected to the PPI_FS2 pin, and FIELD can be connected to the PPI_FS3 pin.

GP 3 FS mode is similar in operation to GP 2 FS mode, except that the start of frame synchronization in GP 3 FS mode also takes into account the PPI_FS3 pin. All the windowing register settings (EPPI_FRAME, EPPI_LINE, EPPI_HDLY, EPPI_HCNT, EPPI_VDLY and EPPI_VCNTregisters), as well as data reception/transmission and error generation are the same as for GP 2 FS mode. In addition, for GP 3 FS mode with internal frame syncs, the EPPI_CTL.FLDSEL bit setting specifies the condition under which the transfer begins.

The PPI_FS3signal is generated by the EPPI and toggles during every assertion of PPI_FS2 or a combination of PPI_FS2 and PPI_FS1(depending on the EPPI_CTL.FLDSEL bit setting). The EPPI skips an PPI_FS2signal if the PPI_FS3value is high. Because of this condition, as a programming guideline, the PPI_FS2period value should be half of the total number of pixels in the frame as in GP 3 FS mode. When in GP 2 FS mode, the PPI_FS2period should be programmed with the value equal to the number of pixels per frame.

Supported Data Formats

The following sections describe EPPI receive and transmit data formats.

Receive Data Formats

The following table provides information about EPPI configuration for specific use models for receive data.

Table 31-12: EPPI Receive Data Formats

Input Data Width	Use Model	Splitting/Packing Options
8	NTSC/PAL data	EPPI_CTL.SPLTEO=1 EPPI_CTL.SUBSPLTODD=1 if required to separate chroma components
	RGB sensor	No splitting possible. EPPI_CTL.PACKEN=1 – Four EPPI words are packed to 32-bit DMA data. EPPI_CTL.PACKEN=0 – Each EPPI word is sent as 8-bit data on the 32-bit DMA bus. This consumes 4 times the DMA bandwidth of the 8-bit case with EPPI_CTL.PACKEN=1;
	ADCs	Gives I (in phase) and Q (quadrature) components. EPPI_CTL.SPLTEO=1 EPPI_CTL.SUBSPLTODD=0 since there are only two components.
10	NTSC/PAL data	Each EPPI word is zero filled/sign extended to 16 bits. EPPI_CTL.SPLTEO=1. EPPI_CTL.SUBSPLTODD=1 if required to separate chroma components.
	RGB sensor	No splitting possible. EPPI_CTL.PACKEN=1. Two EPPI words are zero filled/sign extended to 16 bits and packed to 32-bit DMA data. EPPI_CTL.PACKEN=0. Each EPPI word may be zero filled/sign extended to 16 bits and sent as a 16 bit data on the 32-bit DMA bus. This consumes double the bandwidth of the 10-bit case with EPPI_CTL.PACKEN=1;
	ADCs	Each EPPI word is zero filled/sign extended to 16 bits. EPPI_CTL.SPLTEO=1 SEPPI_CTL.SUBSPLTODD= 0 since there are only two components.

Table 31-12: EPPI Receive Data Formats (Continued)

Input Data Width	Use Model	Splitting/Packing Options
12	RGB sensor	No splitting possible. EPPI_CTL.PACKEN=1. Two EPPI words are zero filled/sign extended to 16 bits and packed to 32-bit DMA data. EPPI_CTL.PACKEN=0. Each EPPI word may be zero filled/sign extended to 16 bits and sent as a 16 bit data on the 32-bit DMA bus. This consumes double the bandwidth of the 12-bit case with EPPI_CTL.PACKEN=1;
	ADCs	Each EPPI word is zero filled/sign extended to 16 bits. EPPI_CTL.SPLTEO=1 EPPI_CTL.SUBSPLTODD=0 since there are only two components.
14	ADCs	Each EPPI word is zero filled/sign extended to 16 bits. EPPI_CTL.SPLTEO=1 EPPI_CTL.SUBSPLTODD=0 since there are only two components.
16	8-bit luma/chroma pair for NTSC or HD	EPPI_CTL.SPLTEO=1, EPPI_CTL.SPLTWRD=1, EPPI_CTL.SUBSPLTODD=1 if required to separate chroma components.
	16-bit luma/chroma pair for NTSC or HD	EPPI_CTL.SPLTEO=1, EPPI_CTL.SPLTWRD=0, EPPI_CTL.SUBSPLTODD=1 if required to separate chroma components.
	RGB565 sensor	No splitting possible. EPPI_CTL.PACKEN=1. Two EPPI words are packed to a 32 bit DMA data. EPPI_CTL.RGBFMTEN is valid only in transmit modes. So, RGB565 cannot be made byte aligned in memory. EPPI_CTL.PACKEN=0. Each EPPI word is sent as a 16 bit data on the 32 bit DMA bus. This consumes double the bandwidth of the 16 bit case with EPPI_CTL.PACKEN=1
	8-bit ADCs I/Q pair	EPPI_CTL.SPLTEO=1, EPPI_CTL.SPLTWRD=1, EPPI_CTL.SUBSPLTODD=0.
	16-bit ADCs I/Q pair	EPPI_CTL.SPLTEO=1, EPPI_CTL.SPLTWRD=0, EPPI_CTL.SUBSPLTODD=0.
20	10-bit luma/chroma pair for NTSC or HD	No support for 20-bit mode.

Table 31-12: EPPI Receive Data Formats (Continued)

Input Data Width	Use Model	Splitting/Packing Options
24	RGB888 sensor	No splitting possible. EPPI_CTL.PACKEN=1. Four EPPI words are packed to three 32-bit DMA data. EPPI_CTL.PACKEN=0. Each EPPI word is sent as a 24 bit data on the 32 bit DMA bus.

Transmit Data Formats

The following table provides information about EPPI configuration for specific use models for transmit data.

Table 31-13: EPPI Transmit Data Formats

Output Data Width	Use Model	Splitting/Packing Options
8	NTSC/PAL data	EPPI_CTL.SPLTEO=1 EPPI_CTL.SUBSPLTODD=1 if the chroma components (U and V) come in separate DMA words.
	Serial RGB for lower-resolution LCDs	No splitting possible. EPPI_CTL.PACKEN=1. The 32 bit DMA data is unpacked to drive four EPPI words. EPPI_CTL.PACKEN=0. The lowest 8 bits of the DMA data is driven on the EPPI data and the rest of the 24 bits are discarded. This consumes 4 times the DMA bandwidth of the 8 bit case with EPPI_CTL.PACKEN=1.
10	NTSC/PAL data	EPPI_CTL.SPLTEO=1 EPPI_CTL.SUBSPLTODD=1 if the chroma components (U and V) come in separate DMA words.
	DACs	EPPI_CTL.SPLTEO=1, EPPI_CTL.SUBSPLTODD=0.
12	DACs	EPPI_CTL.SPLTEO=1, EPPI_CTL.SUBSPLTODD=0.
14	DACs	EPPI_CTL.SPLTEO=1, EPPI_CTL.SUBSPLTODD=0.

Table 31-13: EPPI Transmit Data Formats (Continued)

Output Data Width	Use Model	Splitting/Packing Options
16	8-bit luma/chroma pair for NTSC or HD	EPPI_CTL.SPLTEO=1, EPPI_CTL.SPLTWRD=1, EPPI_CTL.SUBSPLTODD=1 if the chroma components (U and V) come in separate DMA words.
	16-bit luma/chroma pair for NTSC or HD	EPPI_CTL.SPLTEO=1, EPPI_CTL.SPLTWRD=0, EPPI_CTL.SUBSPLTODD=1 if the chroma components (U and V) come in separate DMA words.
	RGB565 LCD	No splitting possible. EPPI_CTL.RGBFMTEN=1. Takes RGB888 data from the memory and drops the LSBs from each component to drive out RGB565 data.
	8-bit ADCs I/Q pair	EPPI_CTL.SPLTEO=1, EPPI_CTL.SPLTWRD=1, EPPI_CTL.SUBSPLTODD= 0
	16-bit ADCs I/Q pair	EPPI_CTL.SPLTEO=1, EPPI_CTL.SPLTWRD=0, EPPI_CTL.SUBSPLTODD=1
18	RGB666 LCD	No splitting possible. EPPI_CTL.RGBFMTEN=1. Takes RGB888 data from the memory and drops the 2 LSBs from each component to drive out RGB666 data.
24	RGB888 LCD	No splitting possible. EPPI_CTL.RGBFMTEN=1. Takes three 32-bit DMA data and drives out four 24-bit data on the EPPI pins. EPPI_CTL.PACKEN=0. Takes the 32-bit DMA word, drops the 8 MSBs and drives out the remaining 24 bits on the EPPI pins

Data Transfer Modes

The following sections describe EPPI data transfer modes, including receive/transmit data packing, sign extension, zero fill, receive/transmit split modes, clock gating, delayed start, and data consistency management.

Data Packing for Receive Modes

For receive modes, if the EPPI_CTL.PACKEN bit=1 and the DMA is 32 bits, the EPPI packs the incoming data into 32-bit words based on the EPPI_CTL.DLENand EPPI_CTL.SWAPEN bit settings. When EPPI_CTL.SWAPEN=0, the EPPI puts the first data in the least significant bits and when EPPI_CTL.SWAPEN=1, the EPPI

puts the first data in the most significant bits. The packing options for the EPPI_CTL.DLENbits are as follows.

- When EPPI_CTL.DLEN=8, four 8-bit words can be packed into one 32-bit word.
- When EPPI_CTL.DLEN=16, two 16-bit words can be packed into one 32-bit word.
- For EPPI_CTL.DLEN values that are more than 8 bits but less than 16 bits, two such words are either sign-extended or zero-filled to 16 bits, and packed into one 32-bit word.
- When EPPI_CTL.DLEN=18, the EPPI sign-extends or zero-fills the 18-bit data to 24 bits and packs four 24-bit words into three 32-bit words.
- When EPPI_CTL.DLEN=24, the EPPI packs four 24-bit words into three 32-bit words.

When EPPI_CTL.PACKEN=0, the EPPI receives the incoming data and sends it on the bus as-is. If EPPI_CTL.DLEN is less than or equal to 16 bits, the DMA is a 16-bit DMA; otherwise it is a 32-bit DMA.

Data Packing for Transmit Modes

For transmit modes, if the EPPI_CTL.DLENbit=1 and the DMA is a 32-bit DMA, the EPPI unpacks the 32-bit word according to the EPPI_CTL.DLENand EPPI_CTL.SWAPEN bit settings.

If EPPI_CTL.SWAPEN=1, the EPPI transmits the most significant bits as the first data, and if EPPI_CTL.SWAPEN=0, the EPPI transmits the least significant bits as the first data. The unpacking options for the EPPI_CTL.DLEN bits are as follows.

- When EPPI_CTL.DLEN=8, the EPPI transmits one 32-bit word from memory as four 8-bit data words.
- For EPPI_CTL.DLEN values greater than 8 bits but less than or equal to 16 bits, the EPPI transmits one 32-bit word from memory as two 16-bit data words.
- When EPPI_CTL.DLEN=18 or 24, the EPPI transmits three 32-bit words from memory as four data words.

Sign-Extension and Zero-Filling

The following list describes the bit settings and functionality for sign-extension and zero-fill.

- For EPPI_CTL.DLEN equal to 10, 12 or 14, data is zero-filled or sign-extended to 16 bits.
- For EPPI_CTL.DLEN equal to 18 bits, data is zero-filled or sign-extended to 24 bits if packing is enabled, and zero-filled or sign-extended to 32 bits if packing is disabled.
- For EPPI_CTL.DLEN equal to 24 bits, data is zero-filled or sign-extended to 32 bits if packing is disabled.
- For EPPI_CTL.DLEN equal to 8 bits, data is zero-filled or sign-extended to 16 bits if packing is disabled.
- If EPPI_CTL.SIGNEXT=1, then the data is sign-extended, otherwise it is zero-filled.

Split Receive Modes

The control register has three control bits for split receive modes: EPPI_CTL.SPLTEO, EPPI_CTL.SUBSPLTODD and EPPI_CTL.DMACFG. Packing is not valid in split modes.

- If EPPI_CTL.SPLTEO=1, the EPPI splits the incoming data stream into two sub-streams, an even stream and an odd stream, and packs them separately.
- The EPPI_CTL.SUBSPLTODD bit is available only when EPPI_CTL.SPLTEO=1. When EPPI_CTL.SUBSPLTODD=1, the EPPI sub-splits the odd sub-stream, and packs the streams separately.
- The EPPI_CTL.DMACFG bit is also available only if EPPI_CTL.SPLTEO=1. If EPPI_CTL.DMACFG=1, the EPPI uses two DMA channels and if EPPI_CTL.DMACFG=0, the EPPI uses only one DMA channel.

Split Transmit Modes

The EPPI_CTL register has three control bits for split transmit modes: EPPI_CTL.SPLTEO, EPPI_CTL.SUBSPLTODD and EPPI_CTL.DMACFG. The DMA is always a 32-bit DMA. Packing is not valid in split modes.

- If EPPI_CTL.SPLTEO=1 the EPPI receives the Luma (Y3Y2Y1Y0) and interleaved Chroma (Cr1Cb1Cr0Cb0) data as 32 bits from the DMA channel and interleaves the data to form a 4:2:2 YCrCb data stream to transmit.
- The EPPI_CTL.SUBSPLTODD bit is available only when EPPI_CTL.SPLTEO=1. In this case if EPPI_CTL.SUBSPLTODD=1, the EPPI receives the Luma (Y3Y2Y1Y0) and de-interleaved Chroma (Cb3Cb2Cb1Cb0 and Cr3Cr2Cr1Cr0) and interleaves the data to form a 4:2:2 YCrCb data stream to transmit. (Note that the EPPI does not decimate the chroma data when formatting it into 4:2:2).
- The EPPI_CTL.DMACFG bit is also valid only if EPPI_CTL.SPLTEO=1 If EPPI_CTL.DMACFG=1, the EPPI uses two DMA channels and if EPPI_CTL.DMACFG=0, the EPPI uses only one DMA channel.

Clock Gating

In ITU-R BT.656 and GP 0/1/2 FS modes, PPI_FS3 becomes a clock-gating input. This is valid for both internally and externally sourced PPI_CLK, in both receive and transmit modes. This clock gating signal must be synchronous with PPI_CLK and must be driven by the external device on the rising edge of PPI_CLK. Its function is to hold the sync and data lines in their current state until PPI_FS3 is driven low. There are no additional latency cycles upon coming out of clock gating mode.

If clock gating is not required, the PPI_FS3 pin must either be tied to ground, or configured to operate as another of its multiplexed functions.

In GP 2 FS transmit mode with internally generated frame syncs, the PPI_FS3 pin functions as a data enable signal.

Support for Delayed Start of EPPI Frame Syncs

The EPPI supports a delayed start of the PPI_FS1 and PPI_FS2frame syncs. The EPPI_FS1_DLYand EPPI_FS2_DLYregisters are programmable registers corresponding to PPI_FS1(HSYNC) and PPI_FS2(VSYNC). These registers are described below.

The first active edge of the internally generated frame sync is delayed by the amount programmed in these registers starting from the first PPI_CLKedge. The delay counter (which is the period counter itself, since they don't run together) runs only for the first time and then shuts off until the EPPI is re-enabled. The delay registers should be programmed prior to the first PPI_CLK edge (similar to the width and period registers). The figure below shows the functioning of PPI_FS1 and PPI_FS2.

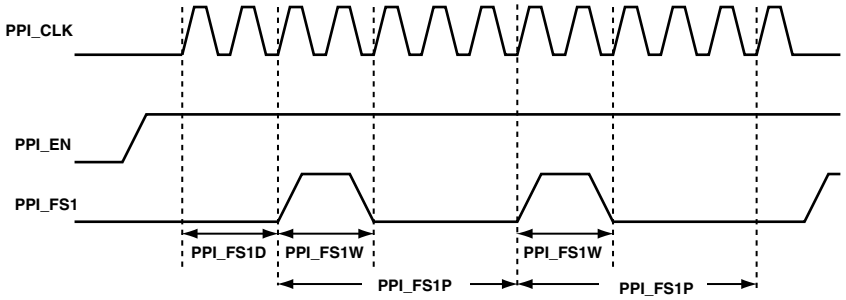


Figure 31-14: EPPI Delayed Frame Sync Generation

Ignoring Premature External Frame Syncs for Data Consistency

Once a frame has started with a VSYNC followed by a HSYNC (or both coming together), a line is tracked. When the count expires, the state machine waits at the end of line for a HSYNC to come. With the arrival of the HSYNC, the state machine starts tracking the next line and so on.

The number of lines tracked is counted separately and once the end of a frame is reached, the state machine waits there for the next VSYNC/HSYNC combination and the next frame starts once they are sampled. The problem with this scheme is that every incoming FS (VSYNC or HSYNC) resets the respective counters and the tracking starts all over (even if the FS signals are premature). This results in incomplete data (or frames) to enter into memory through the PxP interface.

To correct this problem, the EPPI waits for a frame/line completion before considering any incoming FS as valid as described below.

- Single FS mode and line tracking in dual FS mode – When a line is in progress, if HSYNC is detected prematurely, it is ignored. A line track underflow event is generated.
- Dual FS mode – If a VSYNC is received when a frame is in progress, it is ignored. A frame track underflow error (EPPI_STAT.FTERRUNDR) is generated.

Ignoring the FS ensures that once a frame starts, the amount of data that goes into the memory/PxP interface exactly corresponds to the programmed data size in a frame.

NOTE: Even if the premature FS is a valid FS, the state machine loses at most one frame and it recovers in the subsequent FS. At all times the FS to number of data going into the memory relationship is maintained as programmed.

When data underflow errors occur at the DMA interface, the EPPI does the following.

- If a premature line sync is detected, a LT underflow error is generated (EPPI_STAT.LTERRUNDR=1). All further line track errors are ignored until a next valid line sync is detected by the EPPI.
- If a premature frame sync is detected, a FT underflow error is generated(EPPI_STAT.FTERRUNDR=1). All further frame track and line track errors are ignored until a next valid frame sync is detected by the EPPI.

EPPI Event Control

The following sections describe how events are used in EPPI management.

EPPI Status, Error and Interrupt Signals

The EPPI generates error interrupts (flagged in the EPPI_STAT register) if any one of the following error conditions occur.

- EPPI_STAT.YFIFOERR (YFIFO underflow or overflow)
- EPPI_STAT.CFIFOERR (CFIFO underflow or overflow)
- EPPI_STAT.LTERROVR (line track overflow error)
- EPPI_STAT.LTERRUNDR (line track underflow error)
- EPPI_STAT.FTERROVR (frame track overflow error)
- EPPI_STAT.FTERRUNDR (frame track underflow error)
- EPPI_STAT.ERRNCOR (ITU preamble error not corrected)

These conditions are cleared by a W1C (write-1-to-clear) operation. Each of the individual conditions which cause an EPPI error interrupt can be masked. The interrupt mask register (EPPI_IMSK) allows the masking of individual conditions which cause error interrupts.

There is only one interrupt line from each EPPI so all interrupts are internally OR'ed and sent as a single interrupt to the core. The EPPI_STAT register must then be read to discover specific errors. These errors are described in detail in the following sections.

PxP Errors

The following are PxP generated errors.

Receive mode – The receive data sent to the PxP is stalled by the destination in EPPI receive. In this situation the data is lost because the EPPI cannot hold off the incoming data. This condition is reported in the EPPI_STAT.PXPERR bit. This bit is cleared by a W1C operation.

Transmit mode – The transmit data coming from the PxP interface goes through the FIFO. Data underflow errors are reported as EPPI_STAT.YFIFOERR if the FIFO underflows similar to the DMA transmit mode.

Frame and Line Track Errors

When functioning in external frame sync mode, the line track error (EPPI_STAT.LTERROVR and EPPI_STAT.LTERRUNDR) and frame track error (EPPI_STAT.FTERROVR and EPPI_STAT.FTERRUNDR) status bits are used to keep track of the line and frame synchronization errors. They are updated when there is a mismatch detected in the HSYNC and VSYNC as compared to the programmed values in EPPI_LINE and EPPI_FRAMEcount registers.

Line Track Errors

The line track overflow (EPPI_STAT.LTERROVR) and underflow errors (EPPI_STAT.LTERRUNDR) generate a maskable interrupt as soon as they are identified and not at the next frame sync.

- If the frame sync has not arrived when the EPPI_LINE counter expires, then the EPPI_STAT.LTERROVR error is generated.
- When the EPPI_LINE counter is running and a frame sync is detected, the EPPI_STAT.LTERRUNDR error is generated. Both interrupts are cleared by a W1C operation.

Frame Track Errors

The frame track overflow (EPPI_STAT.FTERROVR) and underflow errors (EPPI_STAT.FTERRUNDR) generate a maskable interrupt as soon as they are identified. In the case of the EPPI_STAT.FTERROVR error, when the EPPI_FRAME.VALUE counter expires, the error is reported before the next FS arrives.

When the EPPI_FRAME counter is running, if an FS is detected, then an EPPI_STAT.FTERRUNDR is reported.

Both errors generate an error interrupt and should be cleared by a W1C operation at their respective locations in the status register.

A premature frame sync results in a frame track under run error but the error is logged (register bit set) only after the subsequent blanking period (if any) elapses.

Preamble Error Not Corrected Error

The EPPI supports data embedded frame syncs in ITU and SMPTE formats. In these formats, the module can receive an erroneous preamble which is not correctable. The EPPI_STAT.ERRNCOR error signals when this event occurs.

EPPI Programming Model

The following sections describe programming techniques, including receiving/transmitting ITU-R 656 frames; configuring transfers in GP0, GP1, GP2, and GP3 modes; and managing EPPI mode configurations.

Receiving ITU-R 656 Frames

The EPPI supports the reception of ITU-R 656 compliant frames.

1. Configure the EPPI to receive either full ITU-R 656 frame, active video or blanking information by configuring the `EPPI_CTL.XFRTYPE` bits.
2. In both active video mode and in VBI (vertical blanking information) mode, specify the number of total (active plus vertical blanking) lines per frame in the `EPPI_FRAME` register, and the number of total (active plus horizontal blanking plus 8) samples per line in the `EPPI_LINE` register.
3. Configure DMA descriptors to move the data to memory.
4. Enable DMA.
5. Enable the EPPI.

RESULT:

Depending on how the EPPI is configured, either the full ITU-R 656 frame is moved to memory or just the active video or just the blanking information.

Transmitting ITU-R 656 Frames in GP Transmit Modes

The EPPI can take active video from memory and generate the proper preambles and blanking information to produce valid ITU-R 656 video frames for transmission.

1. Provide active data frame in memory.
2. Set the `EPPI_CTL.BLANKGEN` bit so the EPPI generates blanking information.
3. Configure the `EPPI_FS1_WLHB`, `EPPI_FS1_PASPL`, `EPPI_FS2_WLVB`, `EPPI_FS2_PALPF` registers accordingly.
4. Configure the rest of the EPPI settings.
5. Configure DMA to fetch active frame data from memory buffers.

6. Enable DMA.
7. Enable the EPPI.

RESULT:

The EPPI takes the active data from memory, generates the blanking information and transmits an ITU-R 656 frame

Configuring Transfers in GP 0 FS Mode

The EPPI can be configured so no periodic frame syncs are used to frame the data.

1. Configure the EPPI to operate in GP 0 FS mode by setting `XFRTYPE` = b#11 and `FSCFG` = b#00 in the `EPPI_CTL` register.
2. When receiving, configure the EPPI to trigger on internally or externally by setting the `FLDSEL` field appropriately in the `EPPI_CTL` register. When transmitting, the trigger will always be internally generated.
3. Configure DMA to move the data to or from memory.
4. Enable DMA.
5. Enable EPPI.

RESULT:

The data amount of data transferred will be controlled by the DMA descriptions and not by any frame syncs from the EPPI.

Configuring Transfers in GP 1 FS Mode

The GP 1 FS mode is useful for interfacing the EPPI with analog-to-digital converters (ADCs), digital-to-analog converters (DACs) and other general-purpose devices. This mode works for both transmit and receive.

NOTE: The `EPPI_FRAME`, `EPPI_VDLY` and `EPPI_VCNT` registers have no effect in GP 1 FS mode. As a result, frame track errors and vertical windowing are not possible in this mode.

1. Configure GP 1 FS mode by setting the `EPPI_CTL.XFRTYPE` bit=b#11 and the `EPPI_CTL.FSCFG` bit=b#01. The frame syncs may be provided by an external device or they can be sourced by the EPPI itself.

2. Program the EPPI_LINE register to contain the number clock cycles expected between two assertions of the PPI_FS1 signal to keep track of line track errors. The EPPI_LINE register must be programmed before the EPPI_HCNT register.
3. Program the EPPI_HDLY register to contain the number of clock cycles to wait after the assertion of PPI_FS1, for example the start of frame.
4. Program the EPPI_HCNT register to contain the number of data samples to receive or transmit for each frame.
5. Configure DMA to move the data to or from memory.
6. Enable DMA.
7. Enable the EPPI.

RESULT:

Data moves in or out of memory and is framed by a frame sync for every line.

Configuring Transfers in GP 2 FS Mode

GP 2 FS mode is useful for video applications that use two hardware synchronization signals, HSYNC and VSYNC. The HSYNC can be connected to the PPI_FS1signal and VSYNC can be connected to the PPI_FS2signal.

1. Configure the EPPI to operate in GP 0 FS mode by setting the EPPI_CTL.XFRTYPE bit=b#11 and the EPPI_CTL.FSCFG bit=b#10. The frame syncs may be provided by an external device or they can be sourced by the EPPI itself.
2. Program the EPPI_FRAME register to contain the number of expected lines per frame. The value should be equal to the number of PPI_FS1 signal assertions expected between the start of each frame sync and is used to keep track of frame track errors. The EPPI_FRAME register must be programmed before the EPPI_VCNT register.
3. Program the EPPI_LINE register to contain the number of clock cycles expected between two assertions of the PPI_FS1 signal to keep track of line track errors. The EPPI_LINE register must be programmed before the EPPI_HCNT register.
4. Program the EPPI_HDLY register to configure the number of clock cycles to wait after the assertion of the PPI_FS1 signal, (for example the start of the line).
5. Program the EPPI_HCNT register to contain the number of data samples to receive or transmit for each line.
6. Program the EPPI_VDLY register to contain the number of lines to wait after the start of frame is detected.

7. Program the EPPI_VCNT register to contain the number of lines to receive or transmit.
8. If setting up the EPPI for transmit, the data enable (DEN) pin behaves according to the enabling of the blanking generation and the data length setting (DLEN). More detail can be found at *Data Enable in General-Purpose 2 Frame Sync Transmit Mode*.
9. Enable DMA.
10. Enable the EPPI.

RESULT:

Data is moved in or out of memory and framed by a frame sync for every line and frame.

Configuring Transfers in GP 3 FS Mode

GP 3 FS mode is useful for video applications that use three hardware synchronization signals, HSYNC, VSYNC, and FIELD. The HSYNC can be connected to PPI_FS1, VSYNC can be connected to PPI_FS2, and FIELD can be connected to PPI_FS3.

1. Configure the EPPI to operate in GP 3 FS mode by setting the EPPI_CTL.XFRTYPE bit=b#11 and the EPPI_CTL.FSCFG bit=b#11. The frame syncs may be provided by an external device or they can be sourced by the EPPI itself.
2. Configure the windowing registers according to steps in GP 2 FS mode.
3. Enable DMA.
4. Enable the EPPI.

RESULT:

Data is moved in or out of memory and is framed by a frame sync for every line and frame. Operation and result is similar to operation in GP 2 FS mode but the PPI_FS3 signal is also used.

Configuring the EPPI to Use the Windowing Feature

Windowing is a useful feature for applications where the region of interest is smaller than the active video stream (for example, sensor calibration, auto-focusing, and others). It can result in significant DMA bandwidth reduction. The EPPI supports windowing for GP Input modes.

1. Program the EPPI_FRAME register with the number of lines the frame contains.
2. Program the EPPI_LINE register with the number of samples per line in the frame.
3. Program the EPPI_VDLY register with the number of lines to wait after the start of a new frame before starting to read/transmit data.

4. Program the EPPI_VCNT register with the number of lines to read in or write out after EPPI_VDLY number of lines from the start of the frame.
5. Program the EPPI_HDLY register with the number of clock cycles to delay after the assertion of PPI_FS1 is detected for the start of a new line.
6. Program the EPPI_HCNT register with the number of samples to read in or write out after EPPI_HDLY number of cycles have expired since the assertion of PPI_FS1.

EPPI Mode Configuration

This section describes EPPI mode configurations, including support for all EPPI transmit and receive modes.

Configuring 8-Bit Receive Mode

For 8-bit non-split receive mode, if EPPI_CTL.PACKEN=1, the EPPI packs four bytes of incoming data into a 32-bit word. Alternate even or odd samples may be skipped based on the EPPI_CTL.SKIPEN and EPPI_CTL.SKIPEO bits. The first incoming data can be placed either in the least significant bit positions or in the most significant bit positions, based on the EPPI_CTL.SWAPEN bit setting.

Table 31-14: 8-Bit Receive Mode with Packing Enabled

Pin Data (8 bits)	DMA DATA SKIPEN=0 SKIPEO =X SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=1 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=1 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=1 SIGNEXT=X
0x11						
0x22						
0x33						
0x44	0x4433 2211	0x1122 3344				
0x55						
0x66						
0x77			0x7755 3311		0x1133 5577	
0x88	0x8877 6655	0x5566 7788		0x8866 4422		0x2244 6688
0x99						
0xAA						
0xBB						
0xCC	0xCCBB AA99	0x99AA BBCC				

Table 31-14: 8-Bit Receive Mode with Packing Enabled (Continued)

Pin Data (8 bits)	DMA DATA SKIPEN=0 SKIPEO =X SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=1 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=1 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=1 SIGNEXT=X
0xDD						
0xEE						
0xFF			0xFFDD BB99		0x99BB DDFF	
0x00	0x00FF EEDD	0xDDE EFF00		0x00EE CCAA		0xAACC EE00

If EPPI_CTL.PACKEN=0, the DMA is a 16-bit DMA and the EPPI either sign-extends or zero-fills the bytes of incoming data into a 16-bit word. The EPPI_CTL.SWAPEN bit has no effect if EPPI_CTL.PACKEN=0.

Table 31-15: 8-Bit Receive Mode with Packing Disabled

Pin Data (8 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=1	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=X SIGNEXT=1
0x44	0x0044	0x0044	0x0044	
0x55	0x0055	0x0055		0x0055
0x66	0x0066	0x0066	0x0066	
0x77	0x0077	0x0077		0x0077
0x88	0x0088	0xFF88	0x0088	
0x99	0x0099	0xFF99		0xFF99
0xAA	0x00AA	0xFFAA	0x00AA	
0xBB	0x00BB	0xFFBB		0xFFBB

Configuring 10/12/14-Bit Receive Modes

For 10, 12, or 14-bit non-split receive modes, the EPPI first either zero-fills or sign-extends the incoming data (depending on the setting of the EPPI_CTL.SIGNEXT bit) into a 16-bit word. If EPPI_CTL.PACKEN=1, the EPPI then packs two of these words into one 32-bit word. Alternate even or odd samples may be skipped based on the EPPI_CTL.SKIPEN and EPPI_CTL.SKIPEO bits. The first incoming data can be placed either in the least significant bit positions or in the most significant bit positions, based on the EPPI_CTL.SWAPEN bit setting.

Table 31-16: 10-Bit Receive Mode with Sign Extension, with Packing Enabled

Pin Data (10 bits)	MSB	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=0 SIGNEXT=1	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=1 SIGNEXT=1	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=0 SIGNEXT=1
0x111	0			
0x222	1	0xFE22 0111	0x0111 FE22	
0x333	1			0xFF33 0111
0x044	0	0x0044 FF33	0xff33 0044	
0x155	0			
0x266	1	0xFE66 0155	0x0155 FE66	
0x377	1			0xFF77 0155
0x088	0	0x0088 FF77	0xFF77 0088	

Table 31-17: 10-Bit Receive Mode with Sign Extension, with Packing Enabled

Pin Data (10 bits)	MSB	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=0 SIGNEXT=1	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=1 SIGNEXT=1	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=1 SIGNEXT=1
0x111	0			
0x222	1			
0x333	1		0x0011 FF33	
0x044	0	0x0044 FE22		0xFE22 0044
0x155	0			
0x266	1			
0x377	1		0x0155 FF77	
0x088	0	0x0088 FE66		0xFE66 0088

Table 31-18: 10-Bit Receive Mode, with Zero-Fill, with Packing Enabled

Pin Data (10 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=0 SIGNEXT=0	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=1 SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=0 SIGNEXT=0	DMA DATA SKIP_EN=1 SKIP_EO=0 SWAPEN=0 SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=1 SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=1 SIGNEXT=0
0x111						

Table 31-18: 10-Bit Receive Mode, with Zero-Fill, with Packing Enabled (Continued)

Pin Data (10 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=0 SIGNEXT=0	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=1 SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=0 SIGNEXT=0	DMA DATA SKIP_EN=1 SKIP_EO=0 SWAPEN=0 SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=1 SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=1 SIGNEXT=0
0x222	0x0222 0111	0x0111 0222				
0x333			0x0333 0111		0x0011 0333	
0x044	0x0044 0333	0x0333 0044		0x0044 0222		0x0222 0044
0x155						
0x266	0x0266 0155	0x0155 0266				
0x377			0x0377 0155		0x0155 0377	
0x088	0x0088 0377	0x0377 0088		0x0088 0266		0x0266 0088

The following table shows a 10-bit receive mode example when EPPI_CTL.PACKEN=0:

Table 31-19: 10-bit Receive Mode with Packing Disabled

Pin Data (10 bits)	MSB	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=1	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=X SIGNEXT=1	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=X SIGNEXT=0
0x111	0	0x0111	0x0111	0x0111	
0x222	1	0xFE22	0x0222		0x0222
0x333	1	0xFF33	0x0333	0xFF33	
0x044	0	0x0044	0x0444		0x0444
0x155	0	0x0155	0x0155	0x0155	
0x266	1	0xFE66	0x0266		0x0266
0x377	1	0xFF77	0x0377	0xFF77	
0x088	0	0x0088	0x0088		0x088

Configuring 16-Bit Receive Mode

For 16-bit non-split receive mode, if EPPI_CTL.PACKEN=1, the EPPI packs two 16-bit incoming data into one 32-bit word. Alternate even or odd samples may be skipped based on the EPPI_CTL.SKIPEN and EPPI_CTL.SKIPEO bits. The first incoming data can be placed either in the least significant bit positions or in the most significant bit positions, based on the EPPI_CTL.SWAPEN bit setting.

Table 31-20: 16-Bit Receive Mode with Packing Enabled

Pin Data (16 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=1 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=1 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=1 SIGNEXT=X
0x1111						
0x2222	0x2222 1111	0x1111 2222				
0x3333			0x3333 1111		0x1111 3333	
0x4444	0x4444 3333	0x3333 4444		0x4444 2222		0x2222 4444
0x5555						
0x6666	0x6666 5555	0x5555 6666				
0x7777			0x7777 5555		0x5555 7777	
0x8888	0x8888 7777	0x7777 8888		0x8888 6666		0x6666 8888

Table 31-21: 16-bit Receive Mode with Packing Disabled

Pin Data (16 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=X SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=X SIGNEXT=X
0x1111	0x1111	0x1111	
0x2222	0x2222		0x2222
0x3333	0x3333	0x3333	
0x4444	0x4444		0x4444
0x5555	0x5555	0x5555	
0x6666	0x6666		0x6666
0x7777	0x7777	0x7777	
0x8888	0x8888		0x8888

Configuring 18-Bit Receive Mode

For 18-bit non-split receive mode, if `EPPI_CTL.PACKEN`=0, the EPPI zero-fills or sign-extends the incoming data into a 32-bit word. If `EPPI_CTL.PACKEN`=1, the EPPI first zero-fills or sign-extends the incoming data to 24 bits, and then packs four such 24-bit data words into three 32-bit words. Alternate even or odd samples may be skipped based on the `EPPI_CTL.SKIPEN` and `EPPI_CTL.SKIPEO` bits. The `EPPI_CTL.SWAPEN` bit has no effect.

Table 31-22: 18-bit Receive Mode with Packing Disabled

Pin Data (18 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=X SIGNEXT=0
0x0 6666	0x0000 6666	0x0000 6666	
0x1 7777	0x0001 7777		0x0001 7777
0x2 8888	0x0002 8888	0x0002 8888	
0x3 9999	0x0003 9999		0x0003 9999

Table 31-23: 18-bit Receive Mode with Packing Enabled

Pin Data (18 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=X SIGNEXT=0
0x0 1122			
0x1 3344	0x4400 1122		
0x2 5566	0x5566 0133	0x6600 1122	
0x3 7788	0x0377 8802		0x8801 3344
0x0 99AA		0x99AA 0255	
0x1 BBCC	0xCC00 99AA		0xBBCC 0377
0x2 DDEE	0xDDEE 01BB	0x02DD EE00	
0x3 FF12	0x03FF 122D		0x03FF 1201

Configuring 24-Bit Receive Mode

For 24-bit non-split receive mode, if EPPI_CTL.PACKEN=0, the EPPI zero-fills or sign-extends the incoming data into a 32-bit word. If EPPI_CTL.PACKEN=1, the EPPI packs four incoming 24-bit data words into three 32-bit words. Alternate even or odd samples may be skipped based on the EPPI_CTL.SKIPEN and EPPI_CTL.SKIPEO bits. The EPPI_CTL.SWAPEN bit has no effect.

Table 31-24: 24-bit Receive Mode with Packing Disabled

Pin Data(24 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=X SIGNEXT=0
0x6 6666	0x0066 6666	0x0066 6666	
0x7 7777	0x0077 7777		0x0077 7777
0x8 8888	0x0088 8888	0x0088 8888	
0x9 9999	0x0099 9999		0x0099 9999

Table 31-25: 24-bit Receive Mode with Packing Enabled

Pin Data (24 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=X
0x11 2233			
0x44 5566	0x6611 2233		
0x77 8899	0x8899 4455	0x9911 2233	
0x00 AABB	0x00AA BB77		0xBB44 5566
0xCC DDEE		0xDDEE 7788	
0xFF 1234	0x34CC DDEE		0x1234 00AA
0x56 7890	0x7890 FF12	0x5678 90CC	
0xAB CDEF	0xABCD EF56		0xABCD EFFF

Configuring 8-Bit Split Receive Mode

For 8-bit split receive mode, the `EPPI_CTL.PACKEN` and `EPPI_CTL.SIGNEXT` bits are not valid. The EPPI always packs four bytes of data into one 32-bit word.

Table 31-26: 8-bit Split Receive Mode with SKIPEN = 0 and SWAPEN = 0

Pin Data (8 bits)	SPLTEO=1 SUBSPLTODD= 0 SWAPEN=0 SKIPEN=0 SKIPEO=X			SPLTEO=1 SUBSPLTODD= 1 SWAPEN=0 SKIPEN=0 SKIPEO=X		
	DMACFG=1		DMACFG=0	DMACFG=1		DMACFG=0
	Primary DMA Channel	Secondary DMA Channel	Primary DMA Channel	Primary DMA Channel	Secondary DMA Channel	Primary DMA Channel
V_0						
Y_0						
U_0						
Y_1						
V_1						
Y_2						
U_1		$U_1V_1U_0V_0$	$U_1V_1U_0V_0$			
Y_3	$Y_3Y_2Y_1Y_0$		$Y_3Y_2Y_1Y_0$	$Y_3Y_2Y_1Y_0$		$Y_3Y_2Y_1Y_0$
V_2						
Y_4						
U_2						
Y_5						
V_3					$V_3V_2V_1V_0$	$V_3V_2V_1V_0$
Y_6						
U_3		$U_3V_3U_2V_2$	$U_3V_3U_2V_2$		$U_3U_2U_1U_0$	
Y_7	$Y_7Y_6Y_5Y_4$		$Y_7Y_6Y_5Y_4$	$Y_7Y_6Y_5Y_4$		$Y_7Y_6Y_5Y_4$
V_4						$U_3U_2U_1U_0$

Table 31-27: 8-bit Split Receive Mode with SKIPEN = 0 and SWAPEN = 1

Pin Data (8 bits)	SPLTEO=1 SUBSPLTODD=0 SWAPEN=1 SKIPEN=0 SKIPEO=X			SPLTEO=1 SUBSPLTODD=1 SWAPEN=1 SKIPEN=0 SKIPEO=X		
	DMACFG=1		DMACFG=0	DMACFG=1		DMACFG=0
	PRIMARY DMA CHANNEL	SECONDARY DMA CHANNEL	PRIMARY DMA CHANNEL	PRIMARY DMA CHANNEL	SECONDARY DMA CHANNEL	PRIMARY DMA CHANNEL
V_0						
Y_0						
U_0						
Y_1						
V_1						
Y_2						
U_1		$V_0U_0V_1U_1$	$V_0U_0V_1U_1$			
Y_3	$Y_0Y_1Y_2Y_3$		$Y_0Y_1Y_2Y_3$	$Y_0Y_1Y_2Y_3$		$Y_0Y_1Y_2Y_3$
V_2						
Y_4						
U_2						
Y_5						
V_3					$V_0V_1V_2V_3$	$V_0V_1V_2V_3$
Y_6						
U_3		$V_2U_2V_3U_3$	$V_2U_2V_3U_3$		$U_0U_1U_2U_3$	
Y_7	$Y_4Y_5Y_6Y_7$		$Y_4Y_5Y_6Y_7$	$Y_4Y_5Y_6Y_7$		$Y_4Y_5Y_6Y_7$
V_4						$U_0U_1U_2U_3$

When the bits settings are `EPPI_CTL.SPLTEO`=1, `EPPI_CTL.SUBSPLTODD`=1 and `EPPI_CTL.DMACFG`=0, note that although the second Chroma component (U0U1U2U3 in the tables above) sent over the DMA bus is completely packed before the Luma component (Y4Y5Y6Y7 in the tables above), it is intentionally held until that previous word is moved out. This allows the separation of Luma and Chroma values into individual buffers when using 2D-DMA.

Configuring 10/12/14/16-Bit Split Receive Mode with SPLTWRD=0

For 16-bit split receive mode, the `EPPI_CTL.PACKEN` bit is not valid. The EPPI always packs two 16-bit words into one 32-bit word. For 10, 12, or 14-bit split receive modes, the EPPI first either sign-extends or zero-fills the incoming data into a 16-bit word, and then packs two of these words into one 32-bit word to send to the DMA.

Table 31-28: 16-bit Split Receive Mode with SPLTWRD = 0, SKIPEN = 0 and SWAPEN = 0

	SPLTEO=1 SUBSPLTODD=0 SWAPEN=0 SKIPEN=0 SKIPEO=X			SPLTEO=1 SUBSPLTODD=1 SWAPEN=0 SKIPEN=0 SKIPEO=X		
	DMACFG=1		DMACFG=0	DMACFG=1		DMACFG=0
Pin Data (16 bits)	Primary DMA Channel	Secondary DMA Channel	Primary DMA Channel	Primary DMA Channel	Secondary DMA Channel	Primary DMA Channel
V_0						
Y_0						
U_0		U_0V_0	U_0V_0			
Y_1	Y_1Y_0		Y_1Y_0	Y_1Y_0		Y_1Y_0
V_1					V_1V_0	V_1V_0
Y_2						
U_1		U_1V_1	U_1V_1		U_1U_0	
Y_3	Y_3Y_2		Y_3Y_2	Y_3Y_2		Y_3Y_2
V_2						U_1U_0

Table 31-29: 16-bit Split Receive Mode with SPLTWRD = 0, SKIPEN = 0 and SWAPEN = 1

	SPLTEO=1 SUBSPLTODD=0 SWAPEN=1 SKIPEN=0 SKIPEO=X			SPLTEO=1 SUBSPLTODD=1 SWAPEN=1 SKIPEN=0 SKIPEO=X		
	DMACFG=1		DMACFG=0	DMACFG=1		DMACFG=0
Pin Data (16 bits)	PRIMARY DMA CHANNEL	SECONDARY DMA CHANNEL	PRIMARY DMA CHANNEL	PRIMARY DMA CHANNEL	SECONDARY DMA CHANNEL	PRIMARY DMA CHANNEL
V_0						

Table 31-29: 16-bit Split Receive Mode with SPLTWRD = 0, SKIPEN = 0 and SWAPEN = 1 (Continued)

Pin Data (16 bits)	SPLTEO=1 SUBSPLTODD=0 SWAPEN=1 SKIPEN=0 SKIPEO=X			SPLTEO=1 SUBSPLTODD=1 SWAPEN=1 SKIPEN=0 SKIPEO=X		
	DMACFG=1		DMACFG=0	DMACFG=1		DMACFG=0
	PRIMARY DMA CHANNEL	SECONDARY DMA CHANNEL	PRIMARY DMA CHANNEL	PRIMARY DMA CHANNEL	SECONDARY DMA CHANNEL	PRIMARY DMA CHANNEL
Y_0						
U_0		V_0U_0	V_0U_0			
Y_1	Y_0Y_1		Y_0Y_1	Y_0Y_1		Y_0Y_1
V_1					V_0V_1	V_0V_1
Y_2						
U_1		V_1U_1	V_1U_1		U_0U_1	
Y_3	Y_2Y_3		Y_2Y_3	Y_2Y_3		Y_2Y_3
V_2						U_0U_1

Configuring 16-Bit Split Receive Mode with SPLTWRD=1

For 16-bit split receive mode, the `EPPI_CTL.PACKEN` bit is not valid. The EPPI always packs two 16-bit words into one 32-bit word. The `EPPI_CTL.SPLTWRD` bit is only valid when the `EPPI_CTL.DLEN`bit=16 bits.

Table 31-30: 16-bit Split Receive Mode with SPLTWRD = 1, SKIPEN = 0 and SWAPEN = 0

Pin Data (16 bits)	SPLTEO=1 SUBSPLTODD=0 SWAPEN=0 SKIPEN=0 SKIPEO=X			SPLT_EVEN_ODD=1 SUBSPLTODD=1 SWAPEN=0 SKIPEN=0 SKIPEO=X		
	DMACFG=1		DMACFG=0	DMACFG=1		DMACFG=0
	Primary DMA Channel	Secondary DMA Channel	Primary DMA Channel	Primary DMA Channel	Secondary DMA Channel	Primary DMA Channel
V_0Y_0						
U_0Y_1						
V_1Y_2						
U_1Y_3	$Y_3Y_2Y_1Y_0$	$U_1V_1U_0V_0$	$Y_3Y_2Y_1Y_0$	$Y_3Y_2Y_1Y_0$		$Y_3Y_2Y_1Y_0$

Table 31-30: 16-bit Split Receive Mode with SPLTWRD = 1, SKIPEN = 0 and SWAPEN = 0 (Continued)

Pin Data (16 bits)	SPLTEO=1 SUBSPLTODD=0 SWAPEN=0 SKIPEN=0 SKIPEO=X			SPLT_EVEN_ODD=1 SUBSPLTODD=1 SWAPEN=0 SKIPEN=0 SKIPEO=X		
	DMACFG=1		DMACFG=0	DMACFG=1		DMACFG=0
	Primary DMA Channel	Secondary DMA Channel	Primary DMA Channel	Primary DMA Channel	Secondary DMA Channel	Primary DMA Channel
V_2Y_4			$U_1V_1U_0V_0$			
U_2Y_5						
V_3Y_6					$V_3V_2V_1V_0$	$V_3V_2V_1V_0$
U_3Y_7	$Y_7Y_6Y_5Y_4$	$U_3V_3U_2V_2$	$Y_7Y_6Y_5Y_4$	$Y_7Y_6Y_5Y_4$	$U_3U_2U_1U_0$	$Y_7Y_6Y_5Y_4$
V_4Y_8			$U_3V_3U_2V_2$			$U_3U_2U_1U_0$

Configuring 8-Bit Transmit Mode

For 8-bit non-split transmit mode, if the EPPI_CTL.PACKEN bit=1, the DMA is a 32-bit DMA and the EPPI unpacks the 32-bit word from memory into four bytes to transmit. The EPPI transmits either the MSBs or the LSBs as the first data, depending on the EPPI_CTL.SWAPEN bit setting. If EPPI_CTL.PACKEN=0, the DMA is a 16-bit DMA and the EPPI transmits the lower 8 bits. The EPPI_CTL.SWAPEN bit has no effect when EPPI_CTL.PACKEN=0.

Table 31-31: 8-bit Transmit Mode with Packing Enabled

DMA Data (32 bits)	Pin Data when SWAPEN=0	Pin Data when SWAPEN=1
0x11223344	0x44	0x11
0x55667788	0x33	0x22
	0x22	0x33
	0x11	0x44
	0x88	0x55
	0x77	0x66
	0x66	0x77
	0x55	0x88

Table 31-32: Data Sent in 8-bit Transmit Mode with Packing Disabled

DMA Data (16 bits)	Pin Data SWAPEN=X
0x1234	0x34
0x2345	0x45
0x3456	0x56

Configuring 10/12/14-Bit Transmit Modes

For 10, 12, or 14-bit non-split transmit modes, if the EPPI_CTL.PACKEN bit=1, the DMA is a 32-bit DMA and the EPPI unpacks the 32-bit word from memory into two 16-bit data words, then transmits the required LSBs from each. The EPPI transmits either the most significant word or the least significant word as the first data, depending on the EPPI_CTL.SWAPEN bit setting. If EPPI_CTL.PACKEN=0, the DMA is a 16-bit DMA and the EPPI transmits the required LSBs. The EPPI_CTL.SWAPEN bit has no effect when the EPPI_CTL.PACKEN bit=0.

Table 31-33: 10-bit Transmit Mode with Packing Enabled

DMA Data (32 bits)	Pin Data when SWAPEN=0	Pin Data when SWAPEN=1
0x1111 2222	0x222	0x111
0x3333 4444	0x111	0x222
	0x044	0x333
	0x333	0x044

Table 31-34: 10-bit Transmit Mode with Packing Disabled

DMA Data (16 bits)	Pin Data SWAPEN=X
0x1234	0x234
0x2345	0x345
0x3456	0x056
0x4567	0x167

Configuring 16-Bit Transmit Mode

For 16-bit non-split transmit mode, if the EPPI_CTL.PACKEN bit=1, the DMA is a 32-bit DMA and the EPPI unpacks the 32-bit word from memory into two 16-bit data words to transmit. The EPPI transmits either the MSBs or the LSBs as the first data, depending on the EPPI_CTL.SWAPEN bit setting. If the EPPI_CTL.PACKEN bit=0, the DMA is a 16-bit DMA and the EPPI transmits the data as is. The EPPI_CTL.SWAPEN has no effect when EPPI_CTL.PACKEN bit=0.

Table 31-35: 16-bit Transmit Mode with Packing Enabled

DMA Data (32 bits)	Pin Data when SWAPEN=0	Pin Data when SWAPEN=1
0x1111 2222	0x2222	0x1111
0x3333 4444	0x1111	0x2222
	0x4444	0x3333
	0x3333	0x4444

Table 31-36: 16-bit Transmit Mode with Packing Disabled

DMA Data (16 bits)	Pin Data SWAPEN=X
0x1234	0x1234
0x2345	0x2345
0x3456	0x3456

Configuring 18-Bit Transmit Mode

For 18-bit transmit mode, if the `EPPI_CTL.PACKEN` bit=1, the DMA is a 32-bit DMA and the EPPI unpacks the 32-bit word from memory. In this mode, when `EPPI_CTL.RGBFMTEN` is set, the least significant two bits of R, G, and B are dropped.

Table 31-37: 18-bit Transmit Mode with Packing Enabled

DMA Data	Pin Data (18-bits)	
	RGBFMTEN=0	RGBFMTEN=1
0x0123 4567	0x3 4567	0x0 8459
0x89AB CDEF	0x1 EF01	0x3 3EC0
0x0123 4567	0x3 89AB	0x1 98AA
	0x1 2345	0x0 0211

Table 31-38: 18-bit Transmit Mode with Packing Disabled

DMA Data	Pin Data (18-bits)	
	RGBFMTEN=0	RGBFMTEN=1
0x0123 4567	0x3 4567	0x0 8459
0x89AB CDEF	0x3 CDEF	0x2 ACFB
0x0123 4567	0x3 4567	0x0 8459

Configuring 24-Bit Transmit Mode

For 24-bit transmit mode, if the EPPI_CTL.PACKEN bit=1, the DMA is a 32-bit DMA and the EPPI unpacks three 32-bit words from memory into four 24-bit words to transmit. The effect of the EPPI_CTL.SWAPEN bit setting is shown in the table below.

Table 31-39: 24-bit Transmit Mode

DMA Data (32 Bits)	Pin Data when SWAPEN=0	Pin Data when SWAPEN=1
$R_1B_0G_0R_0$	$B_0G_0R_0$	$R_0G_0B_0$
$G_2R_2B_1G_1$	$B_1G_1R_1$	$R_1G_1B_1$
$B_3G_3R_3B_2$	$B_2G_2R_2$	$R_2G_2B_2$
	$B_3G_3R_3$	$R_3G_3B_3$

Configuring 8-Bit Split Transmit Mode

For 8-bit split transmit mode, the EPPI_CTL.PACKEN bit is not valid. The EPPI always unpacks the 32-bit DMA data into four bytes to transmit.

Table 31-40: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=0 and SWAPEN=0

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	DMA1 DATA (32 bits)	Pin Data (8 bits)	DMA0 DATA (32 bits)	Pin Data (8 bits)
$Y_3Y_2Y_1Y_0$	$U_1V_1U_0V_0$	V_0	$U_1V_1U_0V_0$	V_0
$Y_7Y_6Y_5Y_4$	$U_3V_3U_2V_2$	Y_0	$Y_3Y_2Y_1Y_0$	Y_0
		U_0	$U_3V_3U_2V_2$	U_0
		Y_1	$Y_7Y_6Y_5Y_4$	Y_1
		V_1		V_1
		Y_2		Y_2
		U_1		U_1
		Y_3		Y_3
		V_2		V_2
		Y_4		Y_4
		U_2		U_2
		Y_5		Y_5
		V_3		V_3

Table 31-40: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=0 and SWAPEN=0 (Continued)

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	DMA1 DATA (32 bits)	Pin Data (8 bits)	DMA0 DATA (32 bits)	Pin Data (8 bits)
		Y_6		Y_6
		U_3		U_3
		Y_7		Y_7

Table 31-41: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=1 and SWAPEN=0

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	DMA1 DATA (32 bits)	Pin Data (8 bits)	DMA0 DATA (32 bits)	Pin Data (8 bits)
$Y_3Y_2Y_1Y_0$	$V_3V_2V_1V_0$	V_0	$V_3V_2V_1V_0$	V_0
$Y_7Y_6Y_5Y_4$	$U_3U_2U_1U_0$	Y_0	$Y_3Y_2Y_1Y_0$	Y_0
	$V_7V_6V_5V_4$	U_0	$U_3U_2U_1U_0$	U_0
	$U_7U_6U_5U_4$	Y_1	$Y_7Y_6Y_5Y_4$	Y_1
		V_1		V_1
		Y_2		Y_2
		U_1		U_1
		Y_3		Y_3
		V_2		V_2
		Y_4		Y_4
		U_2		U_2
		Y_5		Y_5
		V_3		V_3
		Y_6		Y_6
		U_3		U_3
		Y_7		Y_7

Table 31-42: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=0 and SWAPEN=1

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	DMA1 DATA (32 bits)	Pin Data (8 bits)	DMA0 DATA (32 bits)	Pin Data (8 bits)
$Y_3Y_2Y_1Y_0$	$U_1V_1U_0V_0$	U_1	$U_1V_1U_0V_0$	U_1

Table 31-42: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=0 and SWAPEN=1 (Continued)

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	DMA1 DATA (32 bits)	Pin Data (8 bits)	DMA0 DATA (32 bits)	Pin Data (8 bits)
$Y_7Y_6Y_5Y_4$	$U_3V_3U_2V_2$	Y_3	$Y_3Y_2Y_1Y_0$	Y_3
		V_1	$U_3V_3U_2V_2$	V_1
		Y_2	$Y_7Y_6Y_5Y_4$	Y_2
		U_0		U_0
		Y_1		Y_1
		V_0		V_0
		Y_0		Y_0
		U_3		U_3
		Y_7		Y_7
		V_3		V_3
		Y_6		Y_6
		U_2		U_2
		Y_5		Y_5
		V_2		V_3
		Y_4		Y_4

Table 31-43: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=1, and SWAPEN=1

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	DMA1 DATA (32 bits)	Pin Data (8 bits)	DMA0 DATA (32 bits)	Pin Data (8 bits)
$Y_3Y_2Y_1Y_0$	$V_3V_2V_1V_0$	V_3	$V_3V_2V_1V_0$	V_3
$Y_7Y_6Y_5Y_4$	$U_3U_2U_1U_0$	Y_3	$Y_3Y_2Y_1Y_0$	Y_3
	$V_7V_6V_5V_4$	U_3	$U_3V_3U_2V_2$	U_3
	$U_7U_6U_5U_4$	Y_2	$Y_7Y_6Y_5Y_4$	Y_2
		V_2		V_2
		Y_1		Y_1
		U_2		U_2
		Y_0		Y_0
		V_1		V_1

Table 31-43: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=1, and SWAPEN=1 (Continued)

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	DMA1 DATA (32 bits)	Pin Data (8 bits)	DMA0 DATA (32 bits)	Pin Data (8 bits)
		Y_7		Y_7
		U_1		U_1
		Y_6		Y_6
		V_0		V_0
		Y_5		Y_5
		U_0		U_0
		Y_4		Y_4

Configuring 10/12/14/16-Bit Transmit Mode with SPLTWRD=0

For 16-bit split transmit mode, the `EPPI_CTL.PACKEN` bit is not valid. The EPPI always unpacks the 32-bit DMA data into two 16-bit words to transmit. For 10, 12, or 14-bit split transmit modes, the EPPI first unpacks the data in the same way as for 16-bit transmit mode, but transmits only the required number of LSBs.

Table 31-44: 16-bit Split Transmit Mode with SPLTEO = 1, SUBSPLTODD = 0, and SWAPEN = 0

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	DMA1 DATA (32 bits)	Pin Data (16 bits)	DMA0 DATA (32 bits)	Pin Data (16 bits)
Y_1Y_0	U_0V_0	V_0	U_0V_0	V_0
Y_3Y_2	U_1V_1	Y_0	Y_1Y_0	Y_0
		U_0	U_1V_1	U_0
		Y_1	Y_3Y_2	Y_1
		V_1		V_1
		Y_2		Y_2
		U_1		U_1
		Y_3		Y_3

Table 31-45: 16-bit Split Transmit Mode with SPLTEO = 1, SUBSPLTODD = 1, and SWAPEN = 0

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	DMA1 DATA (32 bits)	Pin Data (16 bits)	DMA0 DATA (32 bits)	Pin Data (16 bits)
Y_1Y_0	V_1V_0	V_0	V_1V_0	V_0

Table 31-45: 16-bit Split Transmit Mode with SPLTEO = 1, SUBSPLTODD = 1, and SWAPEN = 0 (Continued)

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	DMA1 DATA (32 bits)	Pin Data (16 bits)	DMA0 DATA (32 bits)	Pin Data (16 bits)
Y_3Y_2	U_1U_0	Y_0	Y_1Y_0	Y_0
	V_3V_2	U_0	U_1U_0	U_0
	U_3U_2	Y_1	Y_3Y_2	Y_1
		V_1		V_1
		Y_2		Y_2
		U_1		U_1
		Y_3		Y_3

Table 31-46: 16-bit Split Transmit Mode with SPLTEO = 1, SUBSPLTODD = 0, and SWAPEN = 1

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	DMA1 DATA (32 bits)	Pin Data (16 bits)	DMA0 DATA (32 bits)	Pin Data (16 bits)
Y_1Y_0	V_0U_0	V_0	V_0U_0	V_0
Y_3Y_2	V_1U_1	Y_1	Y_1Y_0	Y_1
		U_0	V_1U_1	U_0
		Y_0	Y_3Y_2	Y_0
		V_1		V_1
		Y_3		Y_3
		U_1		U_1
		Y_2		Y_2

Table 31-47: 16-bit Split Transmit Mode with SPLTEO = 1, SUBSPLTODD = 1, and SWAPEN = 1

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	DMA1 DATA (32 bits)	Pin Data (16 bits)	DMA0 DATA (32 bits)	Pin Data (16 bits)
Y_1Y_0	V_1V_0	V_1	V_1V_0	V_1
Y_3Y_2	U_1U_0	Y_1	Y_1Y_0	Y_1
	V_3V_2	U_1	U_1U_0	U_1
	U_3U_2	Y_0		Y_0
		V_0		V_0

Table 31-47: 16-bit Split Transmit Mode with SPLTEO = 1, SUBSPLTODD = 1, and SWAPEN = 1 (Continued)

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	DMA1 DATA (32 bits)	Pin Data (16 bits)	DMA0 DATA (32 bits)	Pin Data (16 bits)
		Y_3		Y_1
		U_0		U_0
		Y_2		Y_2

Configuring 16-Bit Split Transmit Mode with SPLTWRD=1

For 16-bit split transmit mode, the EPPI_CTL.PACKEN bit is not valid. The EPPI always unpacks the 32-bit DMA data into two 16-bit words to transmit. The EPPI_CTL.SPLTWRD bit is only valid when the EPPI_CTL.DLENbit=16 bits.

Table 31-48: 16-bit Split Transmit Mode with SPLTWRD = 1, SUBSPLTODD = 0, and SWAPEN = 0

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	DMA1 DATA (32 bits)	Pin Data (16 bits)	DMA0 DATA (32 bits)	Pin Data (16 bits)
$Y_3Y_2Y_1Y_0$	$U_1V_1U_0V_0$	V_0Y_0	$U_1V_1U_0V_0$	V_0Y_0
$Y_7Y_6Y_5Y_4$	$U_3V_3U_2V_2$	U_0Y_1	$Y_3Y_2Y_1Y_0$	U_0Y_1
		V_1Y_2	$U_3V_3U_2V_2$	V_1Y_2
		U_1Y_3	$Y_7Y_6Y_5Y_4$	U_1Y_3
		V_2Y_4		V_2Y_4
		U_2Y_5		U_2Y_5
		V_3Y_6		V_3Y_6
		U_3Y_7		U_3Y_7

Table 31-49: 16-bit Split Transmit Mode with SPLTWRD = 1, SUBSPLTODD = 1, and SWAPEN = 0

DMACFG = 1			DMACFG = 0	
PRIMARY DMA DATA (32 bits)	SECONDARY DMA DATA (32 bits)	Pin Data (16 bits)	DMA0 DATA (32 bits)	Pin Data (16 bits)
$Y_3Y_2Y_1Y_0$	$V_3V_2V_1V_0$	V_0Y_0	$V_3V_2V_1V_0$	V_0Y_0
$Y_7Y_6Y_5Y_4$	$U_3U_2U_1U_0$	U_0Y_1	$Y_3Y_2Y_1Y_0$	U_0Y_1
	$V_7V_6V_5V_4$	V_1Y_2	$U_3U_2U_1U_0$	V_1Y_2
	$U_7U_6U_5U_4$	U_1Y_3	$Y_7Y_6Y_5Y_4$	U_1Y_3
		V_2Y_4		V_2Y_4

Table 31-49: 16-bit Split Transmit Mode with SPLTWRD = 1, SUBSPLTODD = 1, and SWAPEN = 0 (Continued)

DMACFG = 1			DMACFG = 0	
PRIMARY DMA DATA (32 bits)	SECONDARY DMA DATA (32 bits)	Pin Data (16 bits)	DMA0 DATA (32 bits)	Pin Data (16 bits)
		U_2Y_5		U_2Y_5
		V_3Y_6		V_3Y_6
		U_3Y_7		U_3Y_7

EPPI Programming Concepts

The following section provides information on SMPTE programming.

SMPTE Modes Programming

The programming model of SMPTE modes is similar to ITU Modes. All programming modes pertaining to ITU modes like XFRTYPE, FSCFG, FLDSEL and BLANKGEN hold true for SMPTE modes as well. The only difference is that since ITU modes use Y-Cr/Cb interleaved data and SMPTE use parallel Y-Cr/Cb data, SPLTWRD should be set while operating in SMPTE modes. The following table describes the programming modes for different SMPTE formats.

Table 31-50: Programming Modes for SMPTE Formats

SMPTE Format	SMPTE Channel Width	EPPI Input Bit Width	EPPI Mode	Remarks
274M	8	16 Cr/Cb - [15:8] Y - [7:0]	DLEN = 16bits SPLTWRD = 1	SIGNEXT not supported
	10	20 Cr/Cb - [19:10] Y - [9:0]	DLEN = 20bits SPLTWRD = 1	SIGNEXT extends each channel data to the 16b boundary
	12	24 Cr/Cb - [23:12] Y - [11:0]	DLEN = 24bits SPLTWRD = 1	SIGNEXT extends each channel data to the 16b boundary
296M	8	16 Cr/Cb - [15:8] Y - [7:0]	DLEN = 16bits SPLTWRD = 1	SIGNEXT not supported
	10	20 Cr/Cb - [19:10] Y - [9:0]	DLEN = 20bits SPLTWRD = 1	SIGNEXT extends each channel data to the 16b boundary

ADSP-BF60x EPPI Register Descriptions

EPPI (EPPI) contains the following registers.

Table 31-51: ADSP-BF60x EPPI Register List

Name	Description
EPPI_STAT	Status Register
EPPI_HCNT	Horizontal Transfer Count Register
EPPI_HDLY	Horizontal Delay Count Register
EPPI_VCNT	Vertical Transfer Count Register
EPPI_VDLY	Vertical Delay Count Register
EPPI_FRAME	Lines Per Frame Register
EPPI_LINE	Samples Per Line Register
EPPI_CLKDIV	Clock Divide Register
EPPI_CTL	Control Register
EPPI_FS1_WLHB	FS1 Width Register / EPPI Horizontal Blanking Samples Per Line Register
EPPI_FS1_PASPL	FS1 Period Register / EPPI Active Samples Per Line Register
EPPI_FS2_WLVB	FS2 Width Register / EPPI Lines Of Vertical Blanking Register
EPPI_FS2_PALPF	FS2 Period Register / EPPI Active Lines Per Field Register
EPPI_IMSK	Interrupt Mask Register
EPPI_ODDCLIP	Clipping Register for ODD (Chroma) Data
EPPI_EVENCLIP	Clipping Register for EVEN (Luma) Data
EPPI_FS1_DLY	Frame Sync 1 Delay Value
EPPI_FS2_DLY	Frame Sync 2 Delay Value
EPPI_CTL2	Control Register 2

Status Register

The EPPI_STAT register contains bits that provide information about the current operating state of the EPPI.

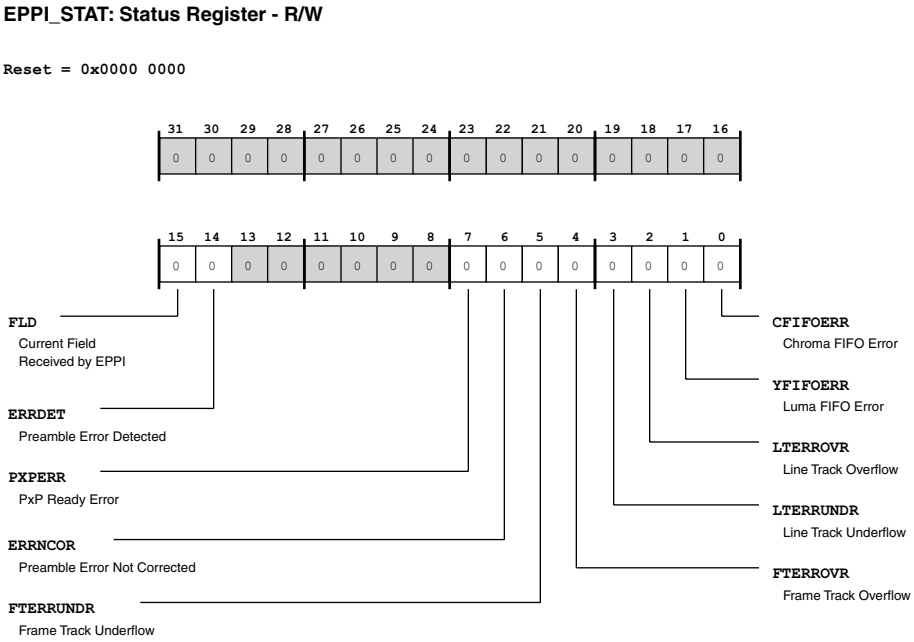


Figure 31-15: EPPI_STAT Register Diagram

Table 31-52: EPPI_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/NW)	FLD	Current Field Received by EPPI. The EPPI_STAT.FLD bit indicates whether the current field being received by the PPI is Field 1 (if clear) or Field 2 (if set).	
		0	Field 1
		1	Field 2
14 (R/W1C)	ERRDET	Preamble Error Detected. The EPPI_STAT.ERRDET bit is useful only in ITU receive modes and indicates if an error has been detected in the status word of EAV or SAV sequences (if set) or not (if clear).	
		0	No preamble error detected
		1	Preamble error detected

Table 31-52: EPPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	PXPERR	PxP Ready Error. The EPPI_STAT.PXPERR bit is valid only in the RX mode. This bit indicates whether the incoming PPI data overflows the PxP interface (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
6 (R/W1C)	ERRNCOR	Preamble Error Not Corrected. The EPPI_STAT.ERRNCOR bit is useful only in the ITU receive modes and indicates if an error in the status word of EAV or SAV sequences can not be cleared (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No uncorrected preamble error has occurred
		1	Preamble error detected but not corrected
5 (R/W1C)	FTERRUNDR	Frame Track Underflow. The EPPI_STAT.FTERRUNDR bit indicates whether a frame track underflow error has occurred (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred
4 (R/W1C)	FTERROVR	Frame Track Overflow. The EPPI_STAT.FTERROVR bit indicates whether a frame track overflow error has occurred (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred
3 (R/W1C)	LTERRUNDR	Line Track Underflow. The EPPI_STAT.LTERRUNDR bit indicates whether a line track underflow error has occurred (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred

Table 31-52: EPPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	LTERROVR	Line Track Overflow. The EPPI_STAT.LTERROVR bit indicates whether a line track overflow error has occurred (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred
1 (R/W1C)	YFIFOERR	Luma FIFO Error. For RX modes, the EPPI_STAT.YFIFOERR bit indicates whether the Luma FIFO has overflowed (if set) or not (if clear). For TX modes, this bit indicates whether the Luma FIFO has underflowed (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred
0 (R/W1C)	CFIFOERR	Chroma FIFO Error. For RX modes, the EPPI_STAT.CFIFOERR bit indicates whether the Chroma FIFO has overflowed (if set) or not (if clear). For TX modes, this bit indicates whether the Chroma FIFO has underflowed (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred

Horizontal Transfer Count Register

The EPPI_HCNT register holds the number of samples to read in or write out per line, after EPPI_HDLY number of cycles have expired since the assertion of PPI_FS1. Any write to the EPPI_LINE register modifies the EPPI_HCNT register, but any write to EPPI_HCNT does not affect the EPPI_LINE register value. So, the EPPI_HCNT register should be programmed after the EPPI_LINE register.

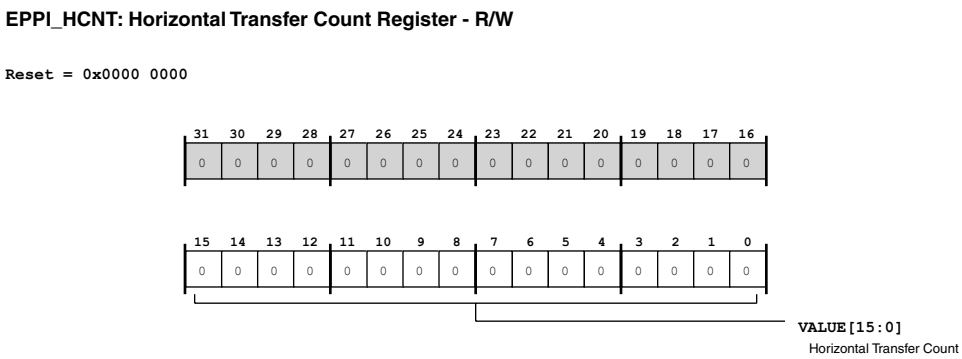


Figure 31-16: EPPI_HCNT Register Diagram

Table 31-53: EPPI_HCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Horizontal Transfer Count. The EPPI_HCNT.VALUE holds the number of samples to read in or write out per line, after EPPI_HDLY number of cycles have expired since the last assertion of PPI_FS1.

Horizontal Delay Count Register

The EPPI_HDLY register contains the number of clock cycles to delay after the assertion of PPI_FS1 is detected before starting to read or write data.

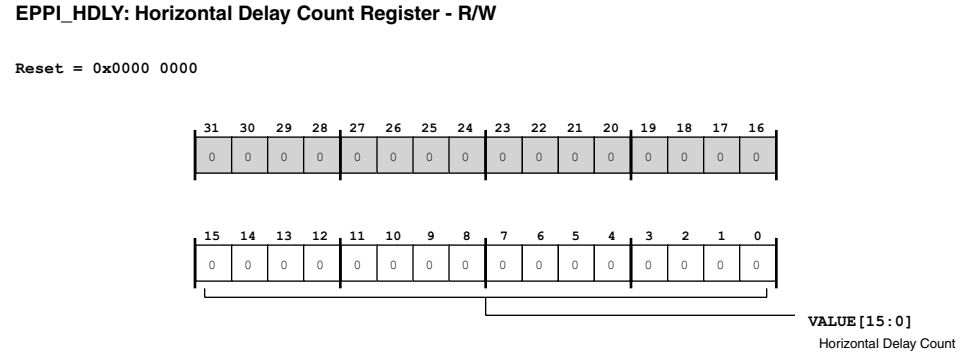


Figure 31-17: EPPI_HDLY Register Diagram

Table 31-54: EPPI_HDLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Horizontal Delay Count. The EPPI_HDLY.VALUE holds the number of PPI_CLK cycles to delay after assertion of PPI_FS1 before starting to read or write data.

Vertical Transfer Count Register

The EPPI_VCNT register holds the number of lines to read in or write out, after EPPI_VDLY number of lines from the start of frame. Any write to the EPPI_FRAME register modifies the EPPI_VCNT register, but any write to EPPI_VCNT does not affect the EPPI_FRAME register value. So, the EPPI_VCNT register should be programmed after the EPPI_FRAME register.

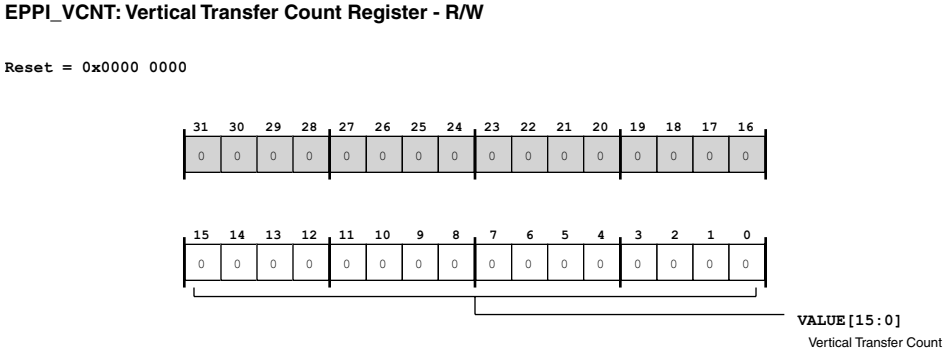


Figure 31-18: EPPI_VCNT Register Diagram

Table 31-55: EPPI_VCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Vertical Transfer Count. The EPPI_VCNT.VALUE holds the number of lines to read in or write out, after EPPI_VDLY number of lines from the start of frame.

Vertical Delay Count Register

The EPPI_VDLY register contains the number of lines to wait after the start of a new frame before starting to read/transmit data.

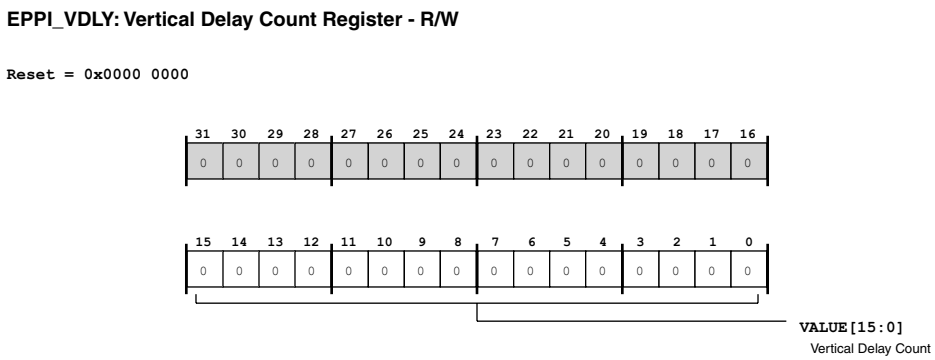


Figure 31-19: EPPI_VDLY Register Diagram

Table 31-56: EPPI_VDLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Vertical Delay Count. The EPPI_VDLY.VALUE holds the number of lines to wait after the start of a new frame before starting to read/transmit data.

Lines Per Frame Register

The EPPI_FRAME register tracks the frame track overflow and underflow errors. This register should be programmed with the number of lines expected per frame. Any write to the EPPI_FRAME register will also write the same value to the EPPI_VCNT register, but any write to EPPI_VCNT does not affect the EPPI_FRAME register value. So, the EPPI_FRAME register should be programmed before the EPPI_VCNT register.

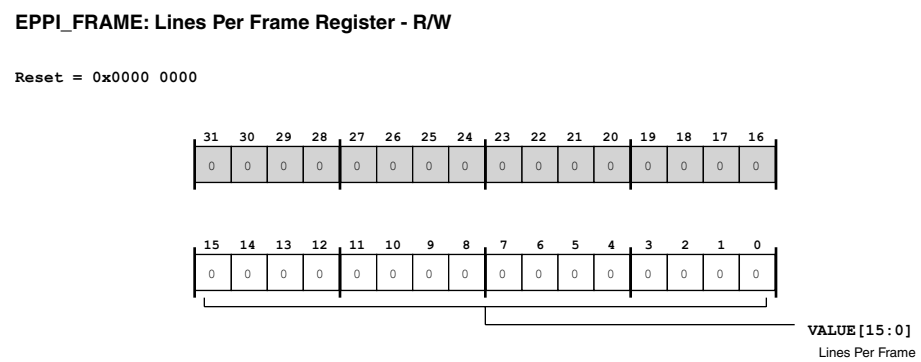


Figure 31-20: EPPI_FRAME Register Diagram

Table 31-57: EPPI_FRAME Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Lines Per Frame. The EPPI_FRAME.VALUE holds the number of lines expected per frame of data.

Samples Per Line Register

The EPPI_LINE register tracks the line track overflow and underflow Errors. This register should be programmed with the number of samples expected per line. Any write to the EPPI_LINE register will also write the same value to the EPPI_HCNT register, but any write to EPPI_HCNT does not affect the EPPI_LINE register value. So, the EPPI_LINE register should be programmed before the EPPI_HCNT register.

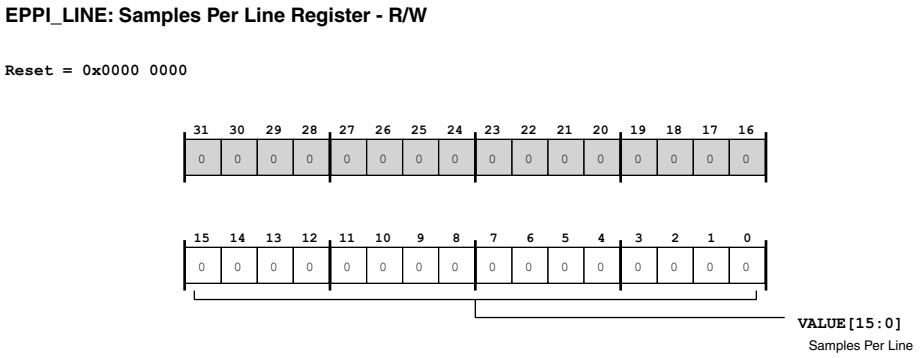


Figure 31-21: EPPI_LINE Register Diagram

Table 31-58: EPPI_LINE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Samples Per Line. The EPPI_LINE.VALUE holds the number of samples expected per line.

Clock Divide Register

The EPPI_CLKDIV register provides the divisor for EPPI internal clock generation. The generated clock frequency is given by following formula:

PPI_CLK = (SCLK) / (2 * (EPPI_CLKDIV + 1))

Note that a value of 0xFFFF is invalid for the EPPI_CLKDIV register.

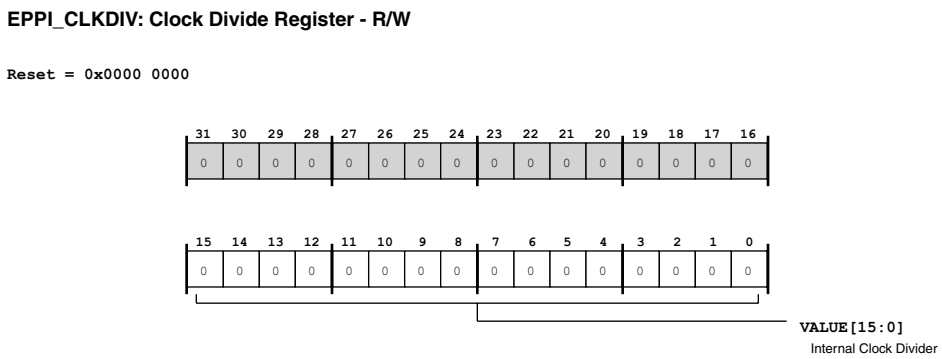


Figure 31-22: EPPI_CLKDIV Register Diagram

Table 31-59: EPPI_CLKDIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Internal Clock Divider.

Control Register

The EPPI_CTL register configures the EPPI for operating mode, control signal polarities, and data width of the port.

EPPI_CTL: Control Register - R/W

Reset = 0x0000 0000

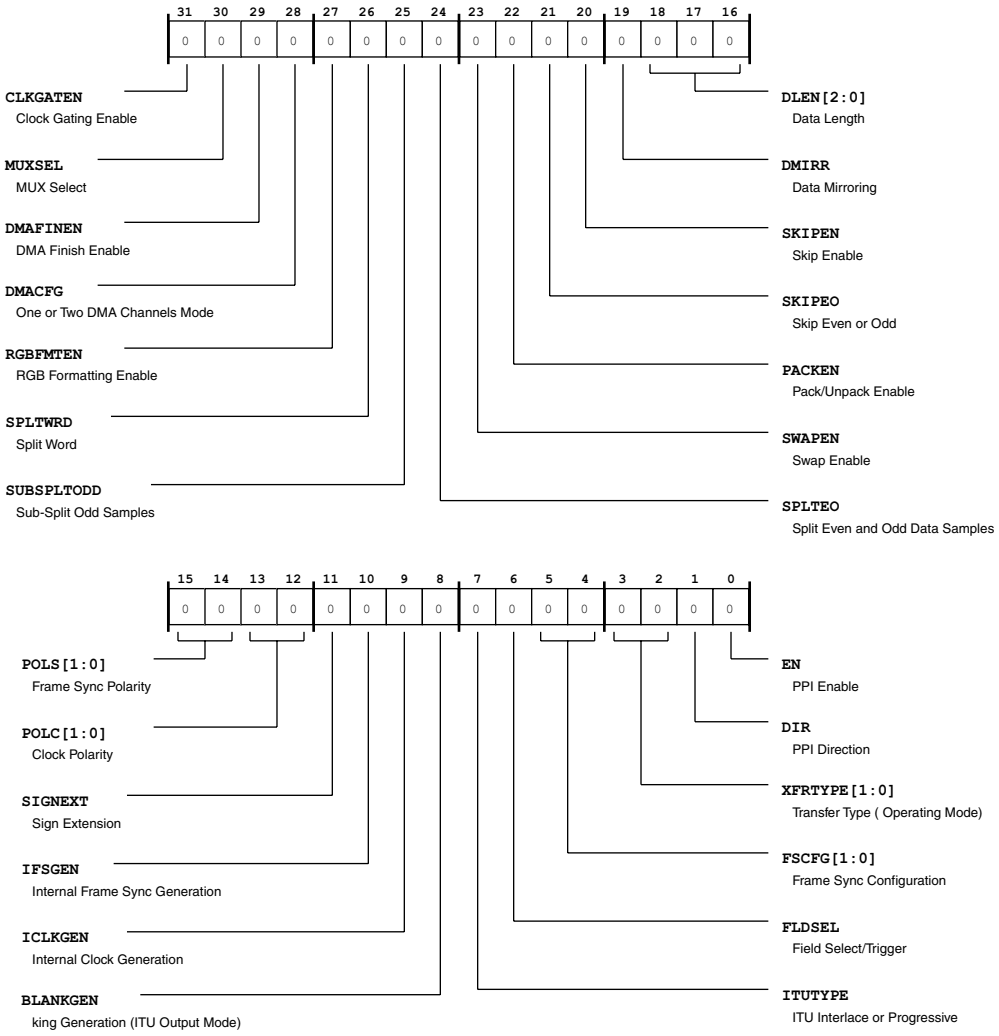


Figure 31-23: EPPI_CTL Register Diagram

Table 31-60: EPPI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	CLKGATEN	Clock Gating Enable. The EPPI_CTL.CLKGATEN bit enables using the PPI_FS3 pin as a clock gating pin. When EPPI_CTL.CLKGATEN is set, the PPI_FS3 pin acts as a clock gating signal, and both the internal and external clock are gated. Note that the PPI_FS3 pin gating signal is active low, and the EPPI_CTL.CLKGATEN selection is ignored if EPPI_CTL.MUXSEL is set or EPPI_CTL.FSCFG equals 0x3.	
		0	Disable
		1	Enable
30 (R/W)	MUXSEL	MUX Select. The EPPI_CTL.MUXSEL bit enables multiplexing of a primary and alternate camera using the EPPI main data and clock lines. For more information on this feature, see the EPPI functional description.	
		0	Normal Operation
		1	Multiplexed Operation
29 (R/W)	DMAFINEN	DMA Finish Enable. The EPPI_CTL.DMAFINEN bit selects whether or not the EPPI sends a finish command (010) through the DDE COMMAND line soon after a frame/line is received completely.	
		0	No Finish Command
		1	Enable Send Finish Command
28 (R/W)	DMACFG	One or Two DMA Channels Mode. The EPPI_CTL.DMACFG bit is valid only if EPPI_CTL.SPLTEO is set. If EPPI_CTL.DMACFG is set, the EPPI uses two DMA channels. And, if EPPI_CTL.DMACFG is cleared, the EPPI uses only one DMA channel.	
		0	PPI uses one DMA Channel
		1	PPI uses two DMA Channels

Table 31-60: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
27 (R/W)	RGBFMTEN	RGB Formatting Enable. For 16- or 18-bit transmit modes only, the EPPI_CTL.RGBFMTEN bit enables conversion of RGB888 from memory into RGB666 output data (18-bit transmit) or enables conversion of RGB888 from memory into RGB565 output data (16-bit transmit). Note that EPPI_CTL.SPLTEO and EPPI_CTL.RGBFMTEN should never be set simultaneously.	
		0	Disable RGB Formatted Output
		1	Enable RGB Formatted Output
26 (R/W)	SPLTWRD	Split Word. The EPPI_CTL.SPLTWRD bit selects split word data placement when the data length (EPPI_CTL.DLEN) selects 16-, 20-, or 24-bit data words. For all other EPPI_CTL.SPLTWRD values, the set or clear selections for EPPI_CTL.SPLTWRD produce the same result (act as though EPPI_CTL.SPLTWRD is cleared). For EPPI_CTL.SPLTWRD set, the EPPI_CTL.DLEN values below result in the following combinations of split words: DLEN Cr/Cb data Y data 16 PPI_DATA[15:8] PPI_DATA[7:0] 20 PPI_DATA[19:10] PPI_DATA[9:0] 24 PPI_DATA[23:12] PPI_DATA[11:0]	
		0	PPI_DATA has (DLEN-1) bits of Y or Cr or Cb
		1	PPI_DATA contains 2 elements per word
25 (R/W)	SUBSPLTODD	Sub-Split Odd Samples. The EPPI_CTL.SUBSPLTODD bit is valid only if EPPI_CTL.SPLTEO is set. If EPPI_CTL.SUBSPLTODD is set, the EPPI sub-splits the odd sub-stream, and packs them separately.	
		0	Disable
		1	Enable
24 (R/W)	SPLTEO	Split Even and Odd Data Samples. If EPPI_CTL.SPLTEO is set, the EPPI splits the incoming data stream into two sub-streams, an even stream and an odd stream, and packs them separately.	
		0	Do Not Split Samples
		1	Split Even/Odd Samples

Table 31-60: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
23 (R/W)	SWAPEN	Swap Enable. The EPPI_CTL.SWAPEN selects whether or not to swap the order of the first data (most-significant bits versus least-significant bits) of the DMA word. For receive modes, the EPPI puts the first data in the most significant bits (if set) or puts the first data in the least significant bits (if cleared) of the DMA word. For transmit modes, the EPPI transmits the most significant bits in the DMA word as the first data (if set) or transmits the least significant bits in the DMA word as the first data (if cleared).	
		0	Disable
		1	Enable
22 (R/W)	PACKEN	Pack/Unpack Enable. The EPPI_CTL.PACKEN select whether or not packing is enabled (for receive modes) and unpacking is enabled (for transmit modes). When this bit is set, EPPI transfer DMA is 32-bits wide. When this bit is cleared and the EPPI_CTL.DLEN is less than or equal to 16 bits, EPPI transfer DMA is 16-bits wide. For receive modes, if this bit is set, then the EPPI packs the incoming data into 32-bit words. If this bit is cleared, then the EPPI does not do any packing. For transmit modes, if this bit is set, then the EPPI always unpacks the 32-bit data from DMA. If this bit is not set, the EPPI does not do any unpacking.	
		0	Disable
		1	Enable
21 (R/W)	SKIPEO	Skip Even or Odd. The EPPI_CTL.SKIPEO bit selects whether even (if set) or odd (if cleared) samples are skipped if sample skipping is enabled EPPI_CTL.SKIPEN is set). This feature only is useful for receive modes.	
		0	Skip Odd Samples
		1	Skip Even Samples
20 (R/W)	SKIPEN	Skip Enable. The EPPI_CTL.SKIPEN bit enables skipping alternate samples. This feature only is useful for receive modes.	
		0	No Samples Skipping
		1	Skip Alternate Samples

Table 31-60: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	DMIRR	Data Mirroring. The EPPI_CTL.DMIRR field enables mirroring (bit reversing) the data coming in or going out on the EPPI data pins. Pin Data / PPI Data (DAT_MRR=0) / PPI Data (DAT_MRR=1) 15 / 15 / 0 14 / 14 / 1 .... / .... / .... 1 / 1 / 14 0 / 0 / 15	
		0	No Data Mirroring
		1	Data Mirroring
18:16 (R/W)	DLEN	Data Length. The EPPI_CTL.DLEN bits select the data length for the EPPI. Note that the 20 bits data length selection is valid only for SMPTE modes (EPPI_CTL.SPLTWRD set).	
		0	8 bits
		1	10 bits
		2	12 bits
		3	14 bits
		4	16 bits
		5	18 bits
		6	20 bits
		7	24 bits
15:14 (R/W)	POLS	Frame Sync Polarity. The EPPI_CTL.POLS selects whether the frame syncs' polarity is active low versus active high.	
		0	FS1 and FS2 are active high
		1	FS1 is active low. FS2 is active high
		2	FS1 is active high. FS2 is active low
		3	FS1 and FS2 are active low

Table 31-60: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13:12 (R/W)	POLC	Clock Polarity. The EPPI_CTL.POLC selects the rising versus falling edge for sampling data and sampling/driving syncs.	
		0	Clock/Sync polarity mode 0 For Receive mode: Sample data on falling edge and sample/drive syncs on falling edge For Transmit mode: Drive data on rising edge and sample/drive syncs on rising edge
		1	Clock/Sync polarity mode 1 For Receive mode: Sample data on falling edge and sample/drive syncs on rising edge For Transmit mode: Drive data on rising edge and sample/drive syncs on falling edge
		2	Clock/Sync polarity mode 2 For Receive mode: Sample data on rising edge and sample/drive syncs on falling edge For Transmit mode: Drive data on falling edge and sample/drive syncs on rising edge
		3	Clock/Sync polarity mode 3 For Receive mode: Sample data on rising edge and sample/drive syncs on rising edge For Transmit mode: Drive data on falling edge and sample/drive syncs on falling edge
11 (R/W)	SIGNEXT	Sign Extension. The EPPI_CTL.SIGNEXT select whether (for receive modes when EPPI_CTL.DLEN selecting 16 bit data length) the data is sign extended or zero filled. Not that EPPI_CTL.SPLTWRD is removed from this shared bit.	
		0	Zero Filled
		1	Sign Extended
10 (R/W)	IFSGEN	Internal Frame Sync Generation. The EPPI_CTL.IFSGEN bit selects whether the frame syncs are generated internally or are supplied by an external device.	
		0	External Frame Sync
		1	Internal Frame Sync

Table 31-60: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	ICLKGEN	Internal Clock Generation. The EPPI_CTL.ICLKGEN bit selects whether the PPI_CLK is generated internally or is supplied by an external device.	
		0	External Clock
		1	Internal Clock
8 (R/W)	BLANKGEN	king Generation (ITU Output Mode). The EPPI_CTL.BLANKGEN enables ITU output with internal blanking. In GP 8, 10 transmit bit modes (when EPPI_CTL.SPLTWRD is cleared) and 16-, 20-, and 24-bit transmit modes (when EPPI_CTL.SPLTWRD is set), EPPI_CTL.BLANKGEN selects whether or not the EPPI generates blanking and generates preamble and insertion with active data from memory.	
		0	Disable
		1	Enable
7 (R/W)	ITUTYPE	ITU Interlace or Progressive. The EPPI_CTL.ITUTYPE selects interlaced or progressive operation for ITU656 mode. This selection is valid for both TX and RX modes.	
		0	Interlaced
		1	Progressive
6 (R/W)	FLDSEL	Field Select/Trigger. The EPPI_CTL.FLDSEL bits configure the EPPI field and trigger selection. These are valid for GP modes (EPPI_CTL.XFRTYPE =0x3) and ITU656 active video mode (EPPI_CTL.XFRTYPE cleared).	
		0	Field Mode 0 Read Field 1 (for ITU656 active video mode). Set internal trigger (for GP RX mode). FS3 is toggled on FS2 assertion followed by FS1 assertion (when EPPI_CTL.FSCFG selects sync mode 3 and EPPI_CTL.IFSGEN selects internal frame sync).
		1	Field Mode 1 Read Field 1 and Field 2 (ITU656 active video mode). Set external trigger (GP RX mode). FS3 is toggled on FS2 assertion (when EPPI_CTL.FSCFG selects sync mode 3 and EPPI_CTL.IFSGEN selects internal frame sync).

Table 31-60: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W)	FSCFG	Frame Sync Configuration. The EPPI_CTL.FSCFG bits configure the EPPI frame syncs. These are valid only for GP modes (EPPI_CTL.XFRTYPE =0x3). The output of the frames syncs also depends on whether the EPPI transfer direction is transmit and the EPPI is in ITU output mode (EPPI_CTL.BLANKGEN is set}.	
		0	Sync Mode 0 FS0 driven in GP mode. FS0 not driven in ITU output mode.
		1	Sync Mode 1 FS1 driven in GP mode. HSYNC driven on FS1 in ITU output mode.
		2	Sync Mode 2 FS2 driven in GP mode. HSYNC driven on FS1 and VSYNC driven on FS1 in ITU output mode.
		3	Sync Mode 3 FS3 driven in GP mode. HSYNC driven on FS1, VSYNC driven on FS2, and FIELD driven on FS3 in ITU output mode.
3:2 (R/W)	XFRTYPE	Transfer Type (Operating Mode). The EPPI_CTL.XFRTYPE bits select the EPPI operating mode. In receive mode (EPPI_CTL.DIR cleared), the EPPI modes include ITU656 active video only mode, ITU656 entire field mode, ITU656 vertical blanking only mode, and non-ITU656 mode (GP mode). For transmit mode (EPPI_CTL.DIR set), the EPPI_CTL.XFRTYPE bits have no effect, and the EPPI (in transmit mode) is always in GP mode.	
		0	ITU656 Active Video Only Mode
		1	ITU656 Entire Field Mode
		2	ITU656 Vertical Blanking Only Mode
		3	Non-ITU656 Mode (GP Mode)
1 (R/W)	DIR	PPI Direction. The EPPI_CTL.DIR bit selects whether the EPPI is in receive mode (if cleared) or in transmit mode (if set).	
		0	Receive Mode
		1	Transmit Mode

Table 31-60: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	PPI Enable. The EPPI_CTL.EN bit enables or disables the EPPI.	
		0	Disable
		1	Enable

FS1 Width Register / EPPI Horizontal Blanking Samples Per Line Register

The EPPI_FS1_WLHB register's content varies depending on whether the EPPI is in GP1/2/3 FS modes or in GP transmit mode.

In GP 1, 2 or 3 FS modes, EPPI_FS1_WLHB is used for the generation of Frame Sync 1. The register contains the width required for PPI_FS1 based on the PPI_CLK clock.

In GP transmit mode with EPPI_CTL.BLANKGEN set, this register contains the number of samples of horizontal blanking per line. When used for blanking generation, only the lower 16 bits are valid.

Note that a value of 0 for the EPPI_FS1_WLHB register is illegal. If programmed as 0, the EPPI regards the EPPI_FS1_WLHB as containing 1.

EPPI_FS1_WLHB: FS1 Width Register / EPPI Horizontal Blanking Samples Per Line Register - R/W

Reset = 0x0000 0000

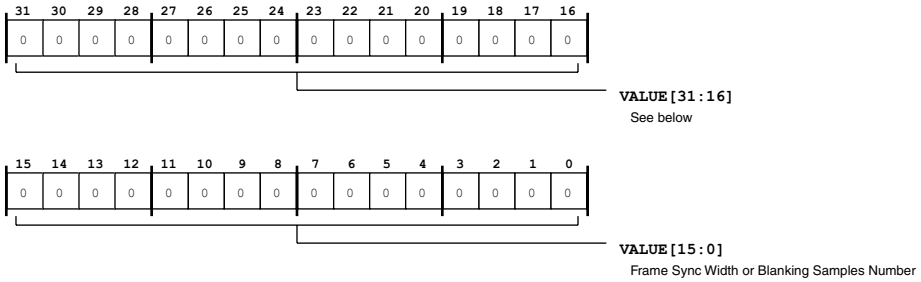


Figure 31-24: EPPI_FS1_WLHB Register Diagram

Table 31-61: EPPI_FS1_WLHB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Frame Sync Width or Blanking Samples Number.

FS1 Period Register / EPPI Active Samples Per Line Register

The EPPI_FS1_PASPL register content varies depending on whether the EPPI is in GP1/2/3 FS modes or in GP transmit mode.

In GP 1, 2, or 3 FS modes, EPPI_FS1_PASPL is used for the generation of Frame Sync 1. The register contains the period required for PPI_FS1 based on the PPI_CLK clock.

In GP transmit mode with EPPI_CTL.BLANKGEN set, this register contains the number of samples of active video or vertical blanking samples per line. When used for blanking generation, only the lower 16 bits are valid.

Note that a value of 0 for this register is illegal. If programmed as 0, the EPPI regards the EPPI_FS1_PASPL as containing 1.

EPPI_FS1_PASPL: FS1 Period Register / EPPI Active Samples Per Line Register - R/W

Reset = 0x0000 0000

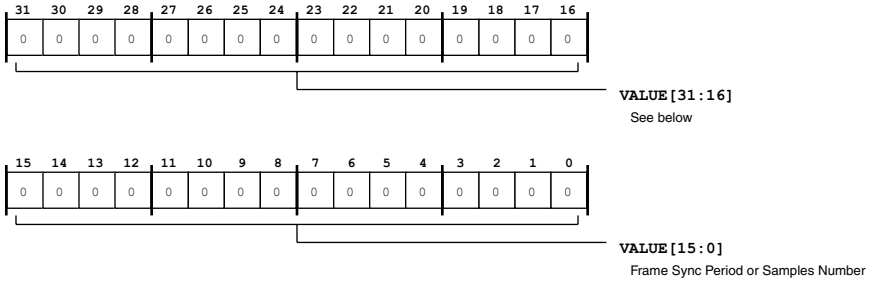


Figure 31-25: EPPI_FS1_PASPL Register Diagram

Table 31-62: EPPI_FS1_PASPL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Frame Sync Period or Samples Number.

FS2 Width Register / EPPI Lines Of Vertical Blanking Register

The EPPI_FS2_WLVB register content varies depending on whether the EPPI is in GP2/3 FS modes or in GP transmit mode.

In GP 2 or 3 FS modes, EPPI_FS2_WLVB is used for the generation of Frame Sync 2. The register contains the width required for PPI_FS2 based on the PPI_CLK clock.

In GP transmit mode with EPPI_CTL.BLANKGEN set, this register contains the number or lines of vertical blanking.

Note that for progressive video, EPPI_FS2_WLVB.F2VBBD and EPPI_FS2_WLVB.F2VBAD are ignored.

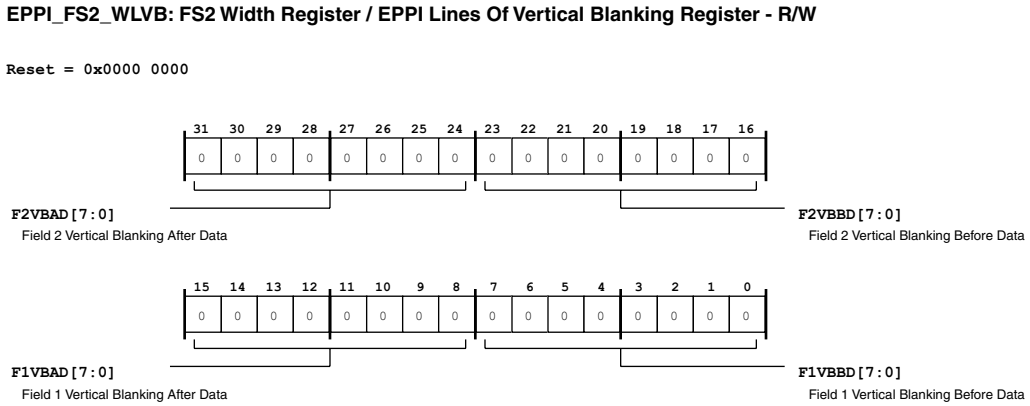


Figure 31-26: EPPI_FS2_WLVB Register Diagram

Table 31-63: EPPI_FS2_WLVB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	F2VBAD	Field 2 Vertical Blanking After Data. The number of lines of vertical blanking after field 2.
23:16 (R/W)	F2VBBD	Field 2 Vertical Blanking Before Data. The number of lines of vertical blanking before field 2.
15:8 (R/W)	F1VBAD	Field 1 Vertical Blanking After Data. The number of lines of vertical blanking after field 1.
7:0 (R/W)	F1VBBD	Field 1 Vertical Blanking Before Data. The number of lines of Vertical blanking before field 1.

FS2 Period Register / EPPI Active Lines Per Field Register

The EPPI_FS2_PALPF register content varies depending on whether the EPPI is in GP2/3 FS modes or in GP transmit mode.

In GP 2 or 3 FS modes, EPPI_FS2_PALPF is used for the generation of Frame Sync 2. This register contains the period required for PPI_FS2 based on the PPI_CLK clock.

In GP transmit mode with EPPI_CTL.BLANKGEN set, this register contains the number of lines of active video per field.

Note that a value of 0 for EPPI_FS2_PALPF.F1ACT or EPPI_FS2_PALPF.F2ACT is illegal. If either is programmed as 0, the EPPI regard the 0 value fields as containing 1.

Also note that for progressive video, EPPI_FS2_PALPF.F2ACT is ignored.

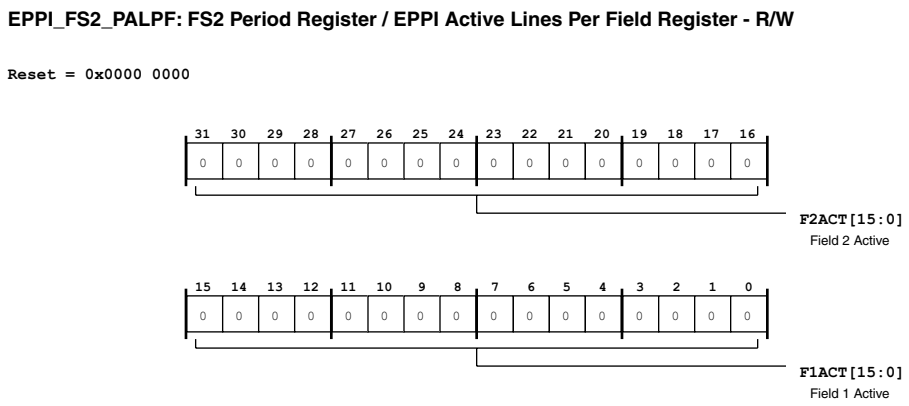


Figure 31-27: EPPI_FS2_PALPF Register Diagram

Table 31-64: EPPI_FS2_PALPF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	F2ACT	Field 2 Active. The number of lines of Active Data in Field 2.
15:0 (R/W)	F1ACT	Field 1 Active. The number of lines of Active Data in Field 1.

Interrupt Mask Register

The EPPI_IMSK permits masking (if associated bit is set) of EPPI error interrupts for YFIFO underflow or overflow, CFIFO underflow or overflow, line track overflow error, line track underflow error, frame track overflow error, frame track underflow error, and ERR_NCOR (ITU preamble error not corrected. These conditions are flagged in the EPPI_STAT register and cleared by write-1-to-clear.

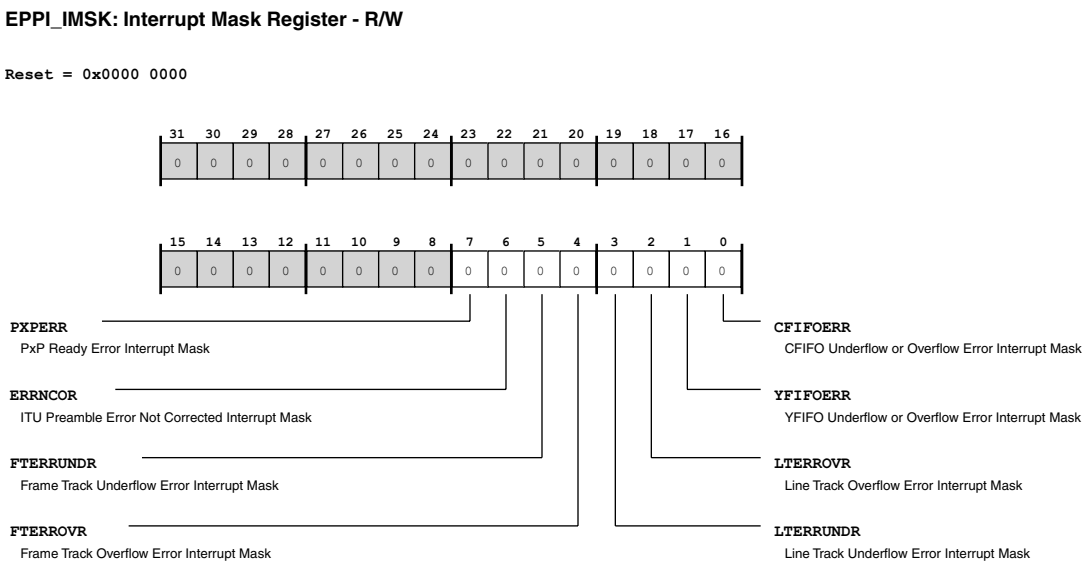


Figure 31-28: EPPI_IMSK Register Diagram

Table 31-65: EPPI_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	PXPERR	PxP Ready Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
6 (R/W)	ERRNCOR	ITU Preamble Error Not Corrected Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
5 (R/W)	FTERRUNDR	Frame Track Underflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
4 (R/W)	FTERROVR	Frame Track Overflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
3 (R/W)	LTERRUNDR	Line Track Underflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt

Table 31-65: EPPI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	LTERROVR	Line Track Overflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
1 (R/W)	YFIFOERR	YFIFO Underflow or Overflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
0 (R/W)	CFIFOERR	CFIFO Underflow or Overflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt

Clipping Register for ODD (Chroma) Data

The EPPI_ODDCLIP register selects the clipping threshold for chroma data, which provides clipping of individual video components.

The high odd and low odd spaces in EPPI_ODDCLIP are 16-bits wide and (depending on the EPPI_CTL.DLEN selection) only the corresponding video component bits are considered for clipping.

For example, if the EPPI is programmed in 10-bit mode, bits [9:0] and bits 25:16 constitute the clipping thresholds. The higher bits are (in this case) ignored.

Using the this method, 8-, 10-, 12- and 16-bit clipping thresholds can be set.

Note that when the EPPI is programmed in 16-, 20-, or 24-bit mode with EPPI_CTL.SPLTWRD set, the luma data gets the clipping threshold levels of EPPI_EVENCLIP, and the chroma data gets the clipping threshold levels of EPPI_ODDCLIP.

Also note that the EPPI_EVENCLIP and EPPI_ODDCLIP registers are ignored when EPPI_CTL.RGBFMTEN is set.

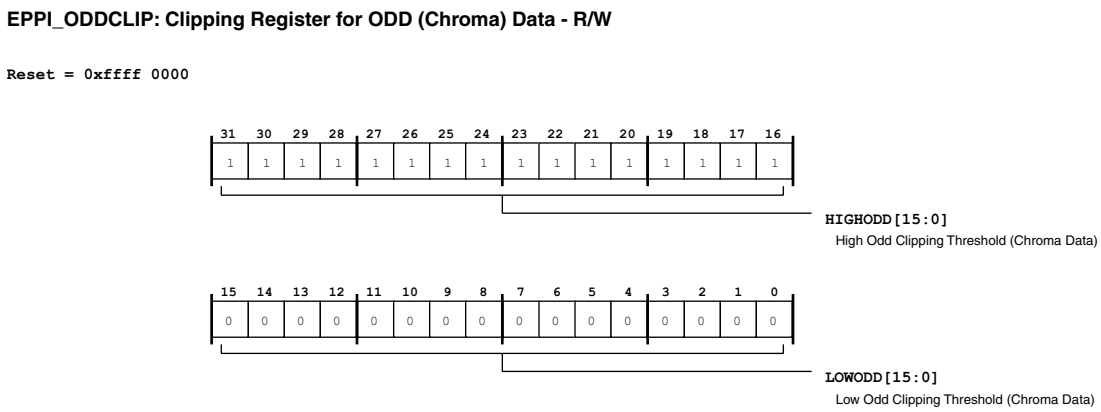


Figure 31-29: EPPI_ODDCLIP Register Diagram

Table 31-66: EPPI_ODDCLIP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	HIGHODD	High Odd Clipping Threshold (Chroma Data).
15:0 (R/W)	LOWODD	Low Odd Clipping Threshold (Chroma Data).

Clipping Register for EVEN (Luma) Data

The EPPI_EVENCLIP register selects the clipping threshold for luma data, which provides clipping of individual video components.

The high even and low even spaces in EPPI_EVENCLIP are 16-bits wide and (depending on the EPPI_CTL.DLEN selection) only the corresponding video component bits are considered for clipping.

For example, if the EPPI is programmed in 10-bit mode, bits [9:0] and bits 25:16 constitute the clipping thresholds. The higher bits are (in this case) ignored.

Using the this method, 8-, 10-, 12- and 16-bit clipping thresholds can be set.

Note that when the EPPI is programmed in 16-, 20-, or 24-bit mode with EPPI_CTL.SPLTWRD set, the luma data gets the clipping threshold levels of EPPI_EVENCLIP, and the chroma data gets the clipping threshold levels of EPPI_ODDCLIP.

Also note that the EPPI_EVENCLIP and EPPI_ODDCLIP registers are ignored when EPPI_CTL.RGBFMTEN is set.

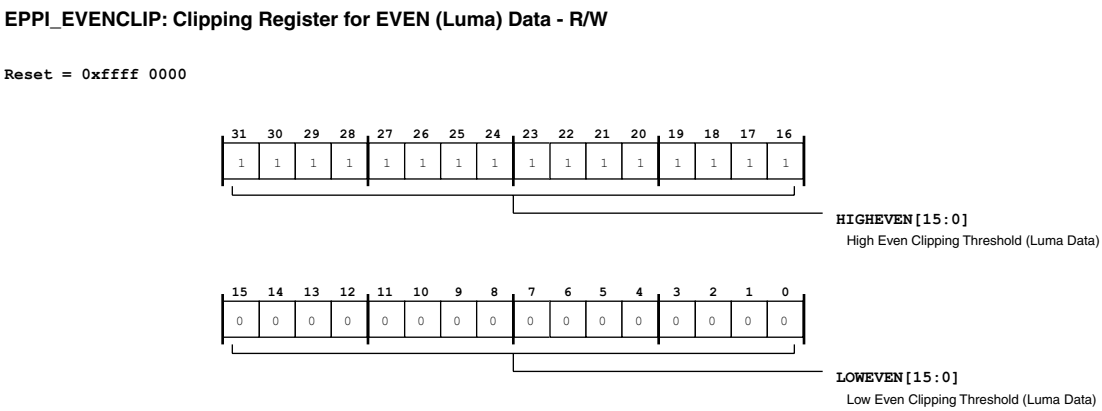


Figure 31-30: EPPI_EVENCLIP Register Diagram

Table 31-67: EPPI_EVENCLIP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	HIGHEVEN	High Even Clipping Threshold (Luma Data).
15:0 (R/W)	LOWEVEN	Low Even Clipping Threshold (Luma Data).

Frame Sync 1 Delay Value

The EPPI_FS1_DLY selects the delay count (based on the period of the PPI_CLK clock) between the first rising edge of PPI_CLK after EPPI enabled and the first active edge of the associated Frame Sync when the internal Frame Sync is used.

Note that if EPPI_FS1_DLY or EPPI_FS2_DLY are programmed with value 0, the EPPI operates as though 0 value is 1, and the first frame sync transition occurs after the completion of one period value of the respective counters.

EPPI_FS1_DLY: Frame Sync 1 Delay Value - R/W

Reset = 0x0000 0000

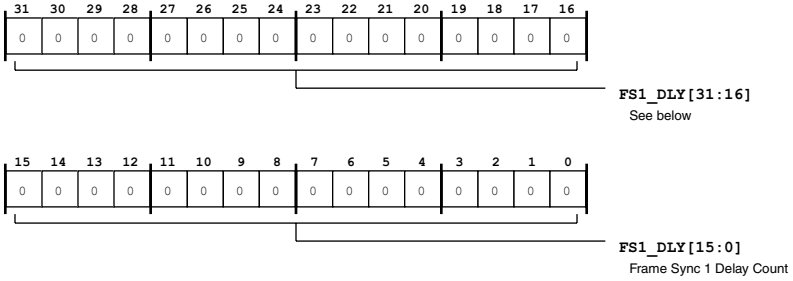


Figure 31-31: EPPI_FS1_DLY Register Diagram

Table 31-68: EPPI_FS1_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	FS1_DLY	Frame Sync 1 Delay Count.

Frame Sync 2 Delay Value

The EPPI_FS2_DLY selects the delay count (based on the period of the PPI_CLK clock) between the first rising edge of PPI_CLK after EPPI enabled and the first active edge of the associated Frame Sync when the internal Frame Sync is used.

Note that if EPPI_FS1_DLY or EPPI_FS2_DLY are programmed with value 0, the EPPI operates as though 0 value is 1, and the first frame sync transition occurs after the completion of one period value of the respective counters.

EPPI_FS2_DLY: Frame Sync 2 Delay Value - R/W

Reset = 0x0000 0000

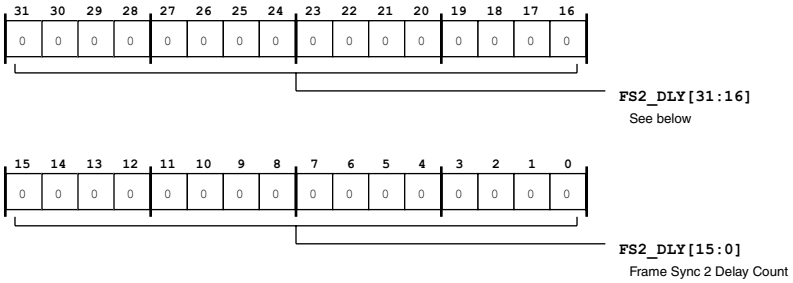


Figure 31-32: EPPI_FS2_DLY Register Diagram

Table 31-69: EPPI_FS2_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	FS2_DLY	Frame Sync 2 Delay Count.

Control Register 2

The EPPI_CTL2 register HSYNC finish signal generation.

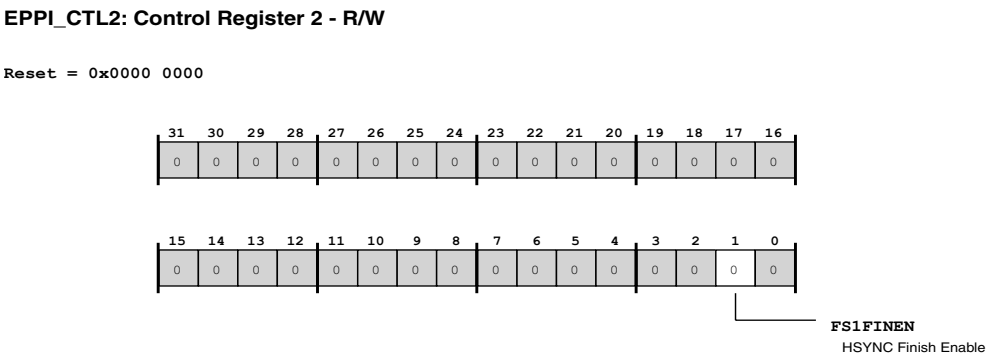


Figure 31-33: EPPI_CTL2 Register Diagram

Table 31-70: EPPI_CTL2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	FS1FINEN	HSYNC Finish Enable. The EPPI_CTL2.FS1FINEN bit selects whether (if set) the EPPI sends a finish command (010) through the DDE COMMAND line soon after a LINE is received completely or (if cleared) the EPPI sends a finish command (010) through the DDE COMMAND line soon after a FRAME is received completely. Note that the EPPI_CTL.DMAFINEN bit must be set for the EPPI to generate either of the finish commands.	
		0	Finish sent after frame RX done PPI sends a finish command (010) through the DDE COMMAND line soon after a FRAME is received completely
		1	Finish sent after frame/line RX done PPI sends a finish command (010) through the DDE COMMAND line soon after a frame/line is received completely

32 Pixel Compositor (PIXC)

The pixel compositor (PIXC) provides data overlay, transparent color, and color space conversion support for different video outputs including active (TFT) flat-panel digital color/monochrome LCD displays and analog NTSC/PAL. The color space conversion and text/graphic overlay capabilities, along with visual effect controls, such as transparency control, shortens the processing time on an image data stream, reduces power consumption and saves system board space by removing the need for external glue logic.

The PIXC is used to combine and format the data streams required by a wide variety of digital LCD panels and NTSC/PAL analog encoders. It provides all the control needed to allow two data streams from two separate data buffers to be combined and converted into appropriate formats for both LCD panels and video output displays. The main image buffer provides the basic background image presented in the data stream. The overlay image buffer allows programs to add foreground text and graphics on top of the main image data stream. This feature is useful for printing additional graphical or textual information on the screen, such as symbols or a menu, while showing the main image in the background.

Overlay is an option and can be enabled or disabled. If it is disabled, the blender/compositor is bypassed and the data stream from the main image buffer goes directly to memory with optional color space conversion.

Transparent color is just a special case of *blending*, masking off the blend operation on a pixel-by-pixel basis. In other words, the overlay region consists of sub-regions in any particular color convenient to the programmer and then, if the color data for a given overlay pixel matches the specified transparent color, the overlay function is masked for that pixel and its data is taken solely from the main image buffer, which is stored in memory in either YUV 4:2:2 interleaved format or RGB888 format .

Regardless of the data format or buffer structure, each color element is 8 bits wide. If overlay is enabled, a graphics/text overlay data buffer is defined in memory. The color space converter can switch positions among any of the three locations; it can be in the image data path, the overlay data path, or after the blender. The exact position of the color space converter depends on the input and output data formats.

Since the end display may be a TV (NTSC/PAL) or an LCD panel, and since the image/overlay input buffers may be in either RGB888 or YUV4:2:2 format, a color space conversion may be needed. The color space conversion is selected according to the input data stream format of the PIXC. A YUV-to-RGB format conversion is necessary if the end display is an LCD and if either of the PIXC input data streams is in YUV 4:2:2 format. Similarly, an RGB-to-YUV format conversion is necessary if the end display is a TV and if either of the PIXC input data streams is in RGB888 format.

If the final display device is an LCD, the output RGB data stream is always packed in RGB 8-bit serial format when transferring back to memory. Similarly, if the final display device is a TV, the YUV data stream is always packed in YUV 4:2:2 interleaved format when transferring back to memory.

PIXC Features

PIXC features include:

- Hardware-based graphics and text overlays
- YUV 4:2:2, RGB888, RGB666 or RGB565 input data formats
- Programmable color space conversion on the main image or the overlay image data path
- Overlay content transparency ratio control
- Transparent color, specified in the desired color space (RGB or YUV)
- Two DMA input channels and one DMA output channel
- Interface to the pixel pipe

PIXC Functional Description

The PIXC implements the following primary functions.

- Graphics/text overlay (including video overlay for small frame sizes)
- Transparency control (alpha blending) of the overlay pixel data
- Transparent color (chroma keying) of the overlay stream
- Color space conversion for LCD panels or NTSC/PAL displays

ADSP-BF60x PIXC Register List

The pixel compositor (PIXC) provides data overlay, transparent color, and color space conversion support for active (TFT) flat-panel digital color/monochrome LCD displays or analog NTSC/PAL video output. A set of registers govern PIXC operations. For more information on PIXC functionality, see the PIXC register descriptions. Programmers should avoid writing to any of the PIXC registers when the module is enabled. Writing to the PIXC registers during the module enabled state can lead to unpredictable behavior of the PIXC. All registers can be read when the PIXC is in the enabled state, and this does not cause any change of status in the PIXC, but register writes should happen only when the PIXC is disabled, or stalled by an interrupt condition.

Table 32-1: ADSP-BF60x PIXC Register List

Name	Description
PIXC_CTL	Control Register

Table 32-1: ADSP-BF60x PIXC Register List (Continued)

Name	Description
PIXC_PPL	Pixels Per Line Register
PIXC_LPF	Line Per Frame Register
PIXC_HSTART_A	Overlay A Horizontal Start Register
PIXC_HEND_A	Overlay A Horizontal End Register
PIXC_VSTART_A	Overlay A Vertical Start Register
PIXC_VEND_A	Overlay A Vertical End Register
PIXC_TRANSP_A	Overlay A Transparency Ratio Register
PIXC_HSTART_B	Overlay B Horizontal Start Register
PIXC_HEND_B	Overlay B Horizontal End Register
PIXC_VSTART_B	Overlay B Vertical Start Register
PIXC_VEND_B	Overlay B Vertical End Register
PIXC_TRANSP_B	Overlay B Transparency Ratio Register
PIXC_IRQSTAT	Interrupt Status Register
PIXC_CONRY	RY Conversion Component Register
PIXC_CONGU	GU Conversion Component Register
PIXC_CONBV	BV Conversion Component Register
PIXC_CCBIAS	Conversion Bias Register
PIXC_TC	Transparency Color Register

ADSP-BF60x PIXC Interrupt List

Table 32-2: ADSP-BF60x PIXC Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
PIXC0 Channel 0 DMA	107	35	LEVEL

Table 32-2: ADSP-BF60x PIXC Interrupt List Interrupt List (Continued)

Description	Interrupt ID	DMA Channel	Sensitivity
PIXC0 Channel 1 DMA	108	36	LEVEL
PIXC0 Channel 2 DMA	109	37	LEVEL
PIXC0 Status	110		LEVEL

ADSP-BF60x PIXC Trigger List

Table 32-3: ADSP-BF60x PIXC Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
PIXC0 Channel 0 DMA	57	PULSE/EDGE
PIXC0 Channel 1 DMA	58	PULSE/EDGE
PIXC0 Channel 2 DMA	59	PULSE/EDGE

Table 32-4: ADSP-BF60x PIXC Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
PIXC0 Channel 0 DMA	57	
PIXC0 Channel 1 DMA	58	
PIXC0 Channel 2 DMA	59	

ADSP-BF60x PIXC DMA List

Table 32-5: ADSP-BF60x PIXC DMA List DMA Channel List

Description	DMA Channel
PIXC0 Channel 0 DMA	DMA35
PIXC0 Channel 1 DMA	DMA36
PIXC0 Channel 2 DMA	DMA37

PIXC Definitions

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

NTSC/PAL

The two most common video standards used. NTSC is the video system or standard used in North America and most of South America. In NTSC, 30 frames are transmitted each second. Each frame is made up of 525 individual scan lines. PAL is the predominant video system or standard mostly used overseas. In PAL, 25 frames are transmitted each second. Each frame is made up of 625 individual scan lines.

RGB888 format

RGB is a color space where pixels are defined by three color values; one red (R), one green (G) and one blue (B). The suffix signifies the bit widths for these color components. In this case, RGB888 means that each red, green and blue value is 8-bits each.

YUV 4:2:2 interleaved format

YUV is a color space where pixels are defined by a luminance (Y) component and chrominance (UV) components. The suffix signifies how the chrominance components have been decimated as well as formatting. In this case, the YUV422 format has the chrominance decimated by two, meaning only half of each chrominance component are available. Typical YUV422 formatting interleaves the luminance and chrominance such as U1Y1V1Y2U2Y3V2Y4.

Data Overlay

Process of blending or replacing a pixel from an image with another pixel.

PIXC Block Diagram

A top-level micro architecture diagram of the PIXC appears in the following figure. As shown in the figure, the PIXC uses three DMA channels: one for the image data, one for the overlay data and one for storing the results back to memory. Frame C (output frame) can also be fed back to the PIXC for multiple stages of processing, taking the place of frame A (main input image) when this happens. The main input image and output frame can alternatively be piped from or to another peripheral via the video subsystem, respectively.

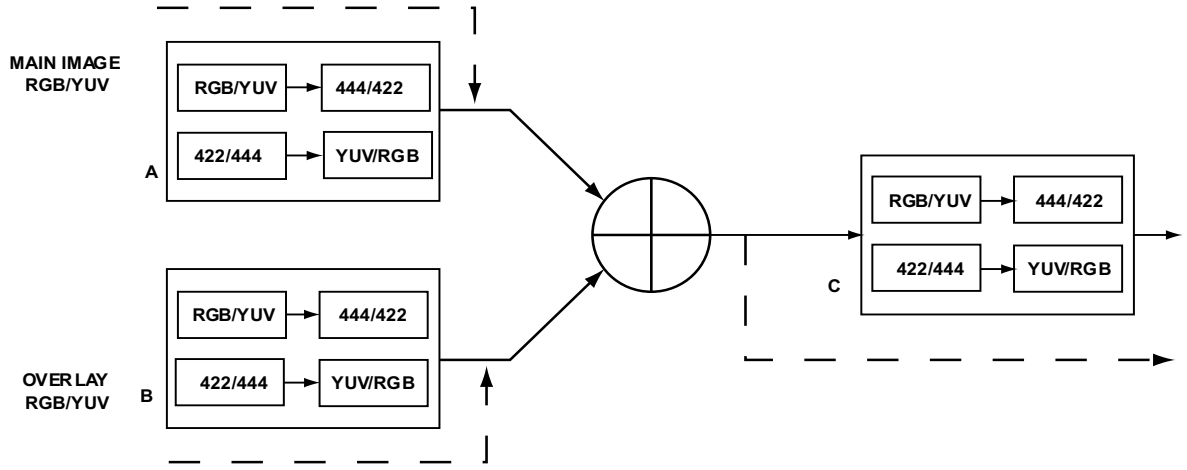


Figure 32-1: Pixel Compositor Top-Level Diagram

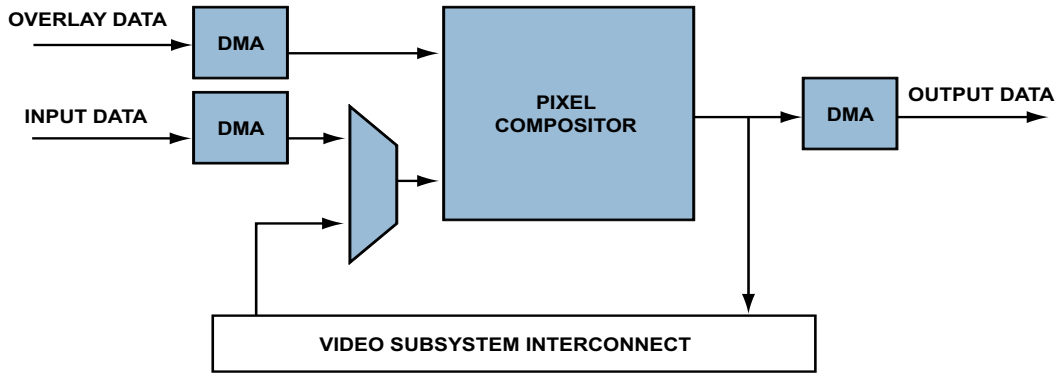


Figure 32-2: PIXC Functional Block Diagram

PIXC Architectural Concepts

The PIXC data interface can be configured to obtain the input main image from memory via DMA or piped from a different peripheral configured by the video subsystem. The output result image can also be sent back to memory via DMA or piped to a peripheral. Also, the output can be fed back into the PIXC for multiple stages of processing. The overlay image data must be brought in from memory via DMA. The following topics provide additional information.

- *Pixel Pipe (PxP) Interface*
- *Start Synchronization in PxP Input Mode*
- *DMA Interface*

Pixel Pipe (PxP) Interface

The PIXC supports a pixel pipe (PxP) interface with inputs and outputs. The input side is multiplexed with the DMA channel carrying the input image data from memory. The video subsystem connectivity determines whether the PIXC image data source is connected to the IFIFO or the pixel pipe. The output composite image/color converted image can be either transferred to memory via DMA or to the PVP/EPPI via the PxP. This is also determined by the video subsystem connectivity.

If the PIXC image input is programmed to connect to the PxP bus:

1. The PIXC expects only complete frames from the PxP. Each line should contain the exact number of pixels as programmed in the `PIXC_PPL` register and each frame should contain as many lines as programmed in the `PIXC_LPF` register. The EPPI is the only source of data when the PIXC input connects to the PxP. The EPPI ensures that once a frame starts, no further frame sync is passed on the PxP bus until the frame is complete.
2. Internal signaling is used to start the pixel processing in the PIXC. The `PIXC_PPL` counter increments at pixel valid. The `PIXC_LPF` counter is incremented only when the `PIXC_PPL` counter rolls over and resets to 1 when the maximum `PIXC_LPF` count is reached (as programmed in the `PIXC_LPF` register).

Start Synchronization in PxP Input Mode

When the PIXC is configured to take image data from the PxP bus, the PIXC internal logic synchronizes itself to the incoming PxP frame sync after it is enabled. The PIXC ignores all incoming data until it receives this PxP frame sync from the input PxP bus. After the internal frame sync is recognized by the PIXC, its engine starts processing the incoming data. This removes any restriction of the relative enable times of the PIXC and the EPPI. Either could be enabled first. If the EPPI is enabled first and the PIXC is enabled later, the PIXC starts its frame processing only at the start of the next frame after the PIXC is enabled.

DMA Interface

The PIXC has a native DMA controller with two input channels and one output channel. One input channel is used to transfer the input image to the PIXC from memory and the other is used to transfer the overlay image to the PIXC from memory. The output channel transfers the blended data to memory. Each channel is a DDE instance.

The two input DMA channels with 32-bit bus widths take the image data and overlay graphics/text data from their buffers into two separate FIFO's where the data is then unpacked. Each of these FIFO buffers is 32 bits wide and contains 8 entries.

In the blender, (8-bit) pixel elements from the two buffers are mixed together. One dedicated DMA channel transfers the combined pixel data back to memory. A local arbiter arbitrates between the three channels. This arbitrated request is forwarded to the system crossbar. The PIXC only has one connection to the system crossbar.

When using the DMA to interface to the PIXC, at least two DMA channels should be enabled and configured appropriately—the image DMA channel and the output DMA channel. Furthermore, when the overlay function is used, the overlay DMA channel should also be configured and enabled.

PIXC Data Overlay

Overlay is an optional function. If it is disabled, all overlay functionality is bypassed, and a single data stream from the main image data buffer goes directly to the image output buffer, after an optional format conversion. If it is enabled, the blender combines the pixel data from the two image input buffers.

The overlay image is located in a user-defined rectangle within the main image and, in most cases, the overlay image is smaller than the main image. The figure below illustrates an example of the main and overlay image regions on a screen, where a foreground triangle overlay sits on top of the main image in the background. Although the figure does not show this explicitly, H-Start and V-Start can equal (0,0).

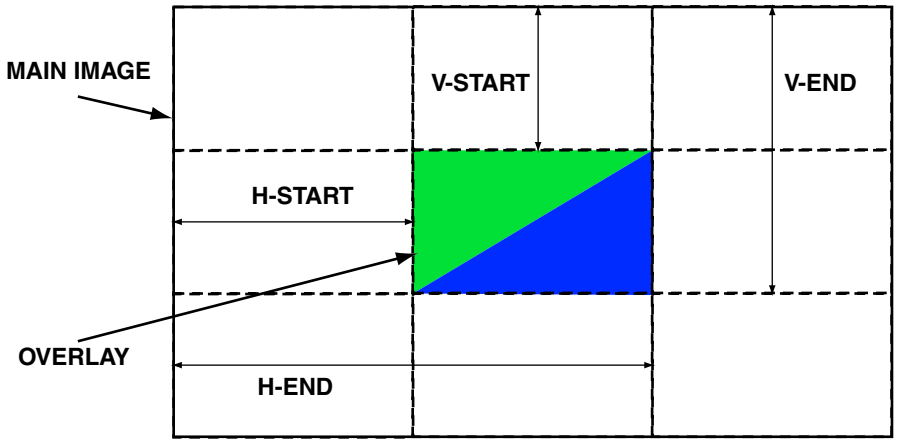


Figure 32-3: Main Image and Overlay Image Region

In certain situations, it may be beneficial to use DMA only on the region of the main image that is affected by the overlay instead of bringing in the entire main image. For example, in a situation where the intent is to overlay an image over the main image and to store the result back over the main image, a program can set up a 2D-DMA to only bring in the area of the main image that is affected by the overlay. This may reduce the amount of DMA activity and potentially improve system performance.

There is a set of an additional registers that can be used to specify a second overlay region, so that two separate overlay blocks can be defined simultaneously. Furthermore, either or both of these overlay coordinate register sets can be enabled or disabled at one time because separate enable bits (`PIXC_CTL.OVENA` and `PIXC_CTL.OVENB`) exist in the PIXC control register for each of the overlay register sets.

If more than two overlay blocks are needed, the two sets of overlay registers must be managed by the program to perform the additional overlays. This can be done using an interrupt service routine, where the interrupt from the PIXC is used to re-program the overlay coordinate registers.

The PIXC can generate an interrupt under two conditions, at the end of the last valid overlay and at the end of a frame.

Either of these interrupts can be enabled or disabled. However, the PIXC only has one interrupt line output, so it raises an interrupt (under the appropriate condition) when either of these two interrupts is triggered. If both interrupts are enabled, the interrupt status register of the PIXC indicates which of the two conditions caused the interrupt to occur. Once the PIXC generates an interrupt, it stalls the pixel processing until a software ISR clears the interrupt. However, the FIFOs do not stall and keep filling up even when the PIXC is in a stalled state. Both interrupts can be cleared by writing a 1 to the respective interrupt status bits.

After each interrupt (whether it is a last-valid-overlay interrupt or an end-of-frame interrupt), the PIXC restarts processing with coordinate register set A. In other words, at the time of clearing the interrupt:

- If coordinate set A is enabled (PIXC_CTL.OVENA = 1), the PIXC assumes that the first incoming data over the bus is to be overlaid on the area specified in coordinate set A.
- If coordinate set A is disabled (PIXC_CTL.OVENA = 0), and coordinate set B is enabled (PIXC_CTL.OVENB = 1), the PIXC assumes that the first incoming data over the data bus is to be overlaid on the area specified in coordinate set B.
- If both coordinate sets are disabled, the PIXC flushes the overlay FIFO and make no more data requests on the overlay DMA channel.

NOTE: The overlay enable bits PIXC_CTL.OVENA and PIXC_CTL.OVENB should only be changed inside the interrupt service routines of the PIXC interrupts, or when the overlay block is disabled.

Note that the module enable bit (PIXC_CTL.EN) is the root enable for the PIXC. Both the PIXC_CTL.OVENA and PIXC_CTL.OVENB bits are gated with PIXC_CTL.EN bit, so if PIXC_CTL.EN = 0, the individual overlay enable bits have no effect, and the module remains disabled. When PIXC_CTL.EN = 1, both the image and overlay FIFOs are flushed and no more DMA requests are made on either of the DMA channels.

Once the DMAs are enabled, the PIXC keeps track of the current pixel being displayed from the main image data by reading from the PIXC_PPL and PIXC_LPF programmable registers.

When the pixel count reaches the top left corner (H-Start, V-Start) of overlay data, the PIXC starts the overlay. When the pixel count reaches the top right corner (H-End, V-Start) of overlay data, the PIXC stops the overlay. It starts again at the next line at (H-Start, V-Start + 1) and stop at (H-End, V-Start + 1), and so on until the entire overlay frame is processed.

NOTE: Internally, the start of the overlay DMA is preempted by the PIXC before the actual processing of the first overlay pixel, and DMA data is requested until the overlay FIFO is full. Similarly, the overlay DMA does not stop at the end of a line. The overlay FIFO continues to be filled with DMA data, even when the current pixel is not an overlay pixel, but the supply of overlay pixels from the overlay FIFO is simply halted.

The PIXC decides whether or not to perform overlay mixing for the current pixel by using the various PIXC register values as follows.

- The PIXC_PPL and PIXC_LPF registers must be programmed correctly (and cannot be 0).
- The PIXC_HSTART_A and PIXC_HEND_A must be less than or equal to PIXC_PPL.

- The PIXC_VSTART_A and PIXC_VEND_A must be less than or equal to PIXC_LPF.

Programs can define multiple rectangular regions covering several separate overlays, using the same number of DMA descriptors, where each DMA descriptor corresponds to an overlay region.

Multiple overlay regions are split into the two following cases.

- Overlay regions with no horizontal overlap – Software can maintain separate areas in memory for both overlay regions, with separate H-Start, V-Start, H-End, and V-End coordinates for each region. After the first overlay is completed, the DMA chain pointer can load the next overlay parameters (index, count, and modifier) to the DMA registers of the corresponding DMA channel.

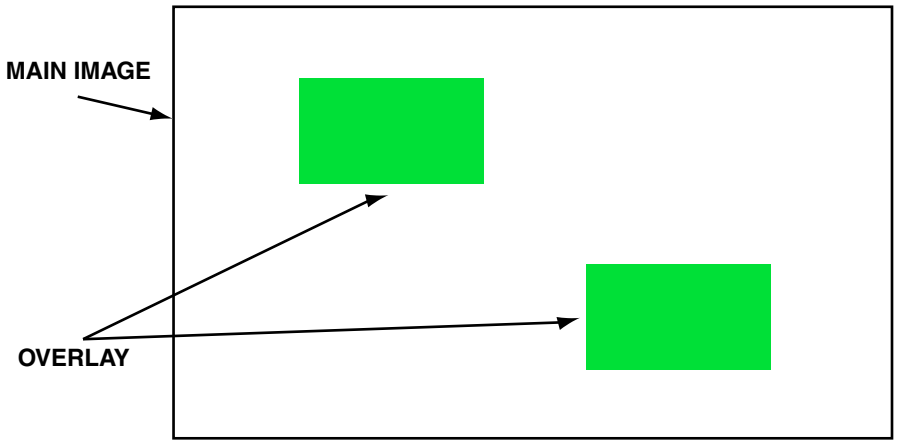


Figure 32-4: Overlay Regions With No Horizontal Overlap

- Overlay regions with horizontal overlap –Software has to maintain a combined overlay region in memory. This includes some in-between area where there is no overlay. This region of memory has to be filled with the transparent color value (explained below). The H-Start, V-Start, H-End and V-End coordinates contain the values of the combined overlay region.

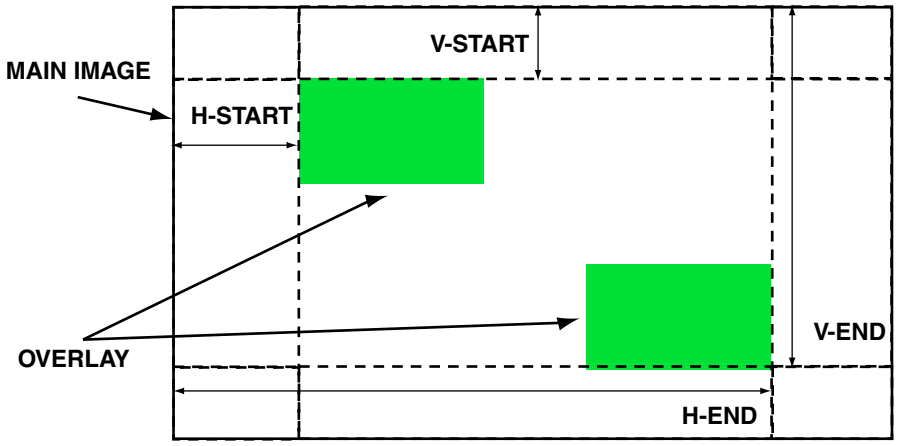


Figure 32-5: Overlay Regions With Horizontal Overlap

PIXC Transparency Control

When the overlay function is enabled, each overlay pixel is combined with each main image pixel to generate the displayed output pixel. Each pixel combination is controlled by a transparency ratio value alpha (α), a 4-bit value that determines the proportion of overlay and main image that contribute to the output pixel. The pixel combination algorithm can be expressed as:

$$C = \frac{B(\alpha + 1)}{16} + \frac{A(15 - \alpha)}{16}$$

The table lists the multiplying factors for various α values.

- A = 8-bit pixel data in main frame buffer *(background*)
- B = 8-bit pixel data in overlay buffer (*foreground*)
- C = 8-bit combined pixel data
- α = Transparency ratio code, which is a 4-bit value present in a memory-mapped register

Table 32-6: Multiplying Factors for Various a Values

α	Overlay Multiplying Factor	Image Multiplying Factor
0	1/16	15/16
1	2/16	14/16
2	3/16	13/16
3	4/16	12/16
4	5/16	11/16
5	6/16	10/16
6	7/16	9/16
7	8/16	8/16
8	9/16	7/16
9	10/16	6/16
10	11/16	5/16
11	12/16	4/16
12	13/16	3/16
13	14/16	2/16

NOTE: Passing the image alone can be achieved by disabling the overlay function.

Rounding is performed at the output of the blender, which rounds the combined pixel data to the nearest integer value.

PIXC Transparency Color

A transparent color is a specific color that is removed from one image to reveal another *behind* it. This technique is also referred to as chroma keying. The principal subject is photographed or filmed against a background having a single color, usually in the blue or green spectrum. When the phase of the chroma signal corresponds to the pre-programmed state associated with the background color(s) behind the principal subject, the signal from the alternate background (which in this case comes from the main image channel) is inserted in the composite signal and presented at the output.

When the phase of the chroma signal deviates from that associated with the background color(s) behind the principal subject, the picture data associated with the principal subject (in this case, the overlay image) is presented at the output. The figure below illustrates this concept.

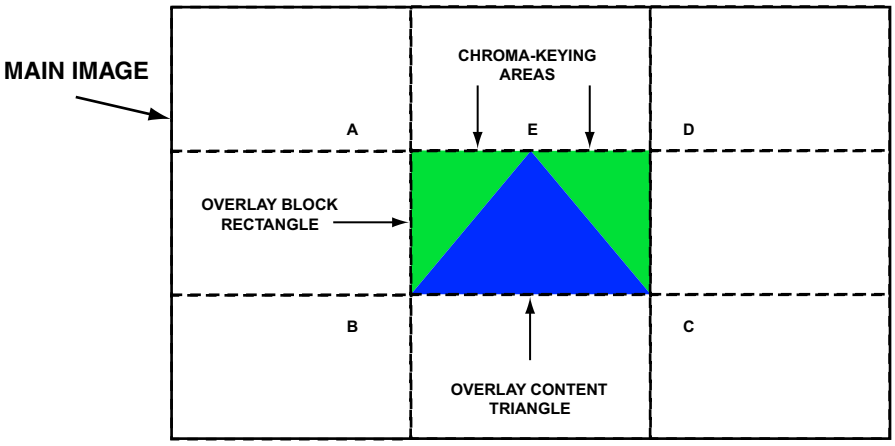


Figure 32-6: Transparent Color (Chroma Keying)

In order to display the main image in the two triangle areas ΔABE and ΔCDE in overlay block ABCD, the data in the overlay buffer corresponding to the pixels in the triangle areas ΔABE and ΔCDE must hold a specific value, called the transparent color.

The PIXC provides a 24-bit MMR (storing three 8-bit color components) for each of the two overlay blocks to designate a particular RGB or YUV value as the transparent color. The transparent color must be in the same format (YUV 4:2:2 or RGB888) as the overlay data, regardless of whether or not a color space conversion is present in the overlay data path. The PIXC then compares each input pixel value on the overlay channel with this transparent color. If there is a match, the overlay pixel at this location is ignored by the blender, and the main image pixel at that location is assigned 100% weight.

NOTE: If YUV 4:2:2 is the overlay channel input data format, artifacts may occur at the edge of the transparent color region. In this case, it is preferable to set the `PIXC_CTL.UDSMOD` bit to 0 (duplicating-dropping mode), in order to get better control of the U and V components at the edge of the transparent color region.

Color Space Conversion

Depending on the input data format and display device used, there may be a color space conversion performed on the data stream of the PIXC. If the input data is in YUV format, a YUV-to-RGB conversion can be performed for output to an LCD panel. If the input data is in RGB format, a RGB-to-YUV conversion can be performed for output to NTSC/PAL displays. The color space conversion may happen on any of the three paths (for example, the main image data path, the overlay image data path, or the combined data path). Registers are used to specify the input, overlay and output formats. The color space converter block controls a number of cases of operation.

- Both the image and the overlay data are in the same format.
- The image and the overlay data are in different formats.
- Color space conversion only

Each case is described below, along with several special usage cases. Note that various scenarios may be shown in the same figure based on the output device chosen, though only a single output destination is supported at one time.

Case 1 - Image and Overlay in the Same Format

Both input data streams (main image and overlay) are in the same format, either YUV 4:2:2 or RGB888, so a color space conversion may be performed after alpha blending, depending on the output type.

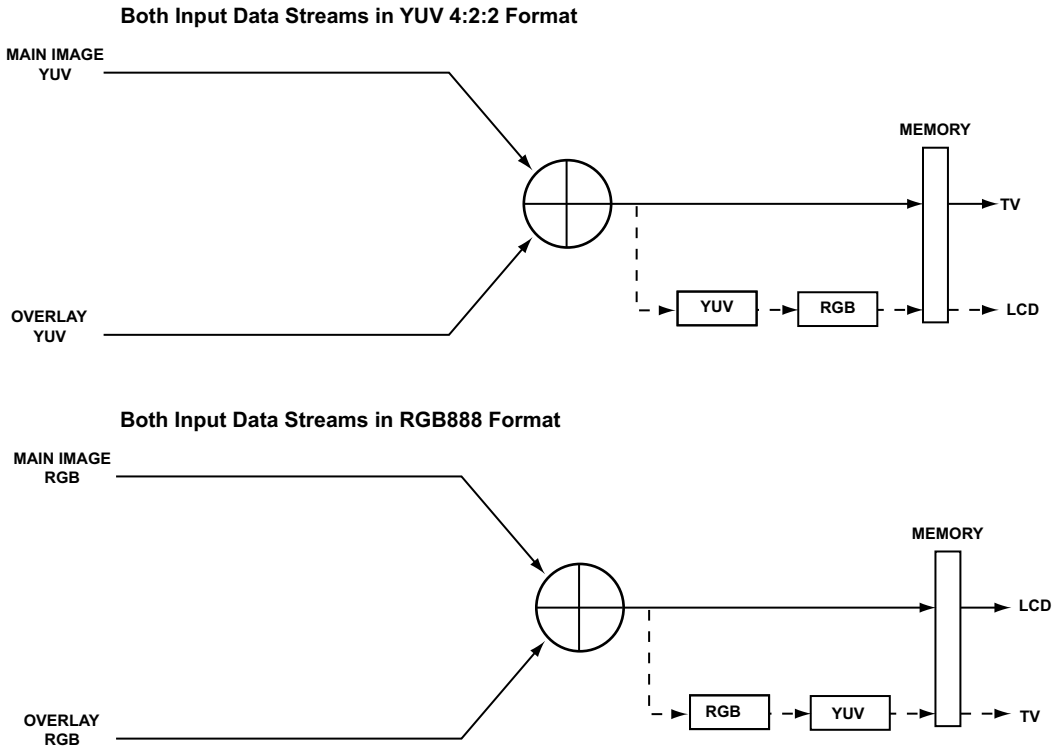


Figure 32-7: Image and Overlay in the Same Format

Case 2 - Image and Overlay in Different Formats

In this case, the two input data streams are not in the same format. The PIXC has to perform a color space conversion on either the main input stream or the overlay input stream (depending on the required output format) before alpha blending can take place.

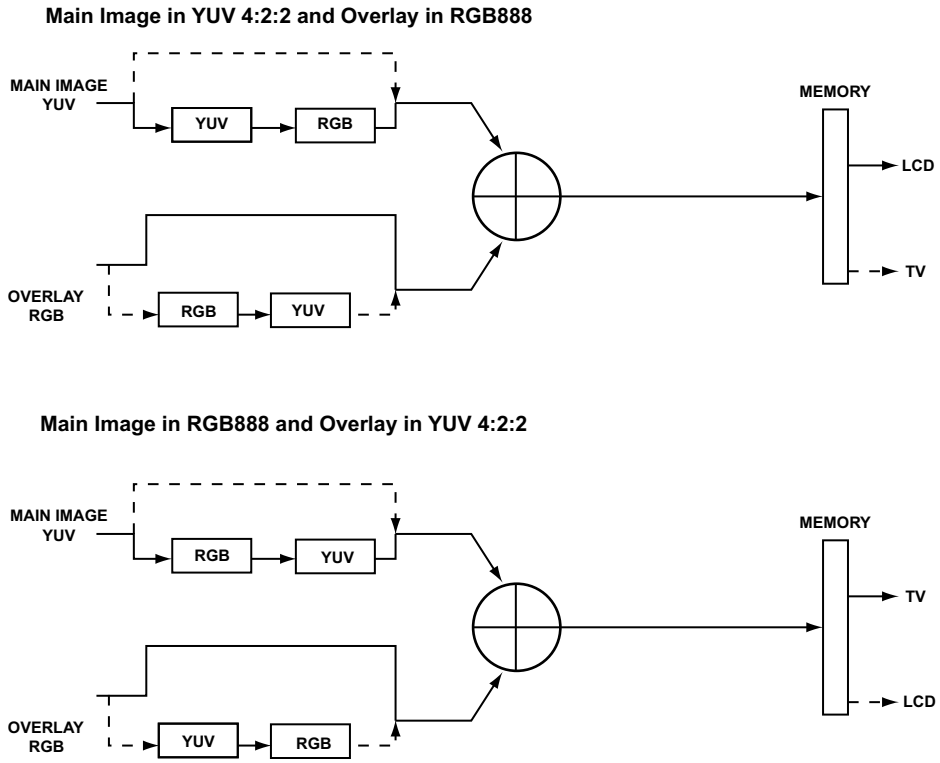


Figure 32-8: Image and Overlay in Different Formats

Case 3- Color Space Conversion Only

In this case, there is no overlay blending. The main image is brought into the PIXC, the color space converted, and then sent back to memory.

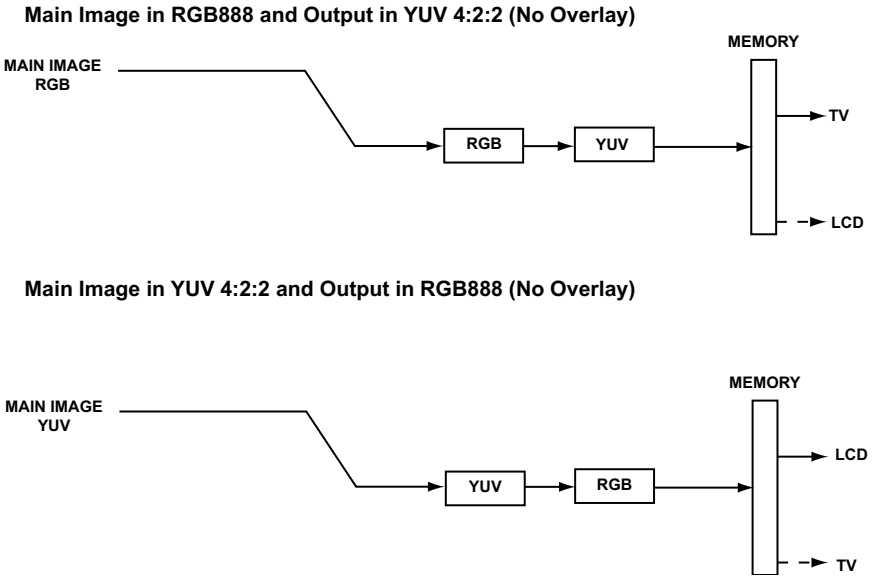


Figure 32-9: Color Space Conversion Only

NOTE: For this mode, the register settings are: `PIXC_CTL.EN`=1, `PIXC_CTL.OVENA`=0 and `PIXC_CTL.OVENB`=0.

PIXC Operating Modes

The color space converter block provides conversion operations in the following situations.

- *PIXC Mode Case 1 - Image/Overlay in the Same Format*
- *PIXC Mode Case 2 - Image/Overlay in Different Formats*
- *PIXC Mode Case 3 - Color Space Conversion Only*

Note that in this section various conversion operations may be shown in the same figure based on the output device chosen, though only a single output destination is supported at one time.

PIXC Mode Case 1 - Image/Overlay in the Same Format

Both input data streams (main image and overlay) are in the same format, either YUV 4:2:2 or RGB888, so a color space conversion may be performed after alpha blending, depending on the output type. See figures below.

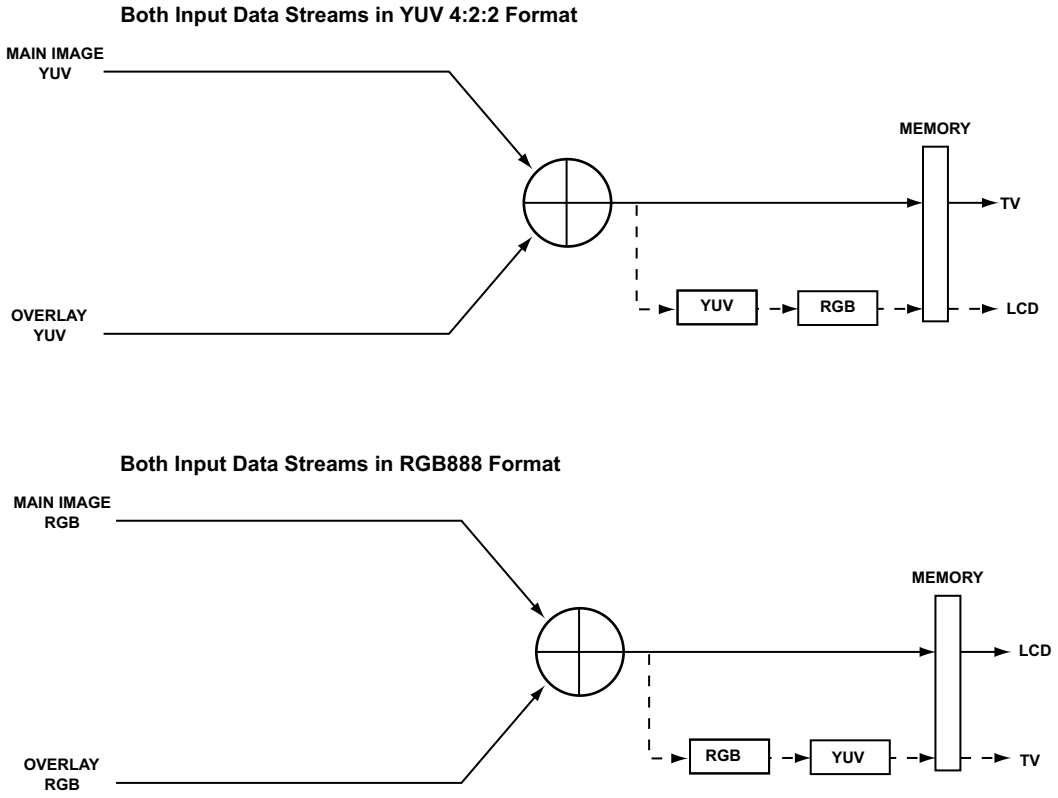


Figure 32-10: Image and Overlay in the Same Format

PIXC Mode Case 2 - Image/Overlay in Different Formats

In this case, the two input data streams are not in the same format. The PIXC has to perform a color space conversion on either the main input stream or the overlay input stream (depending on the required output format) before alpha blending can take place. See figures below.

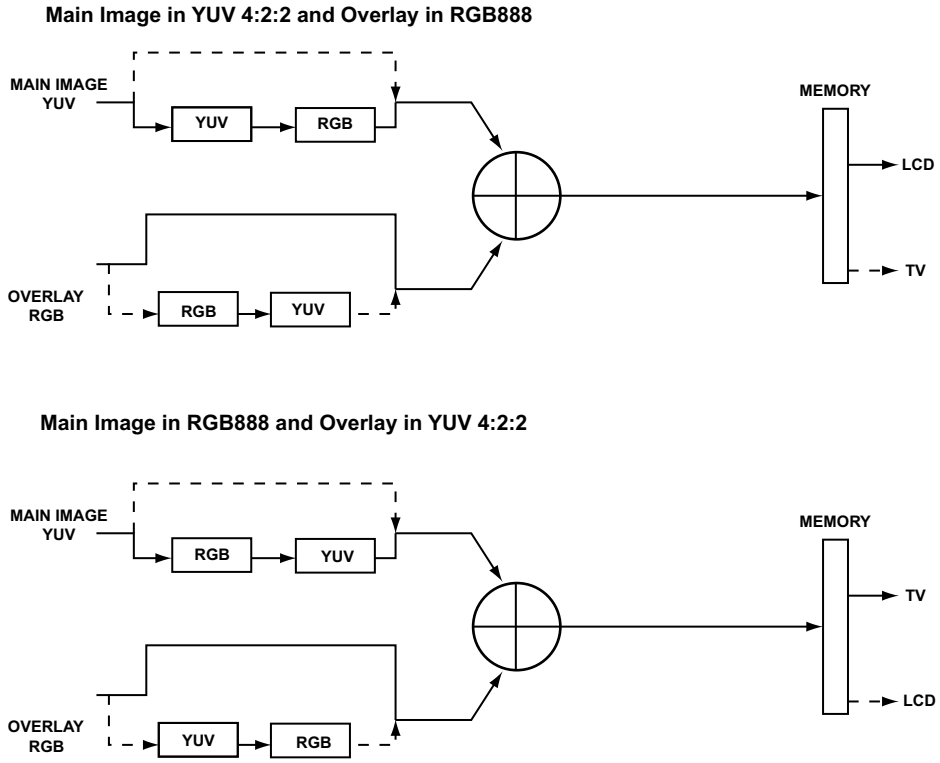


Figure 32-11: Image and Overlay in Different Formats

PIXC Mode Case 3 - Color Space Conversion Only

In this case, there is no overlay blending. The main image is brought into the PIXC, the color space converted, and then sent back to memory. See figures below.

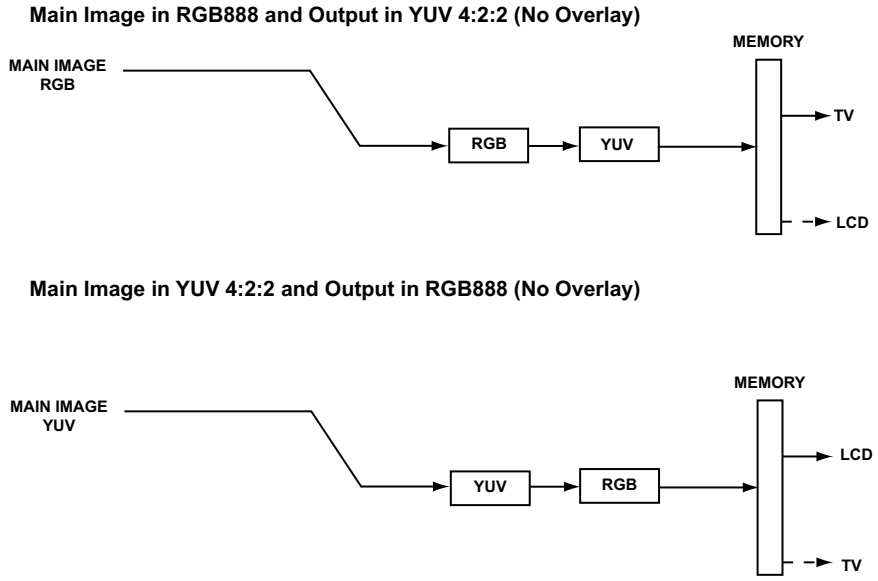


Figure 32-12: Color Space Conversion Only

For this mode, the register settings are: `PIXC_CTL.EN` = *1,* `PIXC_CTL.OVAEN` = *0 and* `PIXC_CTL.OVBEN` = *0.*

Image/Overlay/Format Actions

The table below lists the PIXC actions that take place based on any possible combination of image, overlay, and output data formats. The definitions used in the table are as follows.

- CSC=Color Space Conversion
- US=Up sampling
- DS=Down Sampling
- YUV=YUV 4:2:2 format
- RGB=RGB888 format

Table 32-7: PIXC Actions

Image Data Format	Overlay Data Format	Output Data Format	PIXC Actions
YUV	No Overlay	RGB	US followed by CSC
RGB	No Overlay	YUV	CSC followed by DS
YUV	YUV	YUV	US in both paths followed by DS before output
YUV	RGB	RGB	US in image path, CSC in image path
YUV	YUV	RGB	US in both paths, followed by CSC
YUV	RGB	YUV	CSC in overlay path, US in image path, DS before output
RGB	YUV	YUV	CSC in image path, US in overlay path, DS before output
RGB	YUV	RGB	US in overlay path, CSC in overlay path
RGB	RGB	YUV	CSC followed by DS
RGB	RGB	RGB	No CSC, No US, No DS

Image/Overlay/Format Recommendations

For best results, the overlay should start on an odd-numbered pixel so that the U and V components of the image and the overlay are aligned. Otherwise artifacts may occur in the combined image.

When both the image and the overlay are in YUV 4:2:2 format and the output is also in YUV 4:2:2 format, the duplicating-dropping mode (`PIXC_CTL.UDSMOD` bit) is used to prevent a low-pass filtering effect on the images.

PIXC Image/Overlay/Format Special Use Cases

There are ways by which the PIXC can be made to operate on certain data formats that it does not support in any standard modes. For example, YUV 4:4:4 is similar to RGB 888 with respect to the number of pixels per 32-bit DMA word. So the PIXC can be configured to work with the YUV 4:4:4 data format by intelligently programming the IFRMT, OVFRMT, and OUTFRMT bit fields in the PIXC_CTL register and the color space conversion coefficients.

These special usage cases are shown in the sections that follow.

Example 1 - YUV 4:2:2 to YUV 4:4:4 or LCD/RGB

See the figures for special use of this mode:

- IFRMT = YUV
- OVFRMT = YUV
- OUTFRMT = RGB

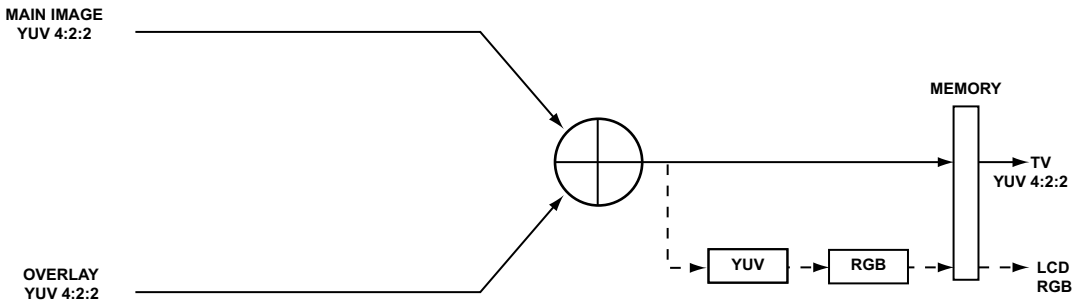


Figure 32-13: YUV 4:2:2/YUV 4:2:2 to YUV 4:2:2 or LCD/RGB

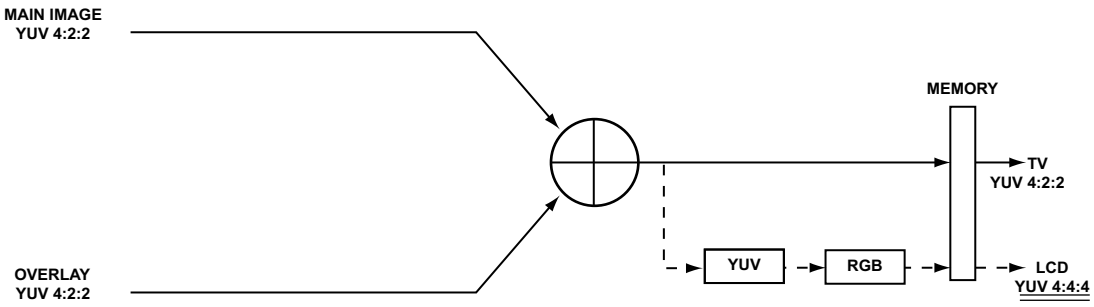


Figure 32-14: YUV 4:2:2/YUV 4:2:2 to YUV 4:2:2 or YUV 4:4:4

Example 2 - YUV 4:4:4 to YUV 4:4:4 or YUV 4:2:2

See the figures for special use of this mode.

In the special usage of this mode, YUV 4:4:4 input produces a blended YUV 4:2:2 or YUV 4:4:4 data stream. A CSC matrix with coefficients of 1 is needed.

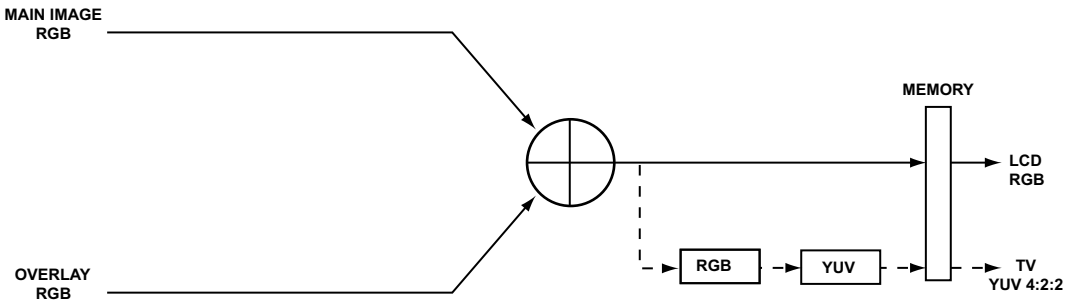


Figure 32-15: RGB/RGB to LCD/RGB or YUV 4:2:2

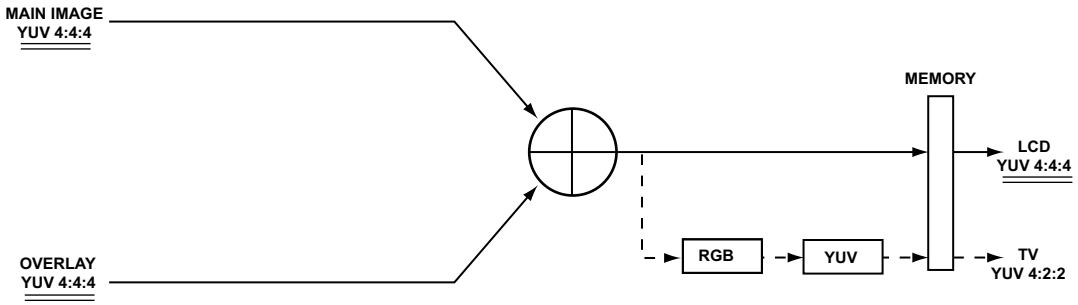


Figure 32-16: YUV 4:4:4/YUV 4:4:4 to YUV 4:4:4 or YUV 4:2:2

Example 3 - YUV 4:2:2/4:4:4 to YUV 4:4:4 or YUV 4:2:2

See the figures for special use of this mode.

In the special usage of this mode, a YUV 4:4:4 input stream and a YUV 4:2:2 input stream can be blended to produce either a YUV 4:4:4 or a YUV 4:2:2 output stream.

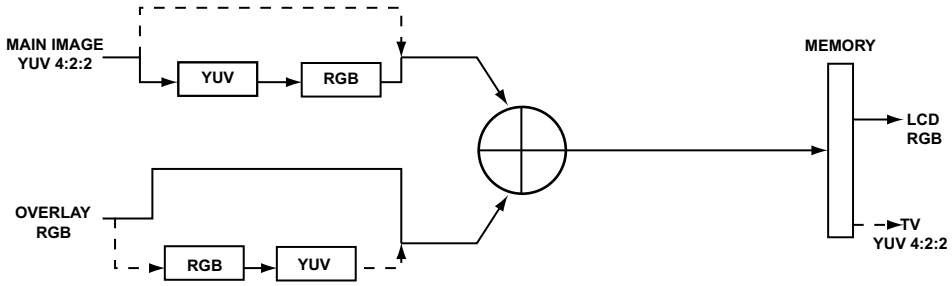


Figure 32-17: YUV 4:2:2/RGB to LCD/RGB or YUV 4:2:2

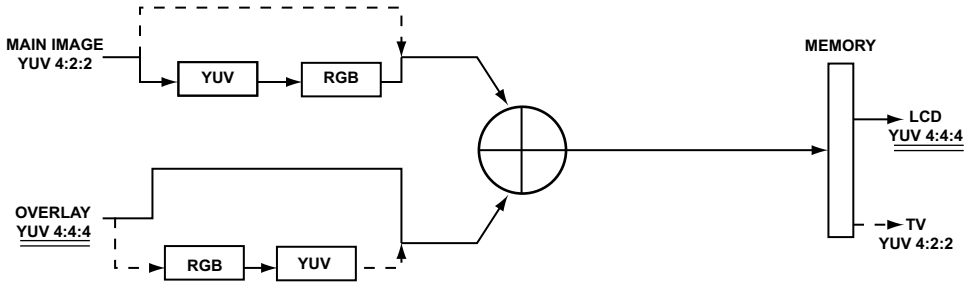


Figure 32-18: YUV 4:2:2 / YUV 4:4:4 to YUV 4:4:4 or YUV 4:2:2

Example 4 - YUV 4:4:4 to YUV 4:2:2

See the figures for special use of this mode:

- PIXC_EN = 1
- OVR_A_EN = 0
- OVR_B_EN = 0
- IMG_FORM = RGB
- OUT_FORM = YUV
- All CSC coefficients = 1

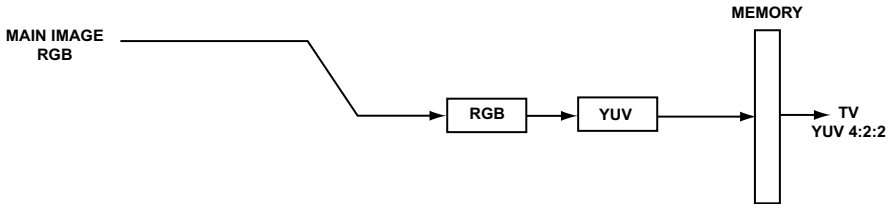


Figure 32-19: RGB to YUV 4:2:2

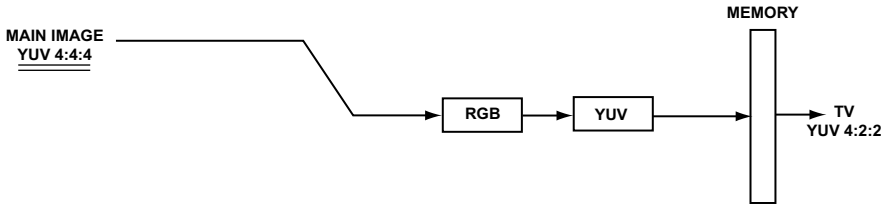


Figure 32-20: YUV 4:4:4 to YUV 4:2:2

Color Space Conversion Matrix Equations

The PIXC color space conversion block implements the following matrix equation.

$$K \times \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix} + \begin{bmatrix} A_{14} \\ A_{24} \\ A_{34} \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ C_2 \end{bmatrix}$$

The A_{xx} coefficients are 10-bit signed values represented in two's complement format. $A_{11}…A_{33}$ are coefficient multipliers (for most cases, it is sufficient to specify these as integers between –512 and 511), and A_{14}, A_{24}, A_{34} are simply offsets added to the result for each row. B_1, B_2, B_3 represent the input pixel component values (for example, YUV or RGB) and C_1, C_2, C_3 are the output pixel component values. Output pixel values are rounded to the nearest integer

The constant K equals 1/512. For example, to set A_{11}'s effective value to 0.299, this coefficient's MMR should be programmed to ROUND(.299*512), or 153. If a coefficient needs to be programmed with a value greater than 1, an extra bit exists in each coefficient's MMR to specify if an extra multiply by 4 must be performed after multiplying the input value by its coefficient. However, this setting can only be specified for an entire row, so if this bit is set, all the coefficients for that row (Ax1–Ax3) should be calculated as ROUND (coeff × 512/4). In other words, the constant K effectively becomes 1/128 for that row.

For reference, the matrix equations representing conversion between YUV and RGB formats are:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.168 & -0.330 & 0.498 \\ 0.498 & -0.417 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.000 & 1.397 \\ 1.000 & -0.343 & -0.711 \\ 1.000 & 1.765 & 0.000 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix} + \begin{bmatrix} -179 \\ 135 \\ -226 \end{bmatrix}$$

NOTE: For YUV-to-RGB conversion, the PIXC expects the input data to be arranged in the following order: VYUY, VYUY, and so on. As a result, if the input data is instead arranged as UYVY, UYVY, and so on, then, the columns Ax2 and Ax3 of the coefficient matrix are swapped.

For RGB-to-YUV conversion, the PIXC arranges the output data by default in the following order: VYUY, VYUY, and so on. If the output data is desired to instead be arranged as UYVY, UYVY, and so on, then, rows A2x and A3x of the coefficient and bias matrices are swapped.

Color Space Converter Output Thresholds

Each PIXC output sample is 8 bits wide, whether it is an R, G, B, Y, U or V component value. Therefore, any output sample must be in the 0 to 255 range. Since all the coefficients are programmable, some of the inputs, when operated upon by the coefficients, may produce an output outside the 0 to 255 range. In such cases, the PIXC clips the output component's value to 0 or 255.

YUV Re-Sampling

When the color space converter operates between two color spaces, it requires all components of each pixel to be present in the data stream.

Therefore, the PIXC internally up samples the YUV 4:2:2 data stream before a YUV-to-RGB conversion and similarly down samples the YUV 4:4:4 data stream after a RGB-to-YUV conversion. The re-sampling always takes place between YUV 4:2:2 and YUV 4:4:4 formats, but a certain flexibility is provided with regard to how the re-sampling is done by the PIXC in each case.

Up sampling – A YUV 4:2:2-to-YUV 4:4:4 conversion can be performed either by averaging or by duplicating the pixel components. The PIXC_CTL.UDSMOD bit specifies the up sampling mode. The default setting of this bit is 0, which duplicates the chroma components (Us and Vs) from the odd pixels to the even pixels as shown below.

YUV 4:2:2 input:	V1Y1, U1Y2, V3Y3, U3Y4, ...
YUV 4:4:4 conversion:	Y1U1V1, Y2U1V1, Y3U3V3, Y4U3V3, ...

Setting the PIXC_CTL.UDSMOD bit to 1 enables the averaging of the chroma components of the preceding and succeeding pixels to obtain the intermediate chroma value. In other words, two consecutive odd-numbered pixel's chroma components are averaged to obtain the intermediate even-numbered pixel's chroma components as shown below.

YUV 4:2:2 input:	V1Y1, U1Y2, V3Y3, U3Y4, ...
YUV 4:4:4 conversion:	Y1U1V1, Y2U2V2 [U2=(U1+U3)/2, V2=(V1+V3)/2], Y3U3V3, Y4U4V4 [U4=(U3+U5)/2, V4=(V3+V5)/2], ...

If the sum of the preceding and succeeding pixel's U/V components is an odd number, the average is rounded down (truncated to an integer value).

Since the last pixel on a line is always an even-numbered pixel, the last odd pixel value on that line is used as the last even pixel value during up sampling.

Down Sampling – A YUV 4:4:4-to-YUV 4:2:2 conversion can be performed either by averaging or by dropping the pixel components. The PIXC_CTL.UDSMOD bit also governs the down sampling mode. Setting the PIXC_CTL.UDSMOD bit to 0 (default) enables the dropping of the chroma components of the even numbered pixels as shown below.

YUV 4:4:4 input:	Y1U1V1, Y2U2V2, Y3U3V3, Y4U4V4, ...
YUV 4:2:2 conversion:	V1Y1, U1Y2, V3Y3, U3Y4, ...

YUV 4:4:4 input:	Y1U1V1, Y2U2V2, Y3U3V3, Y4U4V4, ...

YUV 4:2:2 conversion:	V12Y1, U12Y2, V34Y3, U34Y4, ...
	[U12=(U1+U2)/2, U34=(U3+U4)/2]
	[V12=(V1+V2)/2, V34=(V3+V4)/2]

Supported Data Formats

For the implementation of overlay, the PIXC needs two input data streams from two separate data buffers— a main image buffer and an overlay buffer. The data in the main image buffer can be formatted as YUV 4:2:2, RGB888, RGB666 or RGB565 while the overlay data must be in YUV 4:2:2 or RGB888 format. The main image data and the overlay data can be in different formats. The output data can also be formatted as YUV 4:2:2, RGB888, RGB666 or RGB565. The formats are described in detail in the following sections.

- *Operation in YUV 4:2:2 Format*
- *Operation in RGB888 Format*
- *Operation in RGB565 Format*
- *Operation in RGB666 Format*
- *Operation with RGB656 and RGB666 Formats*

Operation in YUV 4:2:2 Format

Each Y/U/V component is stored in 8 bits of data. The PIXC only accepts a YUV 4:2:2 interleaved format, in the following sequence:

V1, Y1, U1, Y2, V3, Y3, U3, Y4 ...

(Two components with the same suffix number (for example, V1 and U1) implies that they are extracted from the same pixel.)

It is the user's responsibility to ensure that the YUV source data to the PIXC is in the correct interleaved format. Therefore, data processing may be necessary in order to meet this requirement.

The figures below illustrate correct PIXC input buffer structure and data stream format.

YUV 4:2:2 Data in an Interleaved Data Buffer Structure

ADDR_BASE + 0	V1 1	Y1 1	U1 1	Y1 2		U1 639	Y1 640
ADDR_BASE + 1280	V2 1	Y2 1	U2 1	Y2 2		U2 639	Y2 640
	⋮	⋮	⋮	⋮		⋮	⋮
ADDR_BASE + (479 x 1280)	V480 1	Y480 1	U480 1	Y480 2		U480 639	Y480 640

YUV 4:2:2 Expected Data Stream Format to PIXC

B31	B16	B15	B0
Y2	U1	Y1	V1
Y4	U3	Y3	V3
...	...	...	...
...	...	...	...

Figure 32-21: YUV 4:2:2 Expected Buffer Structure and Data Stream Format

The number of pixels per line in YUV mode must be an even number (for both input buffers), and the first chroma component in each line must be a V component.

Operation in RGB888 Format

Each R/G/B component is stored in 8 bits of data. The figures below illustrate correct PIXC input buffer structure and data stream format.

RGB888 Data Expected Data Buffer Structure

ADDR_BASE + 0	R1 1	G1 1	B1 1	R1 2		G1 640	B1 640
ADDR_BASE + 1280	R2 1	R2 1	B2 1	B2 2		G2 640	B2 640
	⋮	⋮	⋮	⋮		⋮	⋮
ADDR_BASE + (479 x 1280)	R480 1					U480 639	Y480 640

RGB888 Expected Data Stream Format to PIXC

B31		B16 B15	B0
R12	B11	G11	R11
G13	R13	B12	G12
...	...	...	...
...	...	...	...

Figure 32-22: RGB888 Expected Buffer Structure and Data Stream Format

For operation in RGB format, the total number of pixels in both input buffers must be a multiple of 4, so that the image boundary aligns with a 32-bit DMA word boundary.

Operation in RGB565 Format

Two pixels are packed into one 32 bit word. Each pixel occupies 16 bits. The figures below illustrates how the pixel data is arranged in memory.

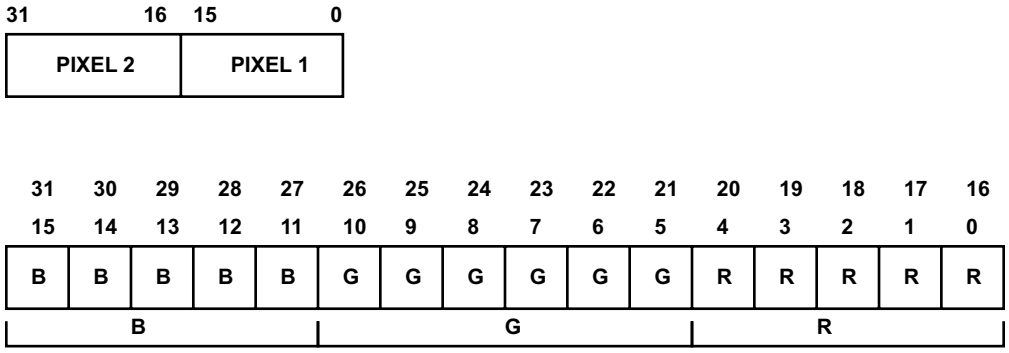


Figure 32-23: Memory Arrangement – RGB565 Format

Operation in RGB666 Format

Four pixels are packed into three 32 bit words. Each pixel occupies 24 bits. Of these 24 bits, the lowest 18 bits are the pixel data and the highest 6 bits are discarded for inputs and zeroed for outputs. The figure

below illustrates how the pixel data is arranged in memory. Note that 1 = pixel 1, 2 = pixel 2, 3 = pixel 3, and 4 = pixel 4.

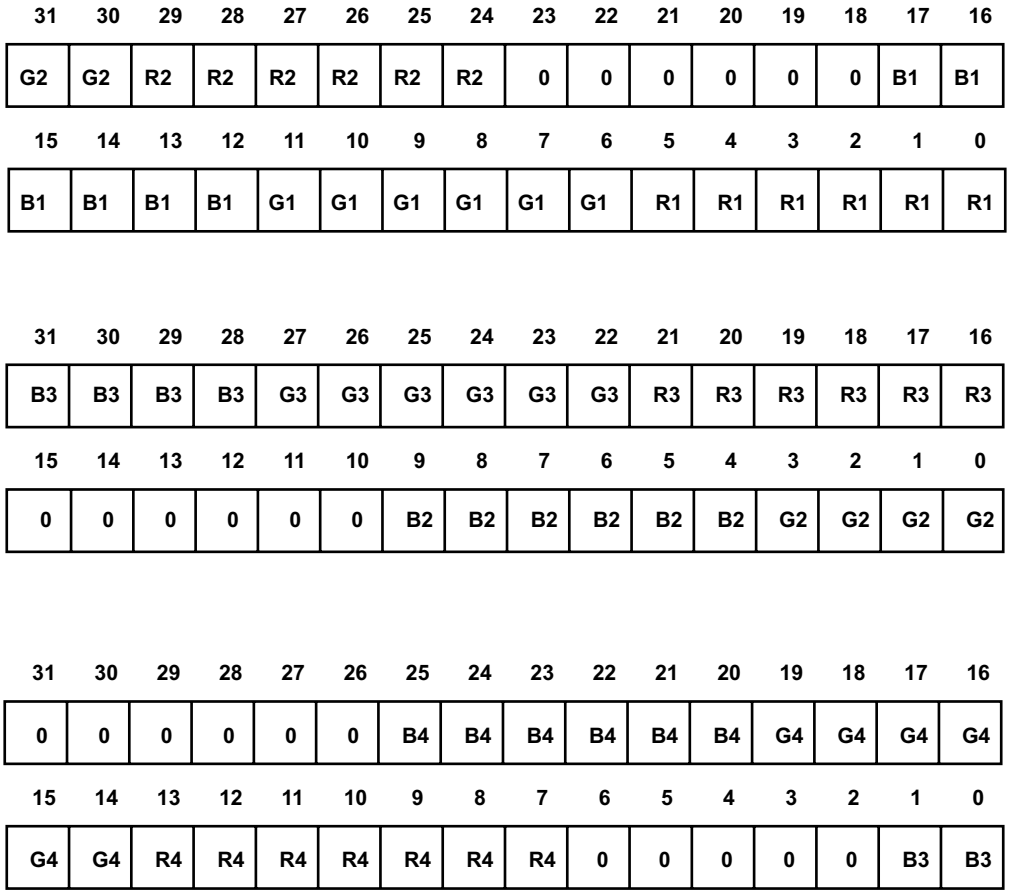


Figure 32-24: Memory Arrangement – RGB666 Format

Operation with RGB656 and RGB666 Formats

For RGB656 and RGB666 formats, the PIXC internal engine reads the relevant bits from the input and zero pads to 8 bits. The internal PIXC data path is always 8 bits. If the output requires the pixels in either RGB565 or RGB666 after the data path operations (mixer, color space conversion), the individual color components are truncated (the lower bits dropped) and converted to the required formats.

PIXC Event Control

Event control for the PIXC consists of working with interrupts, triggers, and DMA channels. For more information, see the *PIXC Functional Description*.

PIXC Programming Model

The PIXC is able to take data in from either memory or the video subsystem. The output is able to go back to memory or piped back to the video subsystem. Therefore, the video subsystem needs to be configured correctly before enabling and using of the PIXC.

Mode Configuration

This section describes typical mode configuration tasks for using the PIXC.

Performing Data Overlay

Data overlay allows a separate image to be over laid or blended with the main input image.

1. Define rectangular region in the input image to be covered by an overlay.
2. Define single DMA descriptor for the overlay transfer.
3. Configure overlay coordinate registers in PIXC (`PIXC_HSTART_A`, `PIXC_HEND_A`, `PIXC_VSTART_A`, and `PIXC_VEND_A`).
4. Configure transparency ratio register (`PIXC_TRANSP_A`).
5. Optionally define a second overlay region and configure `PIXC_HSTART_B`, `PIXC_HEND_B`, `PIXC_VSTART_B`, and `PIXC_VEND_B` and `PIXC_TRANS_B` accordingly.
6. Enable PIXC and overlay functionality by setting `PIXC_CTL.EN` = 1, `PIXC_CTL.OVAEN` = 1 and `PIXC_CTL.OVBEN` = 1(if a 2nd overlay region is defined).

RESULT:

The output image is blended with the overlay images at the specified overlay region.

Performing Color Space Conversion Only

The PIXC can be used to color convert an image or frame from RGB to YUV or from YUV to RGB

1. Configure and set up input connection of the PIXC in the video sub-system to obtain the main input image.
2. Configure and set up output connection of the PIXC in the video sub-system to output the resulting output image.

3. Program the conversion component registers (PIXC_CONRY, PIXC_CONGU and PIXC_CONBV) and conversion bias register (PIXC_CCBIAS).
4. Enable the PIXC by setting PIXC_CTL.EN=1. Ensure overlay is disabled (PIXC_CTL.OVENB=0 and PIXC_CTL.OVENB=0).

RESULT:

Converts an input image from RGB to YUV or an image from YUV to RGB

ADSP-BF60x PIXC Register Descriptions

Pixel Compositor (PIXC) contains the following registers.

Table 32-8: ADSP-BF60x PIXC Register List

Name	Description
PIXC_CTL	Control Register
PIXC_PPL	Pixels Per Line Register
PIXC_LPF	Line Per Frame Register
PIXC_HSTART_A	Overlay A Horizontal Start Register
PIXC_HEND_A	Overlay A Horizontal End Register
PIXC_VSTART_A	Overlay A Vertical Start Register
PIXC_VEND_A	Overlay A Vertical End Register
PIXC_TRANSP_A	Overlay A Transparency Ratio Register
PIXC_HSTART_B	Overlay B Horizontal Start Register
PIXC_HEND_B	Overlay B Horizontal End Register
PIXC_VSTART_B	Overlay B Vertical Start Register
PIXC_VEND_B	Overlay B Vertical End Register
PIXC_TRANSP_B	Overlay B Transparency Ratio Register
PIXC_IRQSTAT	Interrupt Status Register
PIXC_CONRY	RY Conversion Component Register

Table 32-8: ADSP-BF60x PIXC Register List (Continued)

Name	Description
PIXC_CONGU	GU Conversion Component Register
PIXC_CONBV	BV Conversion Component Register
PIXC_CCBIAS	Conversion Bias Register
PIXC_TC	Transparency Color Register

Control Register

The PIXC_CTL register provides module enable, overlay enable, re-sampling mode selection, input/overlay/output data format selection, transparent color enable, and input/output RGB data format.

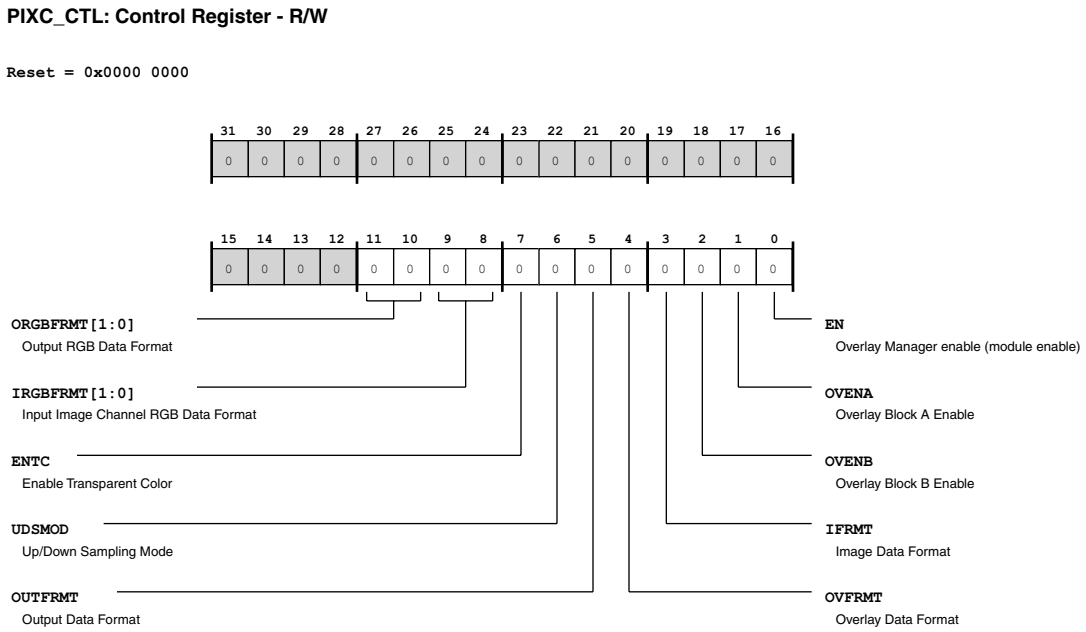


Figure 32-25: PIXC_CTL Register Diagram

Table 32-9: PIXC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	ORGBFRMT	Output RGB Data Format. The PIXC_CTL.ORGBFRMT bits select the output RGB data format. This selection only is relevant when PIXC_CTL.OUTFRMT is set.	
		0	RGB 888
		1	Reserved
		2	RGB 565
		3	RGB 666
9:8 (R/W)	IRGBFRMT	Input Image Channel RGB Data Format. The PIXC_CTL.IRGBFRMT bits select the input image channel RGB data format. This selection only is relevant when PIXC_CTL.IFRMT is set.	
		0	RGB 888
		1	Reserved
		2	RGB 565
		3	RGB 666
7 (R/W)	ENTC	Enable Transparent Color.	
		0	Disable Transparent Color
		1	Enable Transparent Color
6 (R/W)	UDSMOD	Up/Down Sampling Mode. The PIXC_CTL.UDSMOD selects whether the PIXC either uses averaging for up/down sampling or uses duplicating for up sampling and dropping for down sampling.	
		0	Duplicating for up sampling and dropping for down sampling
		1	Averaging for both up sampling and down sampling
5 (R/W)	OUTFRMT	Output Data Format. The PIXC_CTL.OUTFRMT selects YUV or RGB for the output format.	
		0	YUV Data Format
		1	RGB Data Format

Table 32-9: PIXC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	OVFRMT	Overlay Data Format. The PIXC_CTL.OVFRMT selects YUV or RGB for the overlay format.	
		0	YUV Data Format
		1	RGB Data Format
3 (R/W)	IFRMT	Image Data Format. The PIXC_CTL.IFRMT selects YUV or RGB for the input format.	
		0	YUV Data Format
		1	RGB Data Format
2 (R/W)	OVENB	Overlay Block B Enable. If both PIXC_CTL.OVENA and PIXC_CTL.OVENB are cleared, only color space conversion is enabled.	
		0	Disable
		1	Enable
1 (R/W)	OVENA	Overlay Block A Enable. If both PIXC_CTL.OVENA and PIXC_CTL.OVENB are cleared, only color space conversion is enabled.	
		0	Disable
		1	Enable
0 (R/W)	EN	Overlay Manager enable (module enable). The PIXC_CTL.EN bit enables the PIXC.	
		0	Disable
		1	Enable

Pixels Per Line Register

The PIXC_PPL register provides the number of pixels per line of the display.

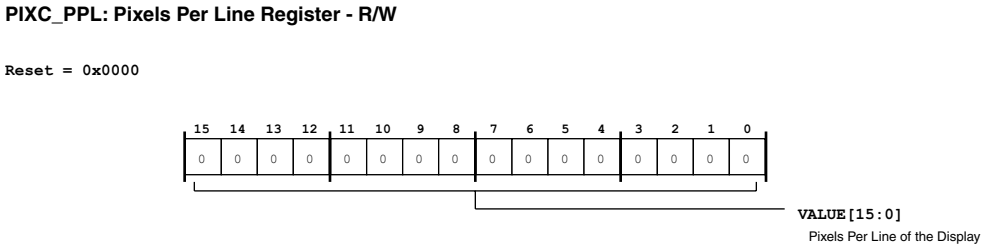


Figure 32-26: PIXC_PPL Register Diagram

Table 32-10: PIXC_PPL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Pixels Per Line of the Display.

Line Per Frame Register

The PIXC_LPF register provides the number of lines per frame of the display.

PIXC_LPF: Line Per Frame Register - R/W

Reset = 0x0000

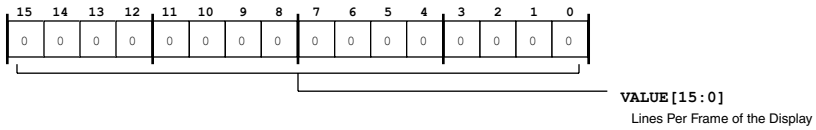


Figure 32-27: PIXC_LPF Register Diagram

Table 32-11: PIXC_LPF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Lines Per Frame of the Display.

Overlay A Horizontal Start Register

The PIXC_HSTART_A register provides the horizontal start pixel coordinates of overlay A data.

PIXC_HSTART_A: Overlay A Horizontal Start Register - R/W

Reset = 0x0000

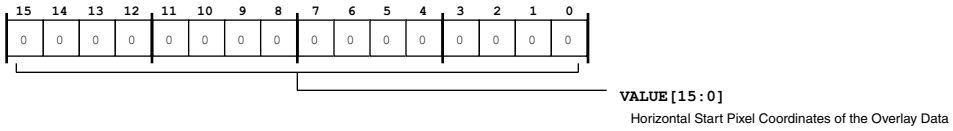


Figure 32-28: PIXC_HSTART_A Register Diagram

Table 32-12: PIXC_HSTART_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Horizontal Start Pixel Coordinates of the Overlay Data.

Overlay A Horizontal End Register

The PIXC_HEND_A registers provides the horizontal end pixel coordinates of overlay A data.

PIXC_HEND_A: Overlay A Horizontal End Register - R/W

Reset = 0x0000

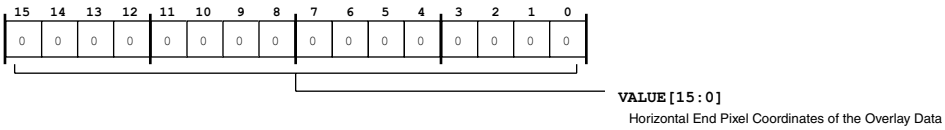


Figure 32-29: PIXC_HEND_A Register Diagram

Table 32-13: PIXC_HEND_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Horizontal End Pixel Coordinates of the Overlay Data.

Overlay A Vertical Start Register

The PIXC_VSTART_A registers provides the vertical start pixel coordinates of overlay A data.

PIXC_VSTART_A: Overlay A Vertical Start Register - R/W

Reset = 0x0000

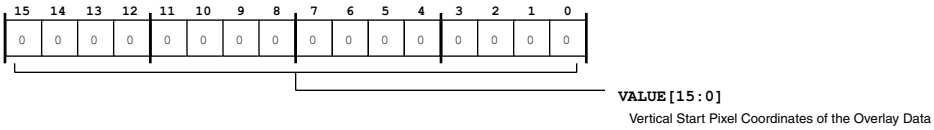


Figure 32-30: PIXC_VSTART_A Register Diagram

Table 32-14: PIXC_VSTART_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Vertical Start Pixel Coordinates of the Overlay Data.

Overlay A Vertical End Register

The PIXC_VEND_A register provides the vertical end pixel coordinates of overlay A data.

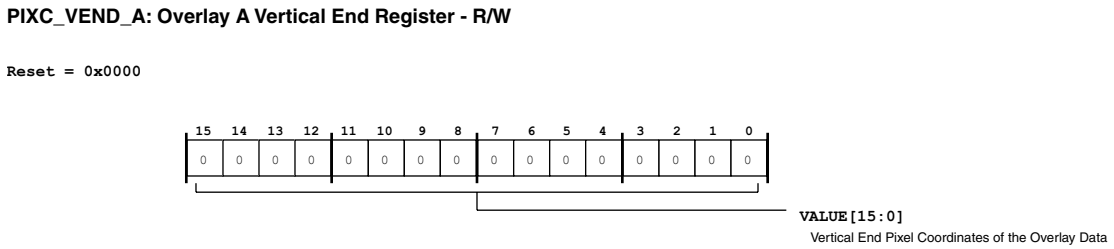


Figure 32-31: PIXC_VEND_A Register Diagram

Table 32-15: PIXC_VEND_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Vertical End Pixel Coordinates of the Overlay Data.

Overlay A Transparency Ratio Register

The PIXC_TRANSP_A register provides overlay A transparency ratio values.

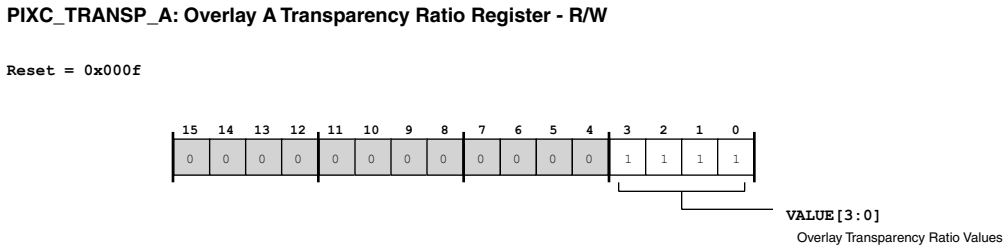


Figure 32-32: PIXC_TRANSP_A Register Diagram

Table 32-16: PIXC_TRANSP_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Overlay Transparency Ratio Values.

Overlay B Horizontal Start Register

The PIXC_HSTART_B register provides the horizontal start pixel coordinates of overlay B data.

PIXC_HSTART_B: Overlay B Horizontal Start Register - R/W

Reset = 0x0000

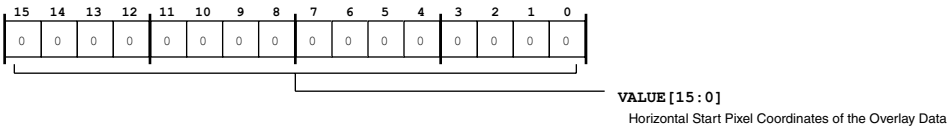


Figure 32-33: PIXC_HSTART_B Register Diagram

Table 32-17: PIXC_HSTART_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Horizontal Start Pixel Coordinates of the Overlay Data.

Overlay B Horizontal End Register

The PIXC_HEND_B registers provides the horizontal end pixel coordinates of overlay B data.

PIXC_HEND_B: Overlay B Horizontal End Register - R/W

Reset = 0x0000

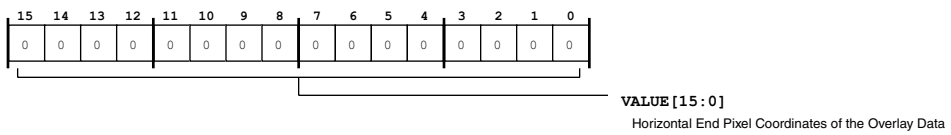


Figure 32-34: PIXC_HEND_B Register Diagram

Table 32-18: PIXC_HEND_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Horizontal End Pixel Coordinates of the Overlay Data.

Overlay B Vertical Start Register

The PIXC_VSTART_B registers provides the vertical start pixel coordinates of overlay B data.

PIXC_VSTART_B: Overlay B Vertical Start Register - R/W

Reset = 0x0000

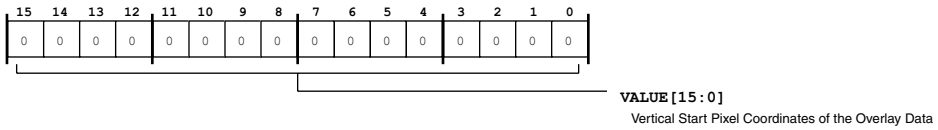


Figure 32-35: PIXC_VSTART_B Register Diagram

Table 32-19: PIXC_VSTART_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Vertical Start Pixel Coordinates of the Overlay Data.

Overlay B Vertical End Register

The PIXC_VEND_B register provides the vertical end pixel coordinates of overlay B data.

PIXC_VEND_B: Overlay B Vertical End Register - R/W

Reset = 0x0000

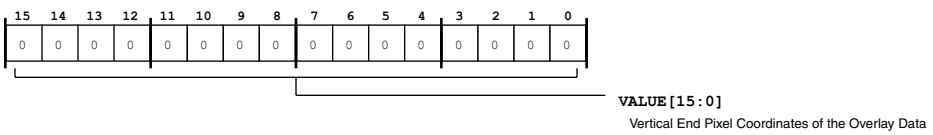


Figure 32-36: PIXC_VEND_B Register Diagram

Table 32-20: PIXC_VEND_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Vertical End Pixel Coordinates of the Overlay Data.

Overlay B Transparency Ratio Register

The PIXC_TRANSP_B register provides overlay B transparency ratio values.

PIXC_TRANSP_B: Overlay B Transparency Ratio Register - R/W

Reset = 0x000f

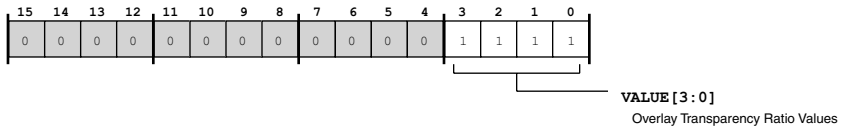


Figure 32-37: PIXC_TRANSP_B Register Diagram

Table 32-21: PIXC_TRANSP_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Overlay Transparency Ratio Values.

Interrupt Status Register

The PIXC_IRQSTAT register provides overlay interrupt configuration and status information.

PIXC_IRQSTAT: Interrupt Status Register - R/W

Reset = 0x0000

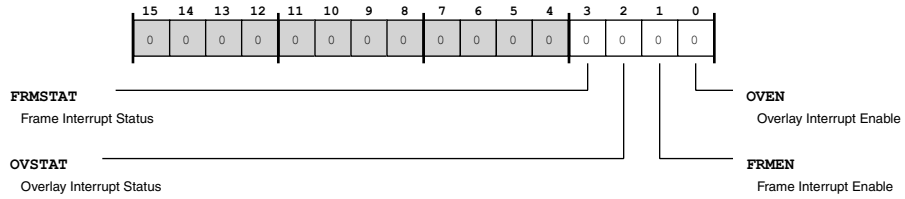


Figure 32-38: PIXC_IRQSTAT Register Diagram

Table 32-22: PIXC_IRQSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W1C)	FRMSTAT	Frame Interrupt Status. The PIXC_IRQSTAT.FRMSTAT indicates an end of frame condition for the PIXC. This status bit is write-1-to-clear.	
2 (R/W1C)	OVSTAT	Overlay Interrupt Status. The PIXC_IRQSTAT.OVSTAT indicates an end of last valid overlay condition for the PIXC. This status bit is write-1-to-clear.	
1 (R/W)	FRMEN	Frame Interrupt Enable. The PIXC_IRQSTAT.FRMEN enables the PIXC end of frame interrupt.	
		0	Disable
		1	Enable
0 (R/W)	OVEN	Overlay Interrupt Enable. The PIXC_IRQSTAT.OVEN enables the PIXC end of last valid overlay interrupt.	
		0	Disable
		1	Enable

RY Conversion Component Register

The PIXC_CONRY register provides the R/Y conversion coefficients in the color space conversion matrix.

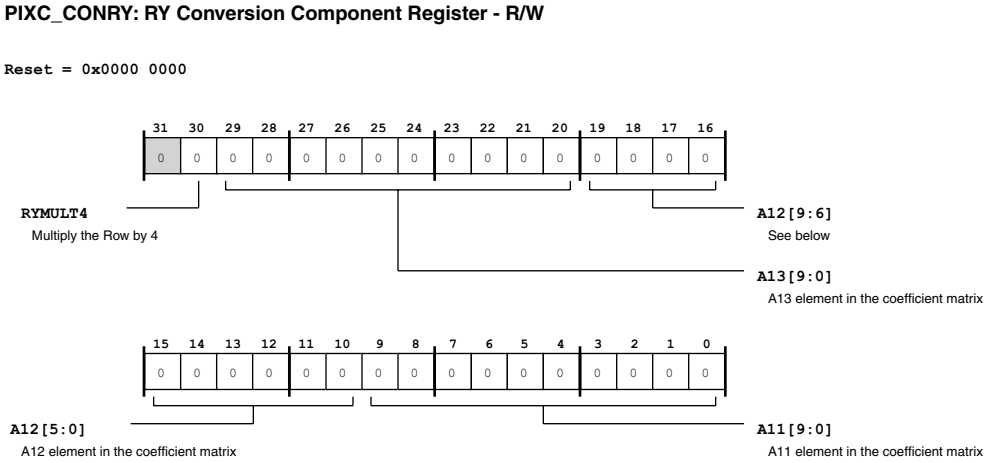


Figure 32-39: PIXC_CONRY Register Diagram

Table 32-23: PIXC_CONRY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	RYMULT4	Multiply the Row by 4.	
		0	Disable
		1	Enable
29:20 (R/W)	A13	A13 element in the coefficient matrix.	
19:10 (R/W)	A12	A12 element in the coefficient matrix.	
9:0 (R/W)	A11	A11 element in the coefficient matrix.	

GU Conversion Component Register

The PIXC_CONGU register provides the G/U conversion coefficients in the color space conversion matrix.

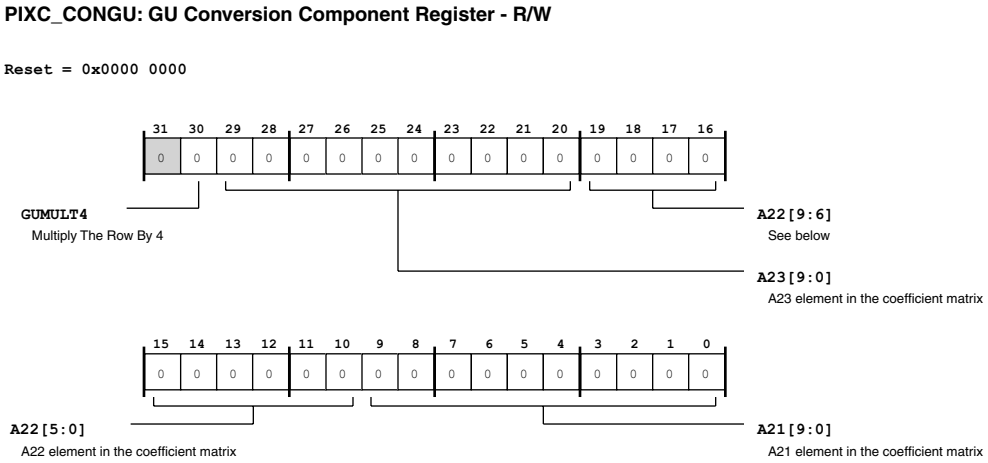


Figure 32-40: PIXC_CONGU Register Diagram

Table 32-24: PIXC_CONGU Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	GUMULT4	Multiply The Row By 4.	
		0	Disable
		1	Enable

Table 32-24: PIXC_CONGU Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29:20 (R/W)	A23	A23 element in the coefficient matrix.
19:10 (R/W)	A22	A22 element in the coefficient matrix.
9:0 (R/W)	A21	A21 element in the coefficient matrix. The PIXC_CONGU register provides the B/V conversion coefficients in the color space conversion matrix.

BV Conversion Component Register

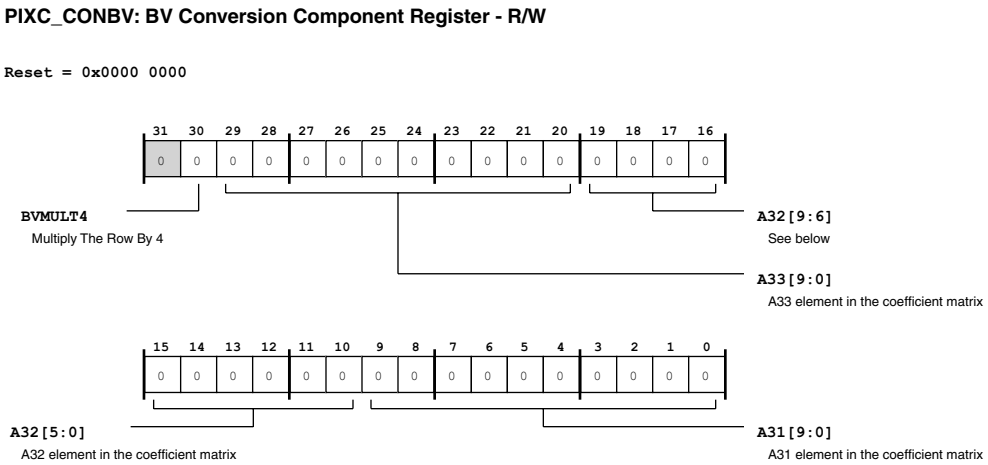


Figure 32-41: PIXC_CONBV Register Diagram

Table 32-25: PIXC_CONBV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	BVMULT4	Multiply The Row By 4.	
		0	Disable
		1	Enable
29:20 (R/W)	A33	A33 element in the coefficient matrix.	
19:10 (R/W)	A32	A32 element in the coefficient matrix.	

Table 32-25: PIXC_CONBV Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	A31	A31 element in the coefficient matrix.

Conversion Bias Register

The PIXC_CCBIAS register provides the bias values in the color space conversion matrix.

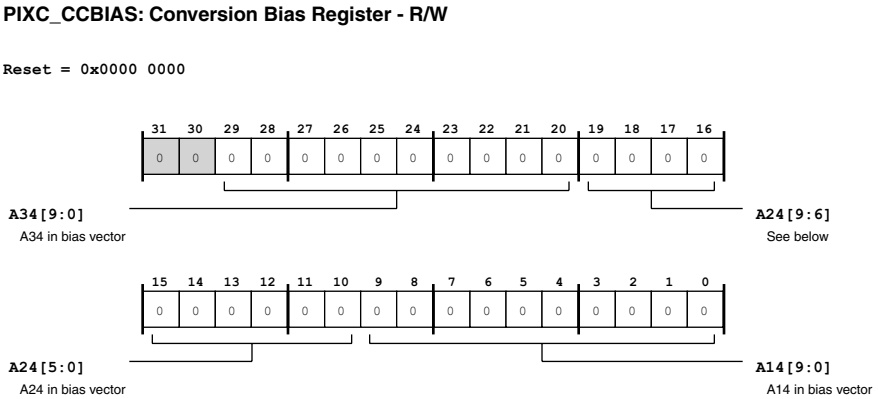


Figure 32-42: PIXC_CCBIAS Register Diagram

Table 32-26: PIXC_CCBIAS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:20 (R/W)	A34	A34 in bias vector.
19:10 (R/W)	A24	A24 in bias vector.
9:0 (R/W)	A14	A14 in bias vector.

Transparency Color Register

The PIXC_TC register provides the transparent color value.

PIXC_TC: Transparency Color Register - R/W

Reset = 0x0000 0000

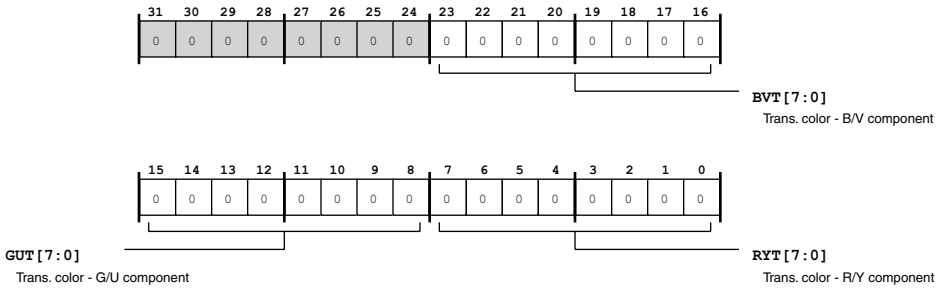


Figure 32-43: PIXC_TC Register Diagram

Table 32-27: PIXC_TC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	BVT	Trans. color B/V component.
15:8 (R/W)	GUT	Trans. color G/U component.
7:0 (R/W)	RYT	Trans. color R/Y component.

33 Reset Control Unit (RCU)

Reset is the initial state of the whole processor (or one of the cores) and is the result of a hardware or software triggered event. In this state, all control registers are set to their default values and functional units are idle. Exiting a full system reset starts with Core-0 only being ready to boot. Exiting a Core n only reset starts with this Core n being ready to boot.

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. This is particularly important when only one of the cores is reset (programs must ensure that there is no pending system activity involving the core that is being reset).

RCU Features

RCU module supports the following features:

- Hardware reset through the $\overline{\texttt{SYS\_HWRST}}$ pin
- Software system reset through RCU registers
- Hardware system reset through:
  - TRU module
  - SEC module
  - SDU module
- Core reset through RCU registers

RCU Functional Description

The RCU provides reset operation control and status features. Use the following sections to provide functional descriptions of the RCU:

- *ADSP-BF60x RCU Register List*
- *ADSP-BF60x RCU Trigger List*
- *RCU Definitions*
- *RCU Architectural Concepts*

ADSP-BF60x 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 (units in different clock domains have to enter reset asynchronously, but units exit reset in a deterministic way) define how reset signals are generated. Reset signals propagate through all the functional units asynchronously. For more information on RCU functionality, see the RCU register descriptions.

Table 33-1: ADSP-BF60x RCU Register List

Name	Description
RCU_CTL	Control Register
RCU_STAT	Status Register
RCU_CRCTL	Core Reset Control Register
RCU_CRSTAT	Core Reset Status Register
RCU_SIDIS	System Interface Disable Register
RCU_SISTAT	System Interface Status Register
RCU_SVECT_LCK	SVECT Lock Register
RCU_BCODE	Boot Code Register
RCU_SVECT0	Software Vector Register 0
RCU_SVECT1	Software Vector Register 1

ADSP-BF60x RCU Trigger List

Table 33-2: ADSP-BF60x RCU Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
None		

Table 33-3: ADSP-BF60x RCU Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
RCU0 System Reset 0	0	

Table 33-3: ADSP-BF60x RCU Trigger List Trigger Slaves (Continued)

Description	Trigger ID	Sensitivity
RCU0 System Reset 1	1	

RCU Definitions

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

The following are types of resets that are defined by their target or source.

Hardware Reset (by target)

All functional units are set to their default states, no exception. History is lost.

System Reset (by target)

All functional units except the RCU are set to their default states.

Core n Only Reset (by target)

Affects Core n only. The system software should guarantee that the core in reset state is not accessed by any bus master.

Hardware Reset (by source)

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

System Reset (by source)

May be triggered by software (writing to the `RCU_CONTROL`) register or by anotherfunctional unit such as the TRU or any of the generic reset inputs.

For processors supporting hibernated mode, writing to the DPM to exit hibernate provides another reset source.

Core n Only Reset (by source)

Triggered by software.

RCU Architectural Concepts

To understand the architecture of the RCU, one must 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 have to enter reset asynchronously but exit reset in a deterministic way.

It is the program's responsibility to guarantee that none of the reset functions put the system in an undefined state or cause resources to stall. This is particularly important when only one of the cores is reset because the program needs to guarantee that there is no pending system activity involving Core n before it is reset. For example, there should be no pending transactions to core n when the core is reset.

The following table defines how reset sources affect the different functional units.

Reset Source	Reset Type	Affected Functional Units
$\overline{\text{SYS\_HWRST}}$ pin assertion	Hardware Reset	All functional units, except RTC (if present)
Hibernate wakeup event (wakeup triggered reset)	System Reset	All functional units, except: • RTC (if present), • RCU_STAT, and • the units on the VDDEXT power domain
SYSCLK clock domain reset	System Reset	All functional units, except: • RTC (if present), • RCU_CRCTL.CRn, • RCU_STAT, • RCU_BCODE, and • the units on the VDDEXT power domain
SCLK (or SCLK0 for multi-core processors) clock domain reset	System Reset	All functional units, except: • RTC (if present), • RCU_CRCTL.CRn, • RCU_STAT, • RCU_BCODE, and • the units on the VDDEXT power domain
SCLK1 clock domain reset	System Reset	All functional units, except: • RTC (if present), • RCU_CRCTL.CRn, • RCU_STAT, • RCU_BCODE, and • the units on the VDDEXT power domain

Reset Source	Reset Type	Affected Functional Units
RCU_CTL.SYSRST bit set (software triggered reset)	System Reset	All functional units, except: • RTC (if present), • RCU_CRCTL.CRn, • RCU_STAT, • RCU_BCODE, and • the units on the VDDEXT power domain
RCU_CRCTL.CRn bit set (software triggered reset)	Core Only Reset	Core n only, for (15 ≥ n ≥ 0)

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 if 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.

Resetting a Core

The RCU allows reset of a given core n using another core.

Core n can be individually reset by software, either setting any of CRn (15 ≥ n ≥ 0) bits in RCU_CRCTL register. Cores that reset themselves will not be able to guarantee that all the system transactions to or from it have completed. Although a core n reset can be triggered by core n itself it is recommended that it is triggered by another core. Core n may be reset to restore its functionality when it cannot execute software. The suggested sequence to reset core n only is shown below.

1. Clear the RCU_CRSTAT.CRn bit.
2. Disable interrupts to core n
3. Set the RCU_SIDIS.SIn bit to disable core n's interfaces in order to stop DMA accesses to its L1, stop core n's accesses to memory, and stop accesses to MMRs.
4. Test the RCU_SISTAT.SIn bit to detect when accesses to core n have been disabled and all the pending transactions have completed.
5. Set the RCU_CRCTL.Rn bit to reset core n.
6. Poll the RCU_CRSTAT.CRn bit until core n is in reset.
7. Once the core is in reset, clear the RCU_SIDIS.SIn bit to re-enable the core interfaces.

8. Clear the RCU_CRCTL.Rn bit to take core n out of reset.
9. Poll the RCU_CRSTAT.CRn bit until core n is out of reset.

ADSP-BF60x Specific Information

The following information specific to ADSP-BF60x processors should be kept in mind when reading this chapter:

- There are two cores, core 0 and core 1
- There is no RTC (real-time clock)
- The only SCLK0 domain source of system reset is the SDU
- There is no SCLK1 source of system reset
- The SYSCLK sources of system reset are TRU and SEC

ADSP-BF60x RCU Register Descriptions

Reset Control Unit (RCU) contains the following registers.

Table 33-4: ADSP-BF60x RCU Register List

Name	Description
RCU_CTL	Control Register
RCU_STAT	Status Register
RCU_CRCTL	Core Reset Control Register
RCU_CRSTAT	Core Reset Status Register
RCU_SIDIS	System Interface Disable Register
RCU_SISTAT	System Interface Status Register
RCU_SVECT_LCK	SVECT Lock Register
RCU_BCODE	Boot Code Register
RCU_SVECT0	Software Vector Register 0
RCU_SVECT1	Software Vector Register 1

Control Register

The RCU control register (RCU_CTL) provides a register lock, controls for the system reset pin, and a control for system reset.

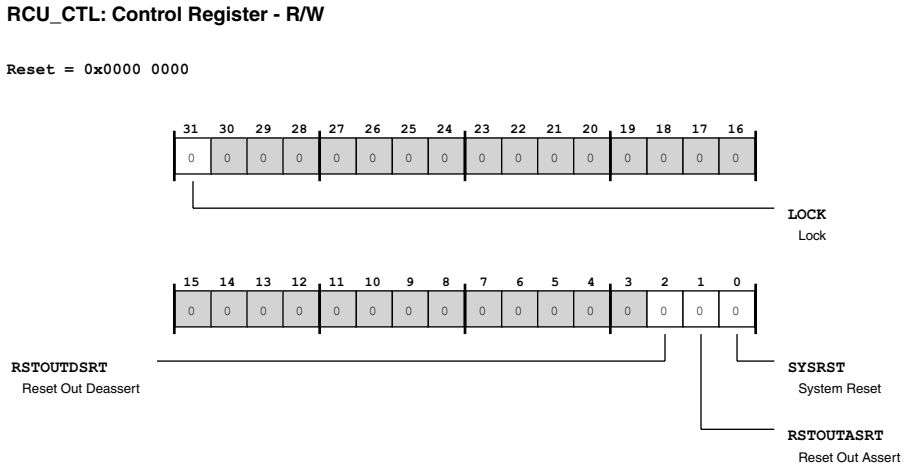


Figure 33-1: RCU_CTL Register Diagram

Table 33-5: RCU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the RCU_CTL.LOCK bit is set, the RCU_CTL register is read only (locked).	
		0	Unlock
		1	Lock
2 (R0/W1A)	RSTOUTDSRT	Reset Out Deassert. The RCU_CTL.RSTOUTDSRT bit controls deassertion of the system reset pin.	
		0	No Action
		1	Deassert RSTOUT
1 (R0/W1A)	RSTOUTASRT	Reset Out Assert. The RCU_CTL.RSTOUTASRT bit controls assertion of the system reset pin.	
		0	No Action
		1	Assert RSTOUT

Table 33-5: RCU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R0/W1A)	SYSRST	System Reset. The RCU_CTL.SYSRST bit provides reset for all system units.	
		0	No Action
		1	System 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 cleared by software. Error status bits are cleared by any reset event.

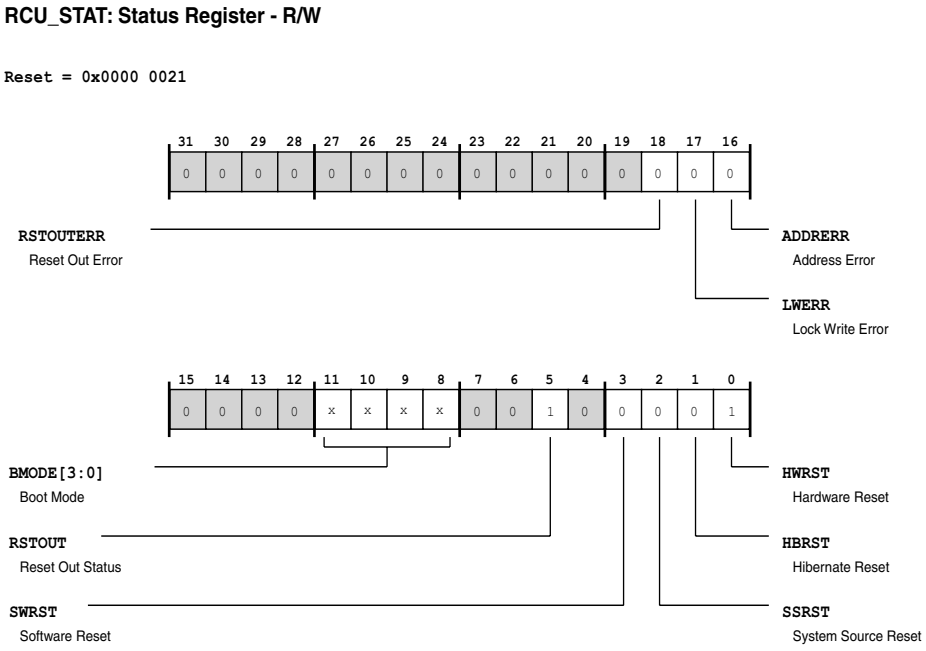


Figure 33-2: RCU_STAT Register Diagram

Table 33-6: RCU_STAT Register Fields

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

Table 33-6: RCU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
1 (R/W1C)	HBRST	Hibernate Reset. The RCU_STAT.HBRST bit indicates that a hibernate reset has occurred since the last time a hardware reset occurred or since the RCU_STAT.HWRST 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

Core Reset 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.CRn) for each processor core on the product.

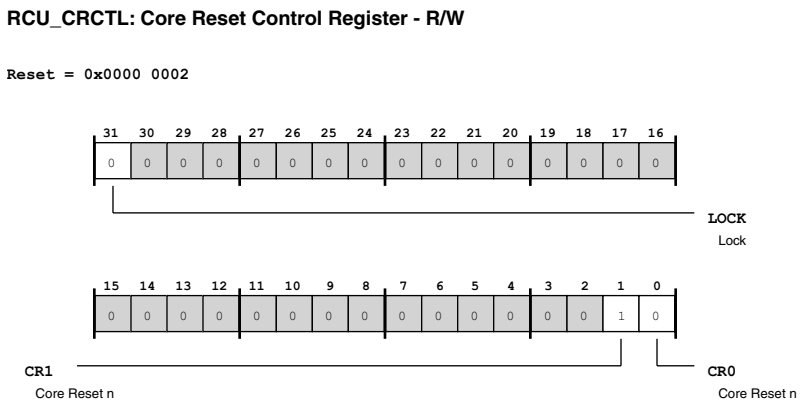


Figure 33-3: RCU_CRCTL Register Diagram

Table 33-7: RCU_CRCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the RCU_CRCTL.LOCK bit is set, the RCU_CRCTL register is read only (locked).	
		0	Unlock
		1	Lock
1:0 (R/W)	CRn	Core Reset n. The RCU_CRCTL.CRn bit resets processor core n.	
		0	Inactive
		1	Reset Processor Core n

Core Reset Status Register

The RCU core reset status register (RCU_CRSTAT) contains status bits, indicating which core or cores have been reset.

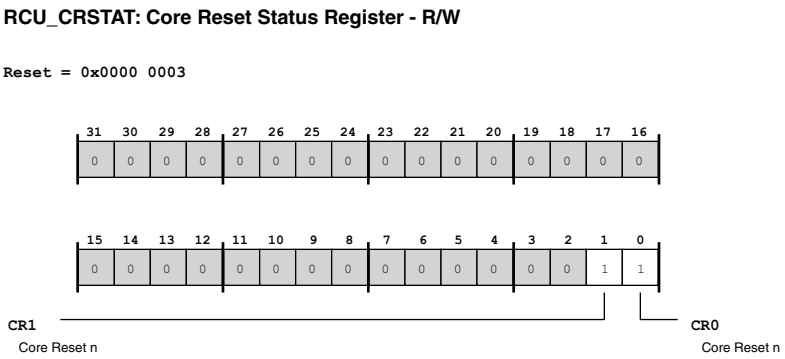


Figure 33-4: RCU_CRSTAT Register Diagram

Table 33-8: RCU_CRSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W1C)	CRn	Core Reset n. The RCU_CRSTAT.CRn bits indicate which cores have been reset since the last time the bit was cleared.	
		0	Inactive
		1	Reset of Core n Occurred

System Interface Disable Register

The RCU system interface disable register (RCU_SIDIS lets the RCU selectively disable the functional units in the processor. For information on mapping between RCU_SIDIS bits and functional units, see the RCU functional description.

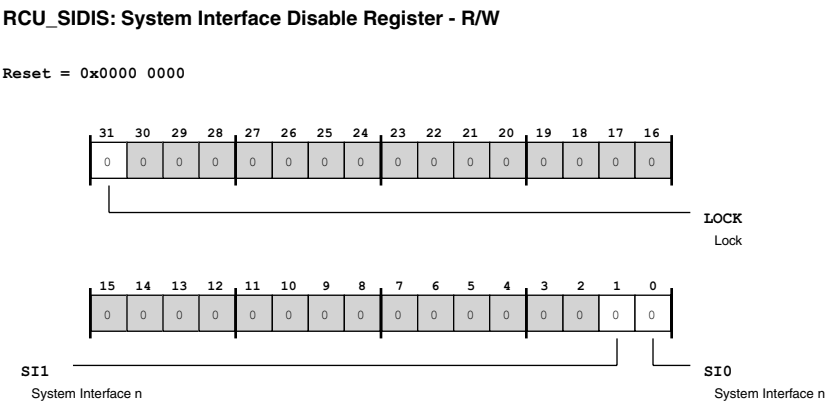


Figure 33-5: RCU_SIDIS Register Diagram

Table 33-9: RCU_SIDIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the RCU_SIDIS.LOCK bit is set, the RCU_SIDIS register is read only (locked).	
		0	Unlock
		1	Lock
1:0 (R/W)	SIn	System Interface n. The RCU_SIDIS.SIn bits select functional units in the processor to be disabled.	
		0	Enable
		1	Disable Functional Unit n

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.

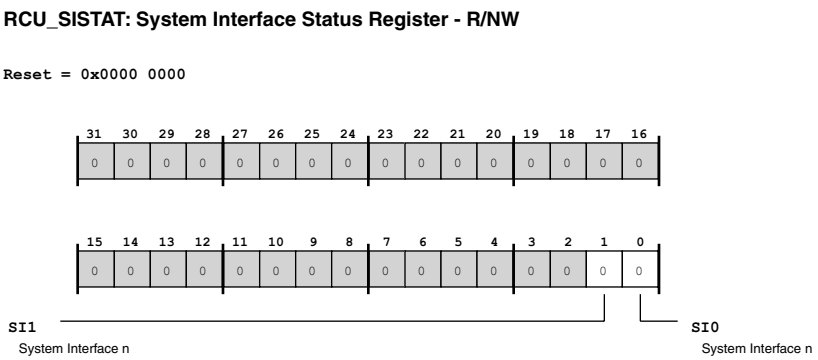


Figure 33-6: RCU_SISTAT Register Diagram

Table 33-10: RCU_SISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/NW)	SIn	System Interface n.	
		0	No Acknowledge
		1	Disable Request Acknowledged

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.

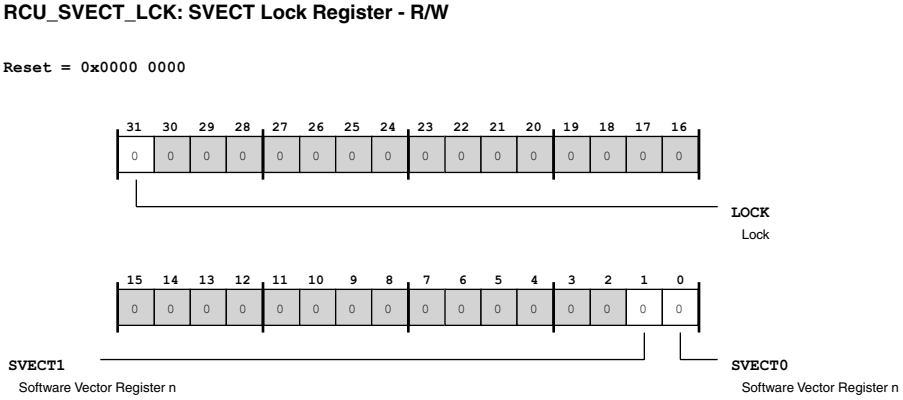


Figure 33-7: RCU_SVECT_LCK Register Diagram

Table 33-11: RCU_SVECT_LCK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the RCU_SVECT_LCK.LOCK bit is set, the RCU_SVECT_LCK register is read only (locked).	
		0	Unlock
		1	Lock
1:0 (R/W)	SVECTn	Software Vector Register n. The RCU_SVECT_LCK.SVECTn bits enable a software vector (reset vector) for each core n.	
		0	Disable
		1	Enable Software Vector for Core n

Boot Code Register

The RCU software vector lock register (RCU_BCODE) provides a register lock and software vector n enable bits for each processor core on the product. For a processor-specific definition of the RCU_BCODE register, see the Booting Register Reference in the Boot ROM chapter.

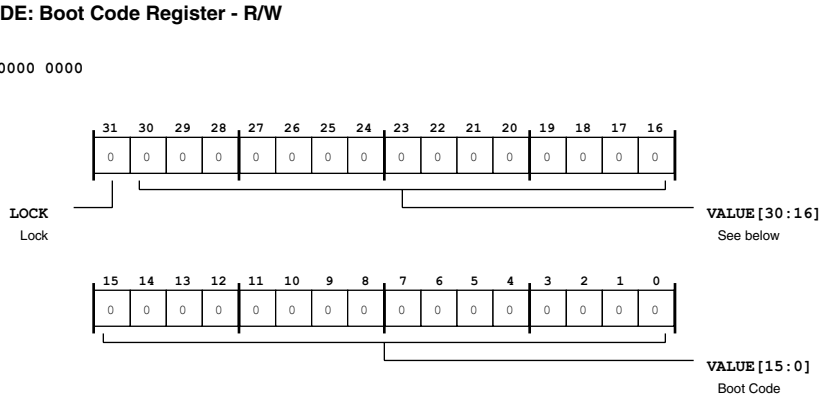


Figure 33-8: RCU_BCODE Register Diagram

Table 33-12: RCU_BCODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the RCU_BCODE.LOCK bit is set, the RCU_BCODE register is read only (locked).	
		0	Unlock
		1	Lock
30:0 (R/W)	VALUE	Boot Code. The RCU_BCODE.VALUE bits contain a boot code for the processor. For more information, see the RCU functional description.	

Software Vector Register 0

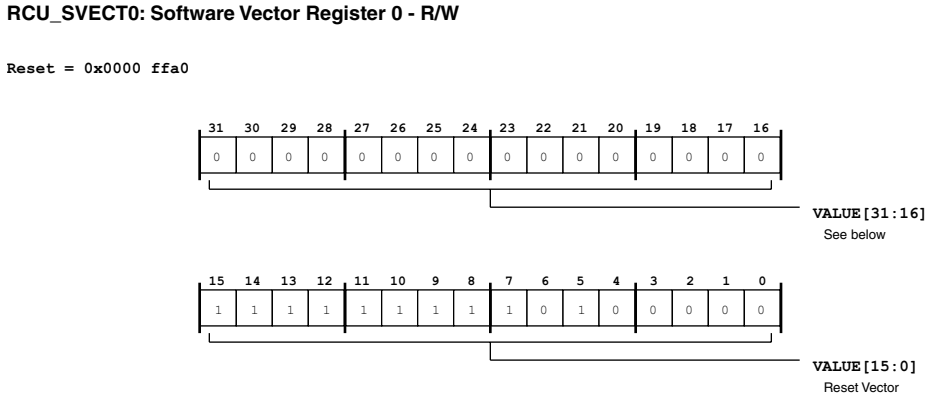


Figure 33-9: RCU_SVECT0 Register Diagram

Table 33-13: RCU_SVECT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Reset Vector.

Software Vector Register 1

RCU_SVECT1: Software Vector Register 1 - R/W

Reset = 0x0000 ffa0

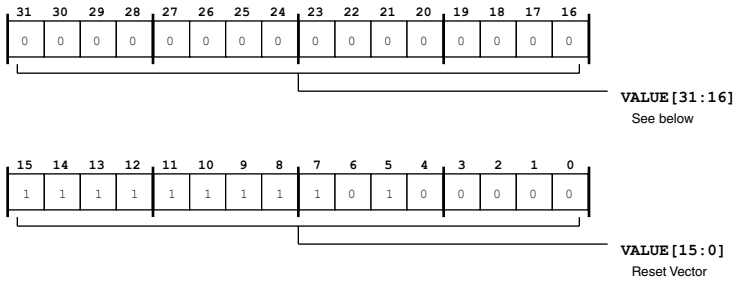


Figure 33-10: RCU_SVECT1 Register Diagram

Table 33-14: RCU_SVECT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Reset Vector.

34 Boot ROM and Booting the Processor

This section provides an overview of all the functions of the boot ROM.

NOTE: `Notice of Changes:` This documentation contains material that is subject to change without notice. The content of the boot ROM as well as hardware behavior may change across silicon revisions. See the anomaly list for differences between silicon revisions.

When the `RESET` input signal releases, the processor starts fetching and executing from the on-chip boot ROM.

The internal boot ROM includes software for initial processor configuration, memory initialization for parity protected and ECC protected memories in addition to a boot kernel that is responsible for processing the boot loader stream received from the boot source and loading the user application code and data.

NOTE: All available peripherals are not supported by the boot kernel, for more information, see specific boot modes.

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 to be loaded to the chip. This section describes in detail different aspects of the stream, its blocks, and some common use cases.

Each block begins with a block header which 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 and 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 some action, most commonly to simply load data to a specified location. Other actions include running code that initializes a peripheral, forwarding data to a peripheral, or processing data then loading it to a location.

A loader stream must always begin with a First block, and end with a Final block. The loader file contains the boot stream and is made available to hardware by programming or burning it into non-volatile external memory, or sending it through a peripheral during boot time.

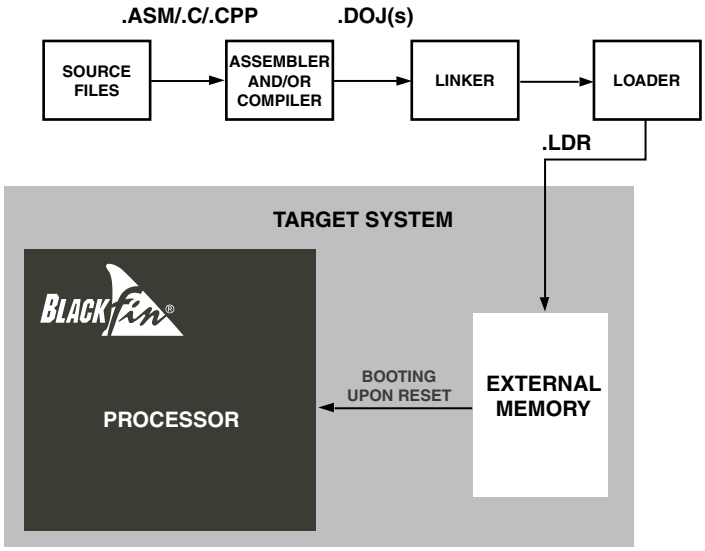


Figure 34-1: Project Flow

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 headers are read and parsed by the on-chip boot kernel 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.

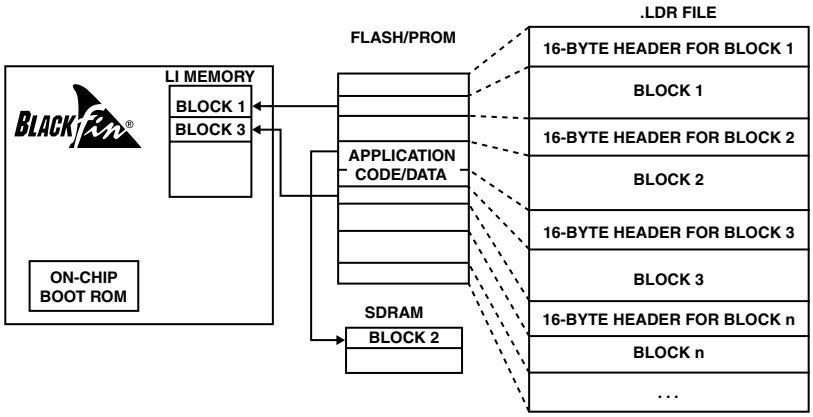


Figure 34-2: Booting Process

In some cases (for example when the BLFAG_INDIRECT flag for any block is set), the boot kernel uses another memory block in L1 data bank B for intermediate data storage. To avoid conflicts, the elfloader utility ensures this region is booted last.

The entire source code of the boot ROM is shipped with the tools. The source code can be reviewed for information not covered in this manual. Note that minor maintenance work may be required to the boot ROM code if silicon is updated.

Block Structure

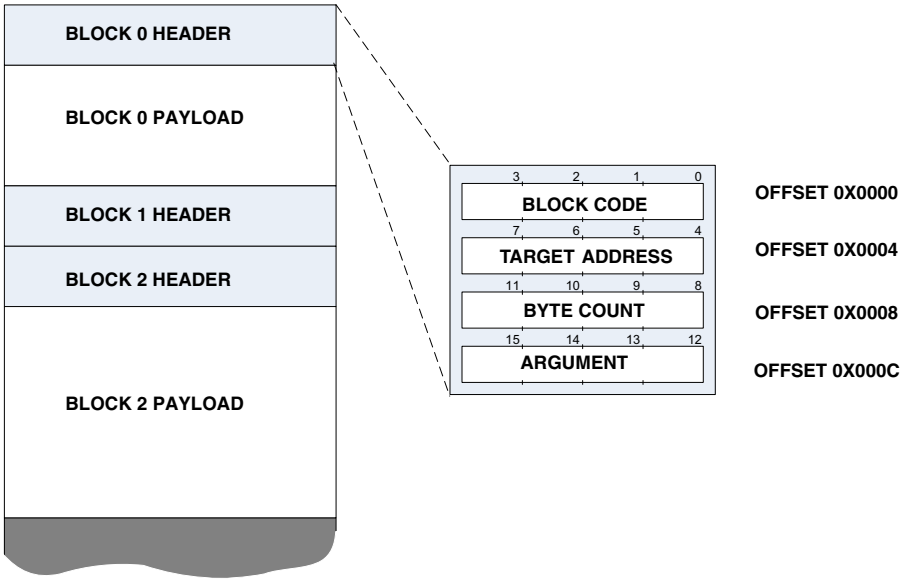


Figure 34-3: Boot Header

A boot stream consists of multiple boot blocks as shown in the figure. Every block is headed by a 16-byte block header. The 16 bytes are functionally grouped into four 32-bit words: the block code, target address, byte count, and the argument field. This section describes the fields in general. The uses may vary depending on the particular block type and boot mode, refer to the block type descriptions and boot modes for further information.

Block Code

Table 34-1: Block Header flags

Bit	Name	Description
0-3	BCODE	Specific to boot modes (see Boot Modes)
4	BFLAG_SAVE	Saves the memory of this block to off-chip memory in case of power failure or a hibernate request. This flag is not used by the on-chip boot kernel.
5	BFLAG_AUX	Nests special block types as required by special-purpose second-stage loaders. This flag is not used by the on-chip boot kernel.
6	reserved	
7	BFLAG_FORWARD	Forward payload to a device. Must be used in conjunction with the BFLAG_INDIRECT flag.
8	BFLAG_FILL	Fill the target location with a specified value.

Table 34-1: Block Header flags (Continued)

Bit	Name	Description
9	BFLAG_QUICKBOOT	Does not process block for a quick boot (warm boot).
10	BFLAG_CALLBACK	Calls function at the address provided.
11	BFLAG_INIT	Calls function at target address after loading payload to the same address.
12	BFLAG_IGNORE	Block is ignored.
13	BFLAG_INDIRECT	Boots to an intermediate storage place.
14	BFLAG_FIRST	Indicates the block to be the first block of a new .dxe
15	BFLAG_FINAL	Indicates the last block of a loader stream. Booting will complete after processing the block.
16-23	HDRCHK	A simple XOR checksum of the other 31 bytes in the boot block header.
24-31	HDRSIGN	0xAD; value is dependent on processor family.

TARGET_ADDRESS

The `TARGET_ADDRESS` holds the address that the block applies to, (where the code or data should be loaded). However, the field is interpreted differently depending on what specific flags are set in the block code. Refer to each Block Type's documentation 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.
- The`BFLAG_INDIRECT` flag must be set if the target address points to L1 instruction memory. For performance reasons, this is also recommended when booting to off-chip memory.

NOTE: Booting into scratchpad memory is not supported. If booting to scratchpad memory is attempted, the processor hangs within the on-chip boot ROM. Similarly, booting into the upper 16 bytes of L1 data bank A is not supported. These memory locations are used by the kernel for intermediate storage of block header information. These memory regions cannot be initialized at boot time. After booting, they can be used by the application during runtime.

BYTE_COUNT

The byte count must be divisible by 4, and also may be zero. This 32-bit field generally holds the size of the block. In some cases it can be used differently (such as when `BFLAG_FILL` is set). See the block types section for information on specific variations.

ARGUMENT

The 32-bit field is a user variable for most block types. The value is accessible by the initcode or the callback routine and can therefore be used for optional instructions to these routines.

The ARGUMENTfield is used in different ways by different block types. See the block type descriptions for further information.

Block Types

Describes how to use different types of loader blocks.

A loader stream is a set of linked blocks and each block is responsible for performing a certain function dependent on the block's type. A block type is defined by its flags in the block header. Operations include functions such as loading data, filling a memory region with data, forwarding data to a peripheral, and instructing the kernel to stop processing. This section describes each block type and how it is used within a boot loader stream.

Normal Block

A block's primary function is to load some 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 34-2: Flags

Flag	Required Value	
TARGET_ADDRESS	Y	Address where payload is loaded (must be valid)
BYTE_COUNT	Y	Size of block in bytes

First Block

A first block indicates the start of a DXE and is always required at the beginning of a loader stream. In the case of a loader stream that contains *Multi-DXE Boot Streams*, a first block occurring within the loader indicates the beginning of a new DXE.

When the kernel processes the first block in a loader stream, the TARGET_ADDRESS also updates the EVT1 register. This register contains the start address of the application.

NOTE: Note that a First Block cannot be combined with a Fill Block

Table 34-3: Flags

Flag	Required Value	
BFLAG_FIRST	Y	1
ARGUMENT	Y	Offset to the next DXE, or first address following loader stream
TARGET_ADDRESS	Y	When the block is the first block in a loader stream, also defines the start address for the application. If the block is not the first in a loader stream the target address is used as in normal operation.

Final Block

The final block marks the last block in a boot stream (not a DXE). After processing a final block the boot kernel jumps to the application's start address. By default this is performed by jumping to the vector stored in the `EVT1` register. The boot kernel provides options to execute an `RTS`instruction or a `RAISE 1`instruction. The default behavior can be changed by an initcode routine, or by the kernel's boot routine API. The `EVT1`register is updated by the boot kernel when processing the first block in the loader stream.

Before the boot kernel passes program control to the application it does some housekeeping. Most of the registers that were used are put in their default state. However, some register values may differ depending on the boot mode. See Boot Modes for more information. The following DMA configuration registers and primary register control registers `UART_CTL`, `SPI_CTL`, `HOST_CONTROL` for example) are restored, while others are purposely not restored. For example `SPI_CLK.BAUD` and `UART_CLK`remain unchanged so that the settings obtained during the boot process are not lost.

Table 34-4: Flags

Flag	Required Value	
BFLAG_FINAL	Y	1

Indirect Block

An intermediate block is first loaded to a storage location before being copied to the destination. This functionality is motivated by the following situations:

- Some boot modes may not use DMA. The core cannot access some memory locations directly (L1 instruction SRAM), and some it cannot access efficiently. An intermediate load to a different location improves overall efficiency.
- In some booting scenarios the data in the payload needs to be operated on or analyzed before it is fully loaded (such as decryption or checksum calculation). By using an intermediate location such scenarios are simplified and can be more efficient when loading to off-chip memories (see Callback Block).

In some cases a boot block may not fit into temporary storage memory so having a larger buffer may 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 `pTempBuffer`and `dTempByteCount` variables in the *`STRUCT_ROM_BOOT_CONFIG`* structure. The default region is at the upper end of a physical memory block. When the size is changed from the default, keep in mind that it is already at the end of a memory block, so the beginning address pTempBuffer must also be modified.

Table 34-5: Flags

Flag	Required Value	
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

An ignore block is a block that is (in most cases) ignored by the loader stream. Ignore blocks are useful when it's not possible to pass information in another block header. For example, if the first block contains data rather than application code, then inserting a first block with the correct application start address ensures the correct start address is used. Since this block has no other function it should be marked as an ignore block so that the kernel will not attempt to process any payload.

Table 34-6: Flags

Flag	Required Value	
BFLAG_IGNORE	Y	1
BYTE_COUNT	Y	Size of block to ignore, may be zero

Init Block

An init (initialization) block instructs the boot kernel to issue a `CALL` instruction to the target address after the entire block has been loaded. The function called is referred to as the *initcode routine*. If the initcode routine has been previously loaded, the block may 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 setup system PLL, bit rates, wait states, and the SDRAM controller. If executed early in the boot process, the boot time can be significantly reduced.

Initcode routines are required to follow the C language calling conventions. The expected C prototype is:

```
void initcode(STRUCT_ROM_BOOT_CONFIG* pBootStruct)
```

When programming in assembly, be certain to return using an `RTS` instruction. See the *C/C++ Compiler and Library Manual* for more information.

The struct provided to the initcode routine by the boot kernel contains a variety of information about the block being processed. This includes header information, locations of temporary block data (for indirect blocks), target address, and byte count. See *Booting Data Structures*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 should set the BFLAG_INIT flag. In the latter case, the last block should be of size zero, and simply instruct the boot kernel to issue the `CALL` instruction.

An initcode routine can be overwritten by a successive block if it is no longer needed, otherwise the routine can be called at multiple points during the boot process, and even remain in memory after booting is completed for use by the application.

NOTE: The following list provides requirements for initcode that is written in C or C++.

- Ensure 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 init 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 init-code routines. For details and more information on the utility, see to the *Loader Utility Manual*.

Initcode routine examples, including SDRAM controller initialization, are described in the *Boot Programming Model*.

Table 34-7: Flags

Flag	Required Value	
BFLAG_INIT	Y	1
TARGET_ADDRESS	Y	Location to load payload data. `CALL` instruction issued to the same location.
ARGUMENT	N	Can be used to supply block specific arguments
BYTE_COUNT	Y	Size of payload, may be zero

Callback Block

A callback block instructs the boot kernel to call a pre-registered function upon completion of loading the block's payload. 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 an init block must be created whose initcode modifies the `pCallBackFunction` pointer in the *`STRUCT_ROM_BOOT_CONFIG`* structure. A callback routine must be registered before a callback block can be processed.

Since callback routines require access to the payload data of the boot blocks, the block data must be loaded before it can be processed. Often an INDIRECT block is used in combination with a callback block.

Callback routines are expected to meet the C language calling conventions. For more information see the *C/C++ Compiler and Library Manual*. The prototype is as follows:

```
int32_t CallBackFunction(struct STRUCT_ROM_BOOT_CONFIG* pBootStruct,

    ADI_BOOT_BUFFER* pCallbackStruct, int32_t dCbFlags)
```

The `pBootStruct`argument contains the *`STRUCT_ROM_BOOT_CONFIG`* information, and the `pCallbackStruct` contains the target address and size of the block (may vary when using indirect). The dCbFlags parameter is specifically used when indirect is also used. The `CBFLAG_DIRECT`flag indicates that the `BFLAG_INDIRECT` bit is not active and so that the callback routine is only called once per block. When the `CBFLAG_DIRECT` is set, the `CBFLAG_FIRST` and `CBFLAG_FINAL` are also set.

See *Booting Data Structures*for more detailed information on these data.

Callback Block Used in Conjunction with Indirect Block

When a block using a callback routine is also loaded indirectly there are slight behavior differences. The procedure for loading is:

1. Data is loaded into the temporary buffer defined by the `pTempBuffer`.
2. The `CALL`to the `pCallBackFunction` is issued.
3. After the callback routine returns, if the return value is zero, the memory DMA copies data to the destination

If a block does not fit entirely into the temporary buffer, loading is performed similar to indirect blocks, and the callback function is called after each chunk is loaded into the temporary storage. The `dCbFlags` parameter gives information on the specific iteration.

When a block does not fit entirely into the temporary storage area, the `dCbFlags` tells the callback routine whether it is invoked for the first time (`CBFLAG_FIRST`) or whether it is called the last time (`CBFLAG_FINAL`) for a specific block.

When DMA is invoked to copy the data, it relies on the `pCallbackStruct` structure, not the global `pTempBuffer` and `dTempByteCount` variables. The callback routine can control the source of the memory

DMA by altering the content of the pCallbackStruct structure, as may be required if the callback routine performs data manipulation such as decompression.

When an indirect block is used, 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 34-8: Flags

Flag	Required Value	
BFLAG_CALLBACK	Y	1

Quick Boot Block

Quick Blocks are only processed for a full boot. In some booting scenarios, not all memories need to be re-initialized. For example, in a wake-up from hibernate state, off-chip SRAM may not be impacted if it is powered while the processor is in a hibernate state. Dynamic RAM may also not be impacted if it was put into a self-refresh mode before the processor powered down.

The boot kernel uses the following parameters to determine whether or not a quick block should be processed. See the "Global Boot Flow" figure also.

- The SYSCRregister is read to determine what kind of boot is expected from the boot kernel. The WURESET bit is compared against other reset bits to distinguish between cold boot and warm boot situations and to identify wake-up from hibernate situations.
- The BCODEbit field in the SYSCRregister can overrule the native decision of the boot kernel for a software boot.
- The BFLAG_WAKEUP bit in the dFlag word of the *STRUCT_ROM_BOOT_CONFIG* structure indicates that the final decision was to perform a quick boot. If the boot kernel is called from the application, then the application can control the boot kernel behavior by setting the BFLAG_WAKEUP flag accordingly.
- The BFLAG_QUICKBOOT flag in the block code word of the block header controls whether the current block is ignored for quick boot.

Table 34-9: Flags

Flag	Required Value	
BFLAG_QUICKBOOT	Y	1

Save Block

A save block saves the payload of the block to off-chip memory. This flag is not used by the on-chip boot kernel

Table 34-10: Flags

Flag	Required Value	
BFLAG_SAVE	Y	1

Forward Block

A forward block is similar to a normal block except that the payload is written to a specified peripheral rather than loaded to memory. Rather than specifying a target address, a forward port is defined and the data is forwarded accordingly. This feature can enable multiple processors in a system to be booted from the same source. This can be done in any hierarchical way using daisy chaining via supported peripherals. An entire boot stream can be contained in the payload of a block, and that entire stream can be forwarded to a peripheral.

The processor supports forwarding to the SPI and link ports. Any peripheral must first be initialized before any forward block that is using that peripheral is processed. This is accomplished by using the boot kernel's *Forward Config* (`FUNC_ROM_FWDCFG`) function. See the ROM API for more information on this function. The forward configuration function takes as arguments the specific peripheral to be initialized and performs all the required initialization for that peripheral to enable forwarding.

As mentioned earlier, the target address for forward blocks is a forward port rather than target address. The details of this parameter are peripheral specific. For purposes of forwarding, the `TARGET_ADDRESS` field in the block header is referred to as `dForwardPort`.

Forwarding to SPI

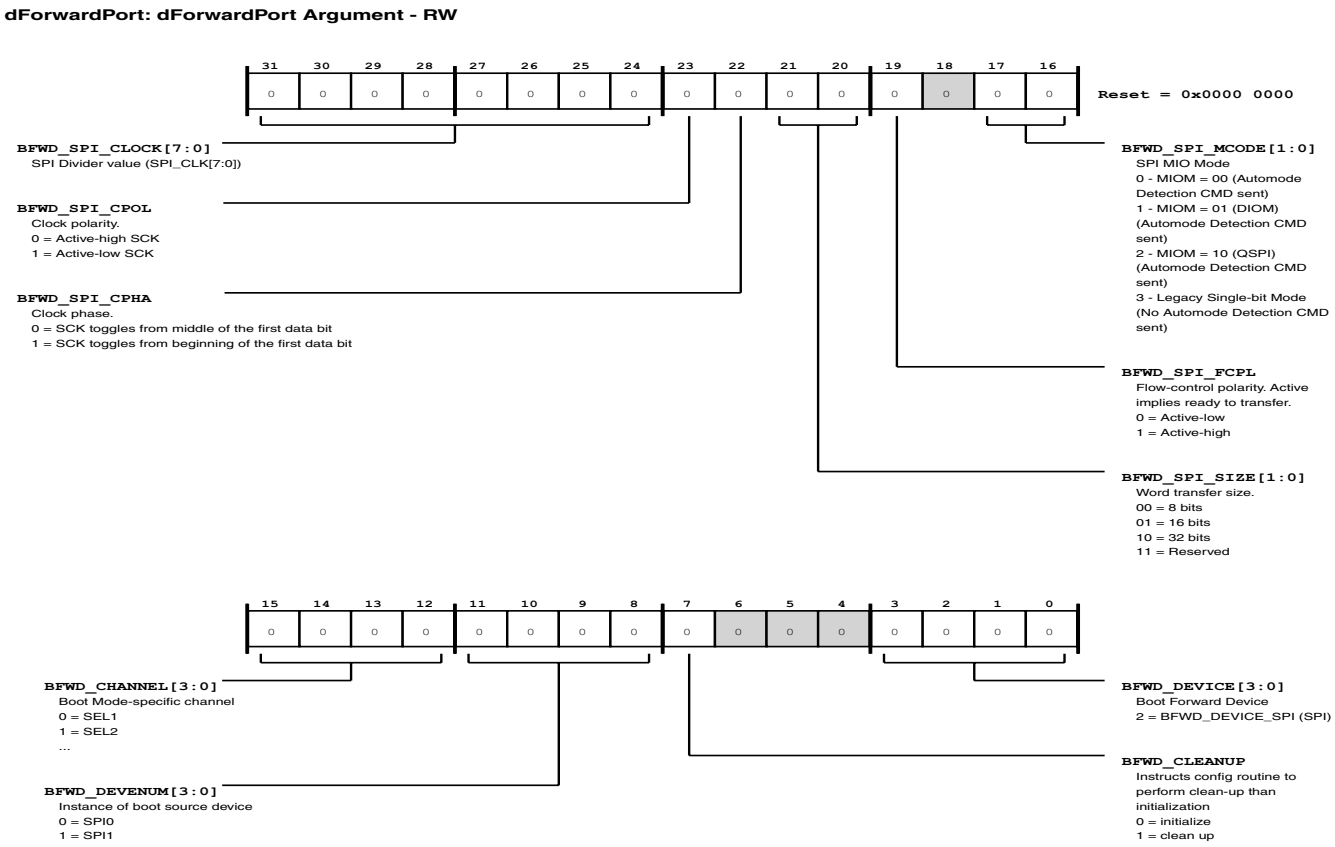


Figure 34-4: SPI dForward Port

The *SPI dForward Port* figure describes the various options available in the `dForwardPort` field for SPI block forwarding.

- For forwarding a boot stream to the SPI port, the `BFWD_DEVICE` field in the dForwardPort argument needs to be set to `BFWD_DEVICE_SPI`. The device source can be SPI0 or SPI1, depending on the `BFWD_DEVENUM` field.
- When using both device sources, the Forward Config API routine needs to be called twice.
- A source device can serve multiple SPI slave devices and is selectable using the SPI SSELx signal (set in `BFWD_CHANNEL`). If multiple slave devices are connected to the FCS signal (`SPI_RDY`), a pull-down resistor is absolutely required as the SPI slave devices have the `SPI_CTL.ODM` bit set to avoid conflicts on the SPI bus. The `BFWD_SPI_FCPL` register must be (set/cleared).

The `BFWD_SPI_MCODE` field is used to choose the I/O mode; 0, 1, or 2 choose single-bit, dual-bit, or quad-bit mode. Setting `BFWD_SPI_MCODE` to 0, 1, or 2 also enables the automode detection command in front of the boot stream. See SPI Slave Boot specification for further details.

The `BFWD_SPI_MCODE` register selects the legacy single-bit mode without sending any automode detection command. The `BFWD_SPI_CLOCK` register is required to set the lower 8 bits of the BAUD field in the `SPI_`

CLOCK divider register. The upper 8 bits of the BAUD field are set to 0. The other fields are used to directly manipulate specific SPI Control register settings.

Forwarding to the Link Ports

dForwardPort: dForwardPort Argument - RW

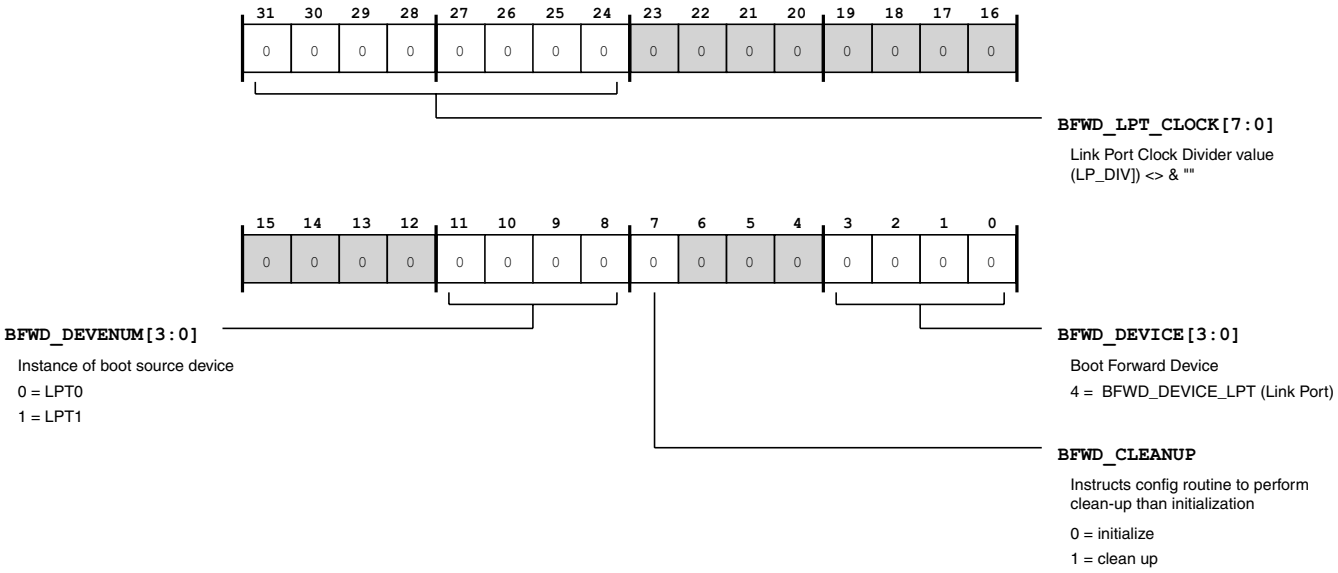


Figure 34-5: Link Port dForward Port

The link port dForward port describes the various options available in the dForwardPort field for link port block forwarding. For forwarding data to any of the link ports, the BFWD_DEVICE field in the dForwardPort argument needs to be set to BFWD_DEVICE_LPT. The BFWD_DEVENUM field selects whether data is forwarded to LPT0, LPT1, LPT2 or LPT3. Data can be forwarded to multiple link ports. However, the Forward Config routine needs to be called for each link port separately. The only programming option for Forwarding through link ports is the BFWD_SPI_CLOCK field in the dForwardPort argument. The Forward Config routines copies the value into the respective LPx_DIV register. The Forward Callback routine does not evaluate this bit field. The Forward Config routine automatically sets the pin muxing as required and initializes the link port peripheral.

Table 34-11: Flags

Flag	Required Value	
BFLAG_FORWARD	Y	1
BFLAG_INDIRECT	Y	1
TARGET_ADDRESS	Y	Contents of the peripheral specific dForwardPortparameter

Multi-DXE Boot Streams

This section describes Multi-DXE boot loader streams.

As seen by the elfloader, a boot stream is always generated from a DXE file. It is therefore common to refer to multi-DXE or multi-application booting. When the elfloader utility accepts multiple DXE files on its command line, it generates a contiguous boot image by default. The second boot stream is appended immediately to the first one. Since the utility updates the ARGUMENT field of all BFLAG_FIRST blocks, the ARGUMENT field of a BFLAG_FIRST block is called the next-DXE pointer.

The next-DXE pointer of the first DXE boot stream points relatively to the start address of the second DXE boot stream. A multi-DXE boot image can be seen as a linked list of boot streams. The next-DXE pointer of the last DXE boot stream points relatively to the next free address. This is illustrated by an example shown in the *Multi-DXE Boot Stream Example*figure.

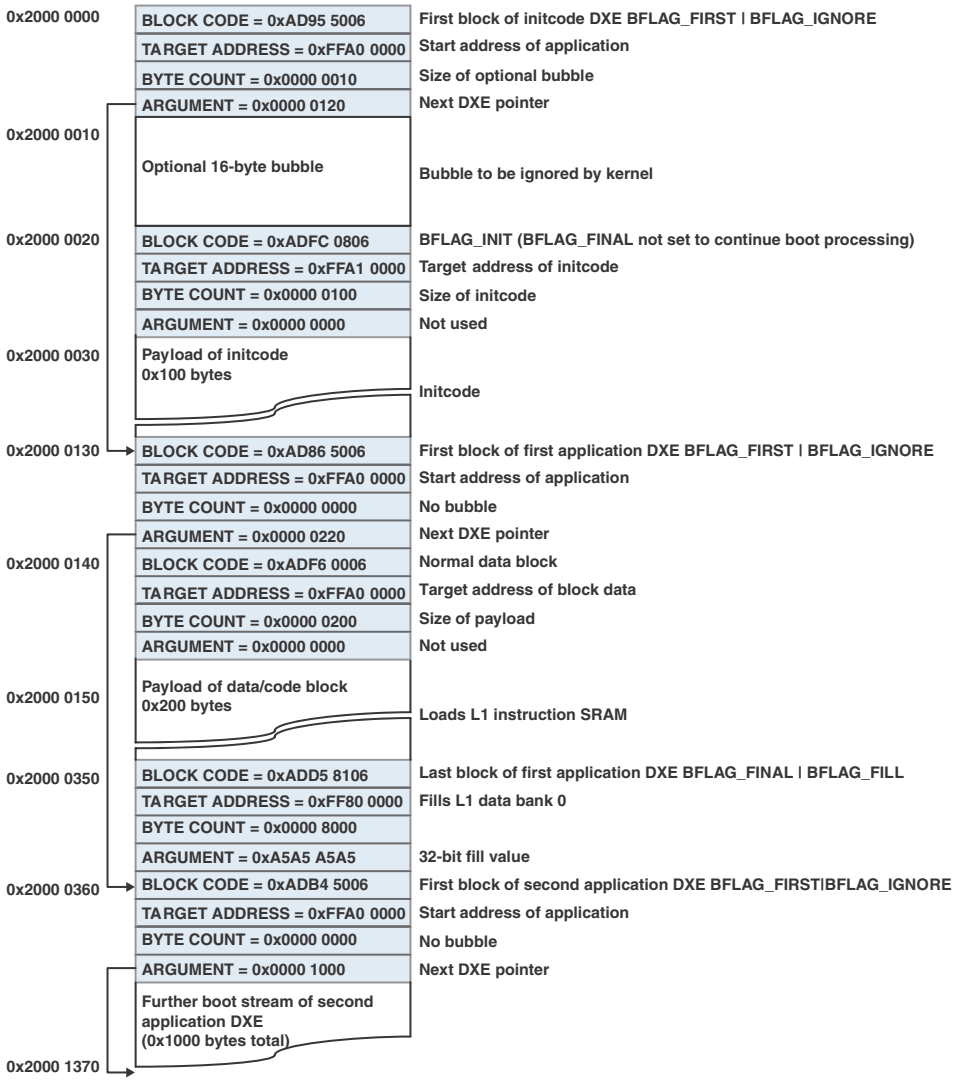


Figure 34-6: Multi-DXE Boot Stream Example

The *Multi-DXE Direct Code Execution Example*figure shows a linked list of initial block headers that 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 ROM that prevents multi-DXE applications from mixing regular boot streams and direct code execution blocks.

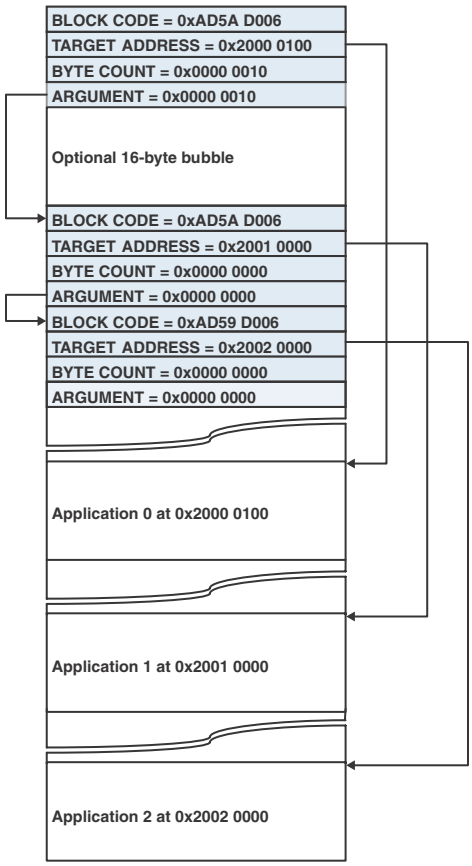


Figure 34-7: Multi-DXE Direct Code Execution Example

Single-Block Boot Streams

This section describes how to bypass booting and execute code directly from SDRAM.

The simplest boot stream consists of a single block header and one contiguous block of instructions. When the appropriate flags are set in the block header, the kernel loads the block to the target address, terminates, and begins executing from the target address of the block.

The *Initial Header for Single-Block Stream*table shows an example of a single-block boot stream header settings that can be loaded using any boot mode. The `BFLAG_FIRST` and `BFLAG_FINAL` flags are both set at the same time and the target address and byte count are determined by the desired location of the application.

Table 34-12: Initial Header for Single-Block Stream

Field	Description of Value
BLOCK_CODE	0xAD000000|XORSUM|BFLAG_FINAL|BFLAG_FIRST
TARGET_ADDRESS	Start address of block and application code
BYTE_COUNT	number of bytes in the block
ARGUMENT	Functions as next-DXE pointer in multi-DXE boot streams.

Direct Code Execution

Applications may want to avoid long booting times and start code execution directly from flash or SDRAM memory. 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 safety mechanisms of the block, such as the header signature and the XOR checksum, are used to avoid unpredictable processor behavior when boot memory is not yet being programmed with valid data. Rather than blindly executing code, the boot kernel first executes the preboot routine for system customization, then loads the first block header and checks it for consistency. If the block header is corrupted, the boot kernel goes into a safe idle state and does not start code execution.

If the initial block header check is good, the boot kernel interrogates the block flags. If the block has the BFLAG_FINAL flag set, the boot kernel immediately terminates and jumps directly to the address stored in the `RCU_SVECT0` register. To cause the boot kernel to customize the `RCU_SVECT0` register in advance, the first block must also have the `BFLAG_FIRST` flag set. The target address field is then copied to the `RCU_SVECT0`register. In this way the target address field of the initial block defines the start address of the application.

For example, if BMODE = 1, when the block header described in the Initial Header table is placed at address 0x20000000, the boot kernel is instructed to issue a JUMP command to address 0x20000020.

The development tools must be instructed to link the above block to address 0x20000000 and the application code to address 0x20000020. An example shown in "Direct Code Execution" illustrates how this is accomplished using the tools suite.

Table 34-13: Initial Header

Field	Value	Comments
BLOCK_CODE	0xAD7BD006	0xAD000000|XORSUM|BFLAG_FINAL|BFLAG_FIRST|BFLAG_IGNORE|(MDMACODE & 0x6)
TARGET_ADDRESS	0x20000020	Start address of application code
BYTE_COUNT	0x00000010	Ignores 16 bytes to provide space for control data such as version code and build data. This is optional and can be zero.
ARGUMENT	0x00000010	Functions as next-DXE pointer in multi-DXE boot streams.

Boot Modes

No-Boot Mode

When the BMODE pins are all tied low (BMODE=0000), the processor does not boot. Instead it throws an emulation interrupt followed by an idle instruction. The purpose of this mode is to bring the processor up to a clean state from any unwanted state that may have been caused by incorrect programming of the boot source memory.

When connecting to an emulator and starting a debug session, the processor uses the emulation interrupt to wake up the processor and it can be debugged in the normal manner.

Memory Boot Mode

The memory booting modes are intended for booting from flash, EEPROM, or any memory-mapped devices. Although this is a single BMODE setting, there are various configurations described in the MDMA Codes table.

By default the boot kernel does not alter any static memory controller (SMC) registers. Therefore, traditional asynchronous memory is assumed and maximum wait states are applied. Once the processor boots, the boot kernel loads an initial burst of 160-bit words, and interrogates the loaded MDMACODE. MDMACODE is the specialized version of the BCODE field defined in a generic block header.

The MDMACODE is filled by the elfloader utility based on boot mode, -width and -dmawidth settings. See the Loader and Utility Manual for more details.

NOTE: If the NOR flash device has been put into burst or page mode, it must be programmed back to the standard mode before the processor is reset. If the processor can reset itself without software control (via watchdog or double-fault error) a mechanism must be installed that also resets the flash device back to its default mode along with the processor.

Hardware configurations for the individual modes are shown in the 8/16-bit Flash Interconnection figure. The chip select is always controlled by the AMS0 strobe. Some flash devices provide write protection mechanisms, which can be activated during the power-up and reset cycles on the processor. In the absence of

such mechanisms, a pull-up resistor on the AMS0 pin prevents the chip select from floating when the state of the processor is unknown.

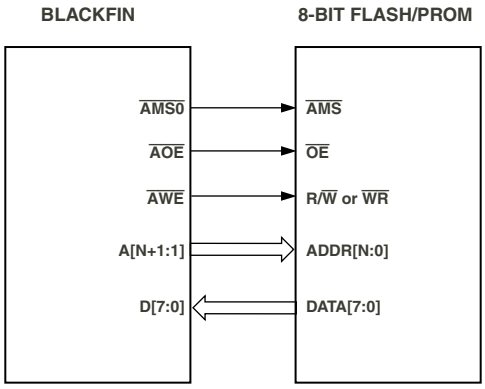


Figure 34-8: 8-bit Flash Interconnection

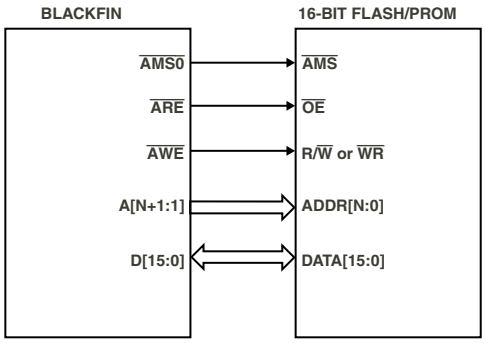


Figure 34-9: 16-bit Flash Interconnection

Table 34-14: MDMA Codes

MDMACODE	Word Width	Source Modify	Purpose
0x0			reserved, protects against legacy 10-byte boot streams
0x1	8 bits	1	8-bit flash on 8-bit SMC
0x2	8 bits	2	8-bit flash on 16-bit SMC
0x3	8 bits	4	8-bit flash on 32-bit SMC
0x4	8 bits	8	reserved - not supported
0x5	8 bits	16	reserved - not supported
0x6	16 bits	2	16-bit flash on 16-bit SMC
0x7	16 bits	4	16-bit flash on 32-bit SMC
0x8	16 bits	8	reserved - not supported
0x9	16 bits	16	reserved - not supported
0xA	32 bits	4	32-bit flash on 32-bit SMC
0xB	32 bits	8	reserved - not supported
0xC	32 bits	16	reserved - not supported
0xD	64 bits	8	reserved - not supported
0xE	64 bits	16	reserved - not supported
0xF	128 bits	16	reserved - not supported

To illustrate how the MDMACODE effects reading memory, consider eight contiguous bytes A through H in the following scenarios. In the first, a 16-bit device is used on the 16-bit SMC, stored as BA, DC, FE, HG.

Table 34-15: MDMACODE effect on memory read

MDMACODE	Read by kernel as
0x1	A, B, C, D, E, F, G, H
0x6	BA, DC, FE, HG
0xA	DCBA, HGFE

Padding Memory

To ensure data is read properly for the specific configuration, it may be necessary to pad data when storing it in the device. A few examples follow.

If an 8-bit flash device is connected to the 16-bit SMC only the lower byte is populated and the upper byte needs to be ignored by the kernel. This is ensured by setting `MDMACODE = 0x2`. To use a 16-bit flash to simulate this, the upper byte is usually zero padded. The 16-bit flash device is programmed with the content of 0A, 0B, 0C, 0D, 0E, 0F, 0G, 0H.

Similarly, setting `MDMACODE = 0x3` supports 8-bit flash on 32-bit SMC. This case can be simulated by connecting a 32-bit flash to a 32-bit SMC, or by connecting a 16-bit flash to a 16-bit SMC. In either case, the flash is programmed with three padding bytes per data byte: 000A, 000B, 000C, 000D, 000E, 000F, 000G, 000H.

Finally, setting `MDMACODE=0x7` supports 16-bit flash on a 32-bit SMC. This case can also be simulated by connecting a 32-bit flash to a 32-bit SMC, or by connecting a 16-bit flash to a 16-bit SMC. This time 16-bit padding is required per 16-bit data byte: 00BA, 00DC, 00FE, 00HG.

Auto Detection

An automatic configuration detection is provided that uses the first nibble of the boot stream (`MDMACODE` field). The boot code requests the initial eight bytes by 32-bit `MDMA`, inspects the `MDMACODE` and then adjusts to 8-bit or 16-bit `MDMA` operation as needed. During this detection sequence, external memory addresses are fetched sequentially; alternatively a sequential device (such as a FIFO) can be connected.

When the Boot Routine API function is called at run time, the same auto-detection is performed. If auto-detection is not wanted, it can be suppressed by the `BCMD_NOAUTO` switch. See the Run-time API section for details. It is important to understand the `BCMD_NOAUTO` enables the `MDMACODE` to be changed at runtime. If data is arranged on the same flash device in different ways, its perfectly valid to rotate though MDMACODE settings 0x01, 0x6 and 0xA, because all three `MDMACODE`s can operate on the same physical arrangement and the same flash image. However, setting `MDMACODE` to 0x2, 0x3 and 0x7 requires a very specific format and are therefore not interchangeable.

Default Static Memory Controller (SMC) settings

In Memory Boot mode, the boot kernel attempts to boot from the ASYNC Bank 0 Memory Space. Bank 1, Bank 2 and Bank 3 are not initialized. If required, they can be initialized by initcode. The SMC registers are set by default as described in the Default SMC Settings table.

Table 34-16: Default SMC Settings

Register Bit	Value	Note
SMC_B0CTL_EN	1	enable bank AMS0
SMC_B0CTL_MODE	0	normal asynchronous mode
SMC_B0CTL_RDYEN	1	enable ARDY
SMC_B0CTL_RDYPOL	1	ARDY is active high
SMC_B0CTL_RDYABTEN	0	disable ARDY abort counter
SMC_B0CTL_PGSZ	0	4-byte page size
SMC_B0CTL_BLK	1	NOR clock = SCLKA/2
SMC_B0CTL_BYTPE	0	wrap bursts
SMC_B0CTL_SELCTRL	0	normal AMS0 strobe
SMC_B0TIM_WST	7	7*8ns = 56ns
SMC_B0TIM_WHT	7	7*8ns = 56ns
SMC_B0TIM_WAT	9	9*8ns = 72ns
SMC_B0TIM_RST	7	7*8ns = 72ns
SMC_B0TIM_RHT	1	1*8ns = 8ns
SMC_B0TIM_RAT	8	8*8ns = 64ns
SMC_B0ETIM_PREST	1	1*8ns = 8ns
SMC_B0ETIM_PREAT	3	3*8ns = 24ns
SMC_B0ETIM_TT	2	2*8ns = 16ns
SMC_B0ETIM_IT	2	2*8ns = 16ns
SMC_B0ETIM_PGWS	9	9*8ns = 72ns

Run-time API

A Memory Boot can be initiated through the Boot Routine API function at run time. Any start address which is divisible by 4 can be specified. The API allows customizations such as booting from a different SMC bank, and disabling auto-detection.

Memory Boot is selected by specifying the BCMD_DEVICE_MEMORY | BCMD_MEMORY in the boot command argument. If BCMD_DEVENUM_0 is specified, the routine extracts from the start address which SMC bank to

configure. If, however, `BCMD_DEVENUM_1` is specified, the BCMD_MEM_SMCBANK field selects the SMC bank which is configured.

In either case the routine also initializes pin muxing as required unless the BCMD_NOCFG flag is specified, which overrules BCMD_DEVENUM and BCMD_MEM_SMCBANK. If the BCMD_NOCFG flag is specified, it is the programs responsibility to configure SMC and pin muxing as required. This is the only way to boot in burst, page or (the so-called) flash mode. The BCMD_NOCFG option is useful if a different application (dxe) has been booted earlier and the SMC and pin muxing have already been set up.

The `MDMACODE` configuration detection can be suppressed by the BCMD_NOAUTO switch. In this case, the desired configuration can be passed through the BCMD_MODE bit field. It follows MDMACODE through table convention.

RSI Master Boot Mode

The RSI master boot mode loads a loader stream through the Removable Storage Interface (RSI). After reset or power on, the boot kernel performs auto-detection of the device to determine its type. Device types supported by auto-detection are:

- Secure Digital (SD), low/high capacity, 1/4-bit
- Multi Media Card (MMC), low/high capacity, 1/4-bit
- MMC plus, low/high capacity, 1/4/8-bit
- Embedded MMC (eMMC) low/high capacity, 1/4/8-bit as aaaaa

The boot kernel optimizes clock settings based on an RSICODE provided in the first nibble of the first block (defined as BCODE code in the block header definition). When booting from RSI, pagemode is used to load data. This provides better support for block based devices (most RSI devices are block based).

Every block loaded in RSI boot mode is loaded implicitly as an indirect, callback block. A callback routine is not required, but may be useful. An indirect callback block causes the boot kernel to first load pages to a temporary buffer where a callback routing may be used to load the data to a file system or perform other modifications. If no callback routine is specified the block is simply loaded. The Boot Routine API allows more advanced customization for booting from RSI. Using this API, auto-detection can be disabled and the device can be specified and configured manually.

PageMode Customization

Devices attached to the RSI peripheral are typically block based devices. The boot kernel has built-in support for booting from this type of block based device called pagemode. In pagemode data is loaded by fixed block sizes. The default page size used is 512 bytes. This value can be changed using the initcode routine to modify the `dTempByteCount` parameter in the `struct STRUCT_ROM_BOOT_CONFIG` structure. Refer to pagemode and Init Block for details on making these changes.

Callback Customization

Frequently, devices connected to the RSI have filesystem implemented on them. The motivation for implicitly enabling a callback for every block is to ease implementation on such systems. By default the callback routine is empty and simply returns a value of zero. The standard procedure for loading and registering a callback function should be used early in the stream. See the callback block section for more detailed information on performing this customization.

RSI Code

When performing auto-detection, the first block of data is loaded using the bus width defined by the `RSICODE`. The RSI code is defined by the first block in the boot stream and is located in the block header where `BCODE` is defined in a generic block code, (see Block Code). The boot kernel then reconfigures the RSI peripheral according to the value detected in the `RSICODE`.

Table 34-17: RSI Code

RSICODE	uwFlag	ubBusWidth	ubRsiClk	ubRsiClk High Speed	Description
0	0x0000	0x00	0x3	N/A	1 bit, regular speed protocol
1	0x0020	0x00	0x3	0x1	1 bit, high speed protocol(1)
2	0x0000	0x01	0x3	N/A	4 bit, regular speed protocol
3	0x0020	0x01	0x3	0x1	4 bit, high speed protocol(1)
4	0x0000	0x02	0x3	N/A	8 bit, regular speed protocol(2)
5	0x0020	0x02	0x3	0x1	8 bit, high speed protocol(1)(2
6-15	reserved				

Run-Time API

The RSI Boot mode can also be initiated through the Boot Routine API. The boot routine allows for further customization by modifying values in the `dBootCommand` parameter. This can be useful when it is necessary to bypass auto-detection of the device.

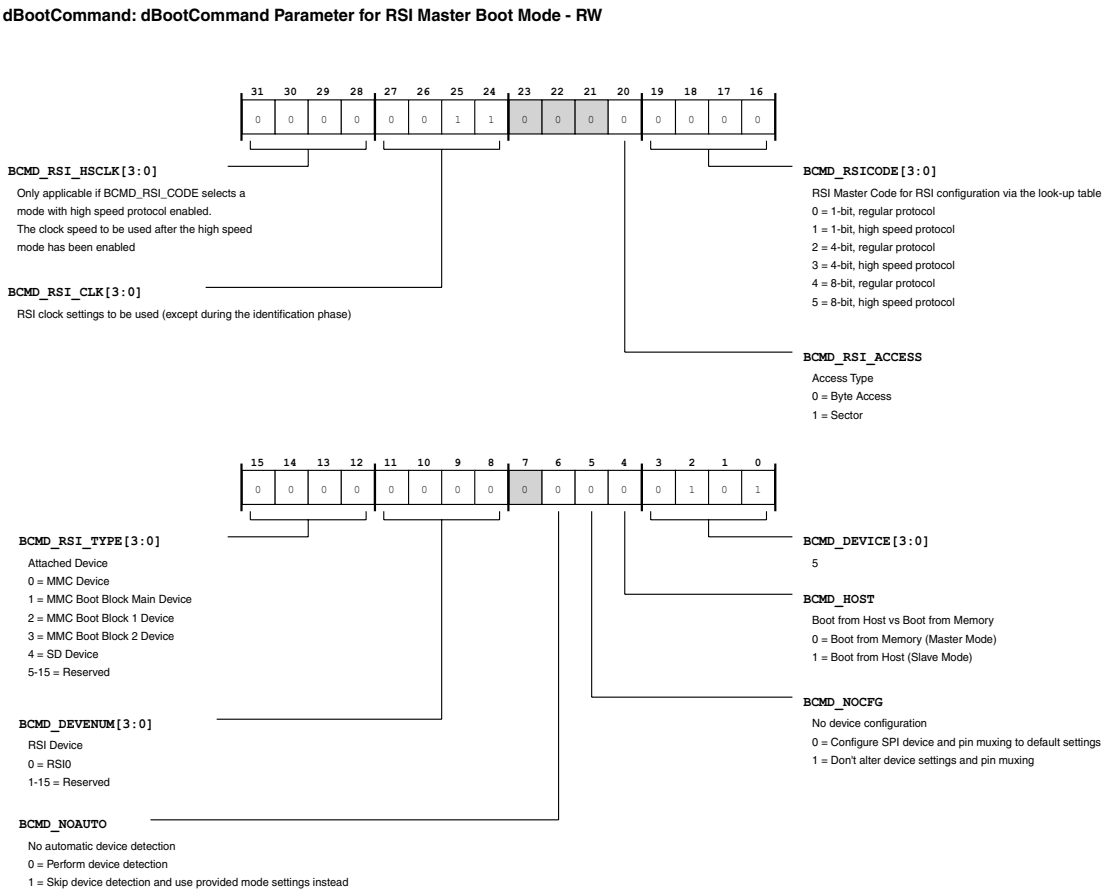


Figure 34-10: dBootCommand for RSI Boot

When BMC_NOAUTO is set, the RSI clock divider is taken from the BCM_RSI_CLK field. The device type of the attached device is taken from the BCMD_CHANEL field and the access type is taken from the ACCESS field. The RSICODE field is still applicable and processed in order to set the bus width and speed protocol to use.

NOTE: The device identification phase is not executed. Therefore the attached device must be in the transfer state before executing the API call.

In order to communicate with an attached device that is in the transfer state, the RCA of the device is required. For example, CMD13 reads the card's status in the event the switch command was executed to change the bus width or timings of the device. The card's RCA is therefore required to be written to the uwRca field of the RSI structure via the use of the hook function that can be passed to the boot kernel via the API. Failure to do so may lead to a boot failure

For MMC devices that support boot blocks, the contents of byte 179 of the Extended CSD register should be written to the ubBootPartByte of the RSI structure. This should be preformed via the hook function. In particular bits [7:3] should be set accordingly. Bits [2:0] may be modified by the boot code depending on the device passed via the BCMD_RSI_TYPE field of dBootCommand.

The boot kernel uses a byte write operation to write byte 179 of the Extended CSD register which impacts the non-volatile bits of the byte. When BCMD_NOAUTO is set, the Extended CSD register is not read which speeds up the process. Set the non-volatile bits accordingly. Programs may do this by reading the Extended CSD register at some point in the application and passing in byte 179 accordingly.

Notes on eMMC

The boot kernel includes support to handle boot partitions on eMMC devices. The extended CSD register is a register on the eMMC device with a byte containing boot partition information. The boot kernel uses this byte to activate the required partition accordingly to then get the boot stream. While the boot partition may be small, it may be sufficient to store something like UBOOT in loader stream format. UBOOT can be loaded from this special boot partition and then it can continue to boot having knowledge of the main file system are of the eMMC.

SPI Master Boot Mode

Describes booting from the Serial Peripheral Interface (SPI).

This SPI Master Boot mode boots from SPI memory connected to the SPI0 interface. 8-bit, 16-bit, 24-bit and 32-bit address words are supported. Standard SPI memory is read using either the standard 0x03 SPI read command or the 0x0B SPI fast read command.

For booting, the SPI memory is connected as shown in *SPI Memory Connections*

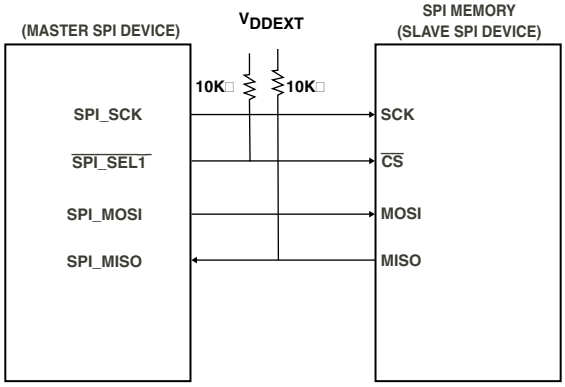


Figure 34-11: SPI Memory Connections

The pull-up resistor on the MISO line is required for automatic device detection. The pull-up resistor on the SEL1 line ensures that the memory is in a known state when the GPIO is in a high-impedance mode such as during reset. A pull-down resistor on the SCK line displays cleaner oscilloscope plots during debugging

For SPI master boot the SPE, MSTR and SZ bits are set in the SPI0_CTL register. The TIMOD=2 bits enable the receive DMA mode. Clearing both the CPOL and CPHA bits results in SPI mode 0. The boot kernel does not allow SPI0 hardware to control the SEL1 pin. Instead, this pin is toggled in GPIO mode by software. Initcodes are allowed to manipulate the uwSsel variable in the *STRUCT_ROM_BOOT_CONFIG* structure to

extend the boot mechanism to a second SPI memory connected to another GPIO pin.

SPI Device Detection Routine

Since `BMODE = 011` 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 an 8, 16, 24 or 32-bit addressing scheme, the boot kernel performs a device detection sequence prior to booting. The MISO signal requires a pull-up resistor, since 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 a read command (either 0x03 or 0x0B) on the MOSI line, which is immediately followed by two zero bytes. Once the transmission is finished, the boot kernel interrogates the data received on the MISO line. If it does not equal 0xFF, the valid byte (0x1-0xE in the lower nibble) tells the boot code whether the memory device requires 8, 16, 24, 32 address bits. This is referred to as the `SPIMCODE`. The boot kernel has the following settings according the *SPIMCODE Descriptions* table.

If the received value equals 0xFF, it is assumed that the memory device has not driven its data output and that the 0xFF value is due to the pull-up resistor. 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 if a byte different than 0xFF is received. The boot kernel continues with normal boot operation and it reissues a read command to re read from address 0. The first block header is loaded by two read sequences, further block headers and block payload fields are loaded by separate read sequences.

Figure *SPI Device Detection Principle* illustrates how individual devices behave.

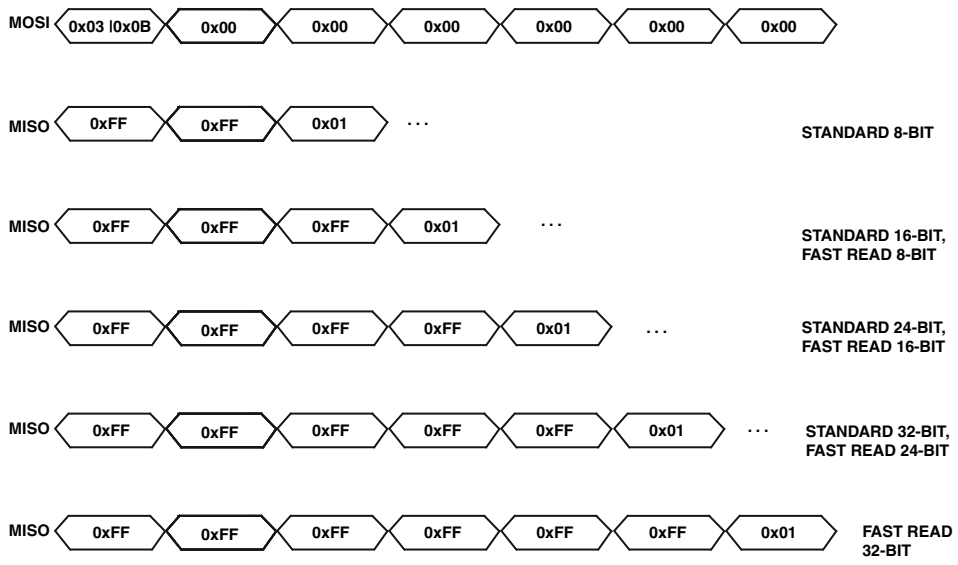


Figure 34-12: SPI Device Detection Principle

Table 34-18: SPIMCODE Descriptions

SPIMCODE	Mode	Command	Dummy Bytes	Data Lines	Address Lines	SPI Clock	Purpose
0x0							unused
0x1	STANDARD	0x03	0	1	1	SCLK/32	legacy single-bit SPI mode
0x2	STANDARD	0x03	0	1	1	SCLK/5	legacy single-bit SPI mode
0x3	STANDARD	0x0B	1	1	1	SCLK/2	single bit with dummy address byte
0x4	FAST_MODE	0x0B	1	1	1	SCLK/2	single bit with dummy address bytes
0x5	FAST_MODE	0x03	0	1	1	SCLK/3	Atmel mode
0x6	FAST_MODE	0x0B	1	1	1	SCLK/1	Atmel mode with dummy address byte
0x7	FAST_ MODE(RAPI DS)	0x1B	2	1	1	SCLK/1	Atmel mode with 2 dummy address bytes
0x8	DOR FAST_ MODE	0x3B	1	2	1	SCLK/2	dual bit data, single bit address
0x9	DIOR FAST_ MODE	0xBB	1	2	2	SCLK/2	dual bit data, dual bit address
0xA	QOR FAST_ MODE (Atmel)	0x6B	1	4	1	SCLK/2	quad bit data, single-bit address
0xB	QIOR FAST_ MODE (Atmel)	0xEB	3	4	4	SCLK/2	quad bit data, quad-bit address (AT25DQ)
0xC	QOR FAST_ MODE	0x6B	1	4	1	SCLK/2	quad bit data, single-bit address
0xD	QIOR FAST_ MODE	0xEB	3	4	4	SCLK/2	quad bit data, quad bit address
0xE	reserved						
0xF							unused

Run-time API

The Boot Routine API function can be used to initiate a boot through the SPI device. Using the API, further customizations can be made such as selecting a device other than SPI0, or disabling automated device initialization and pin muxing. Refer to the specific flags in the *dBootCommand for RSI Boot* parameter.

dBootCommand: dBootCommand Parameter for SPI Master Boot Mode - RW

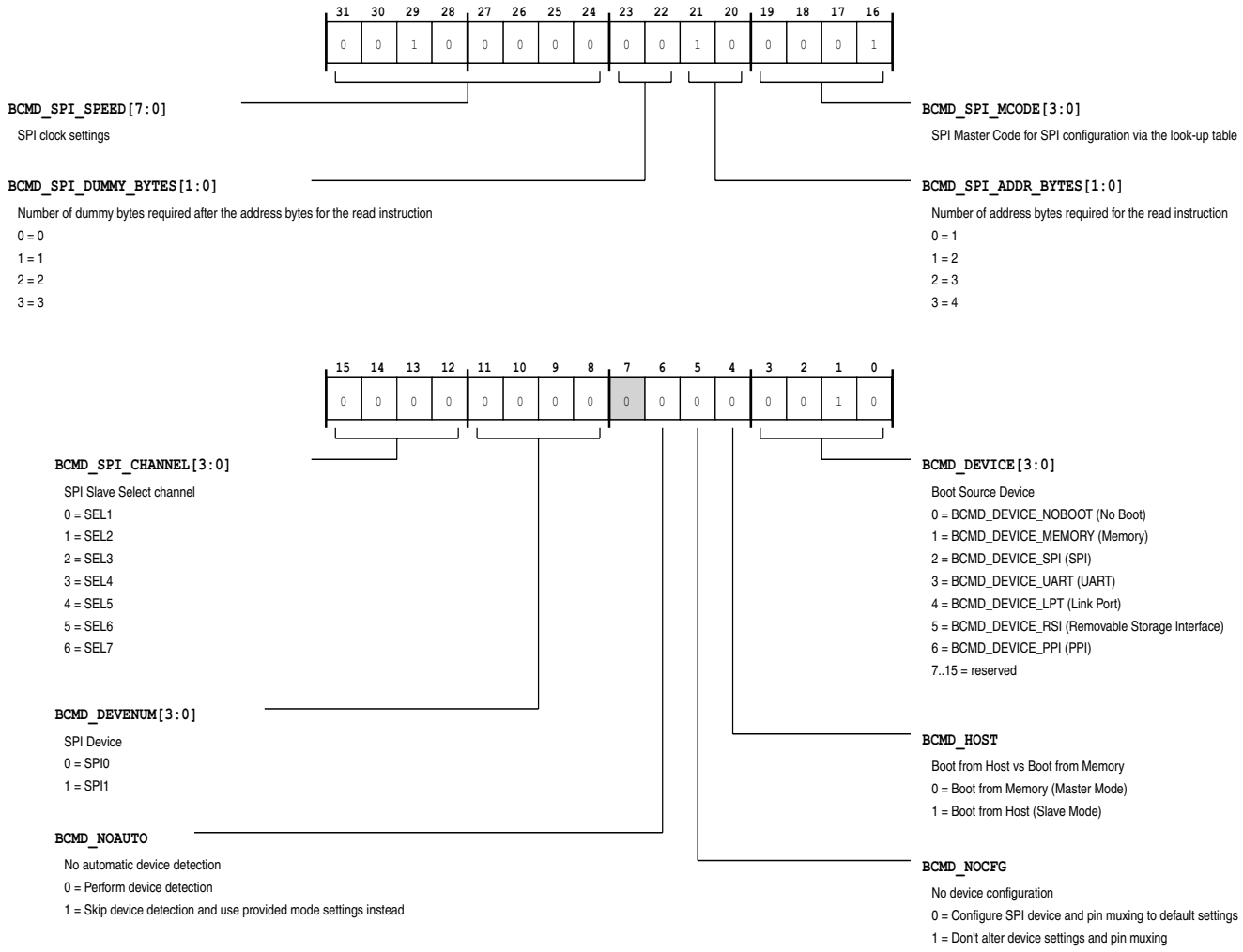


Figure 34-13: dBootCommand for RSI Boot

SPI Slave Boot Mode

Describes booting from the Serial Peripheral Interface (SPI) with the processor as a slave.

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 hardware configuration for the modes is shown in the following figures. As in all slave boot modes, the host device controls the Blackfin processor's RESET input.

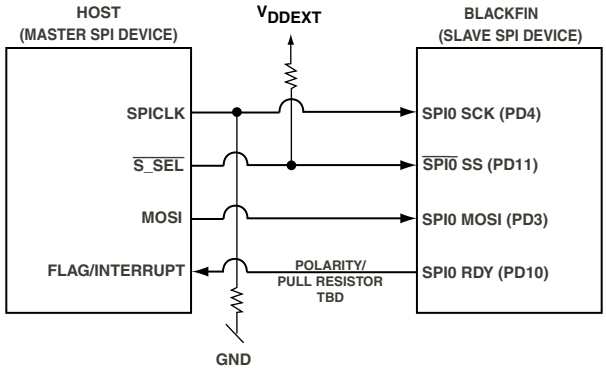


Figure 34-14: Connection Between Host (SPI Master) and Processor (SPI Slave)

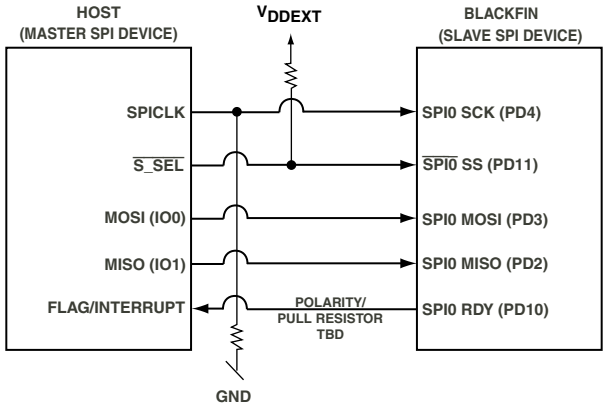


Figure 34-15: Connection Between Host (SPI Master) and Processor (SPI Slave) DIOM

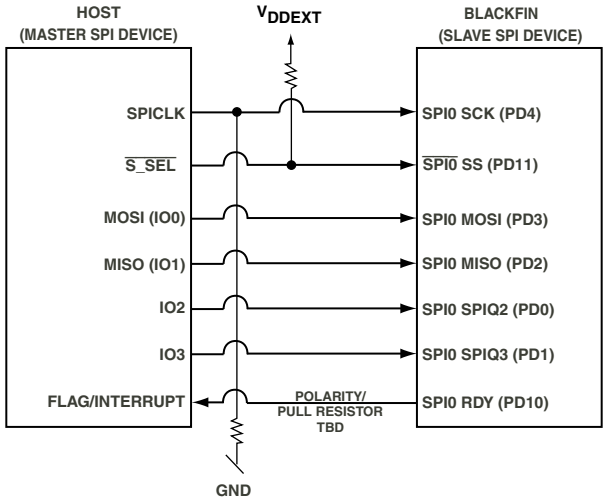


Figure 34-16: Connection Between Host (SPI Master) and Processor (SPI Slave) QSPI

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 SPI0 SS input of the Blackfin processor. It can toggle 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 CPHA bit and clears the CPOL bit in the SPI_CTL register. Therefore the MISO pin is latched on the falling edge of the MOSI pin. For details see "SPI Compatible Port Controllers" in the Processor Hardware Reference.

The SPI slave processor detects the correct bit mode from the host SPI device by reading the first byte sent, defined as the SPICMD. The following table describes the available codes. If the host starts in dual or quad-bit mode, additional bytes need to be sent to transmit the correct code.

Table 34-19: SPICMD Descriptions

SPICMD	Description
Starting in Single bit Mode	
0x3	keep single-bit mode
0x7	switch to dual-bit mode
0xB	switch to quad-bit mode
If host device starts in DIOM or QSPI	
0xAA,0xBF	switch to dual-bit mode
0xEE,0xEE,0xFE,0xFF	switch to quad-bit mode

In SPI slave boot mode, SPI_RDY functionality is critical. The SPI_RDY output is used for back pressure and requires a pull-up resistor. When high, the resistor shown in Figure programs SPI_RDY to hold off the host. SPI_RDY holds the host off while the Blackfin processor is in reset or executing the preboot. Once SPI_RDY turns inactive, the host can send boot data. The SPI module does not provide very large receive FIFOs, so the host must test the HWAIT signal for every byte. Figure *SPI Slave Boot Mode* illustrates the

required program flow on the host side.

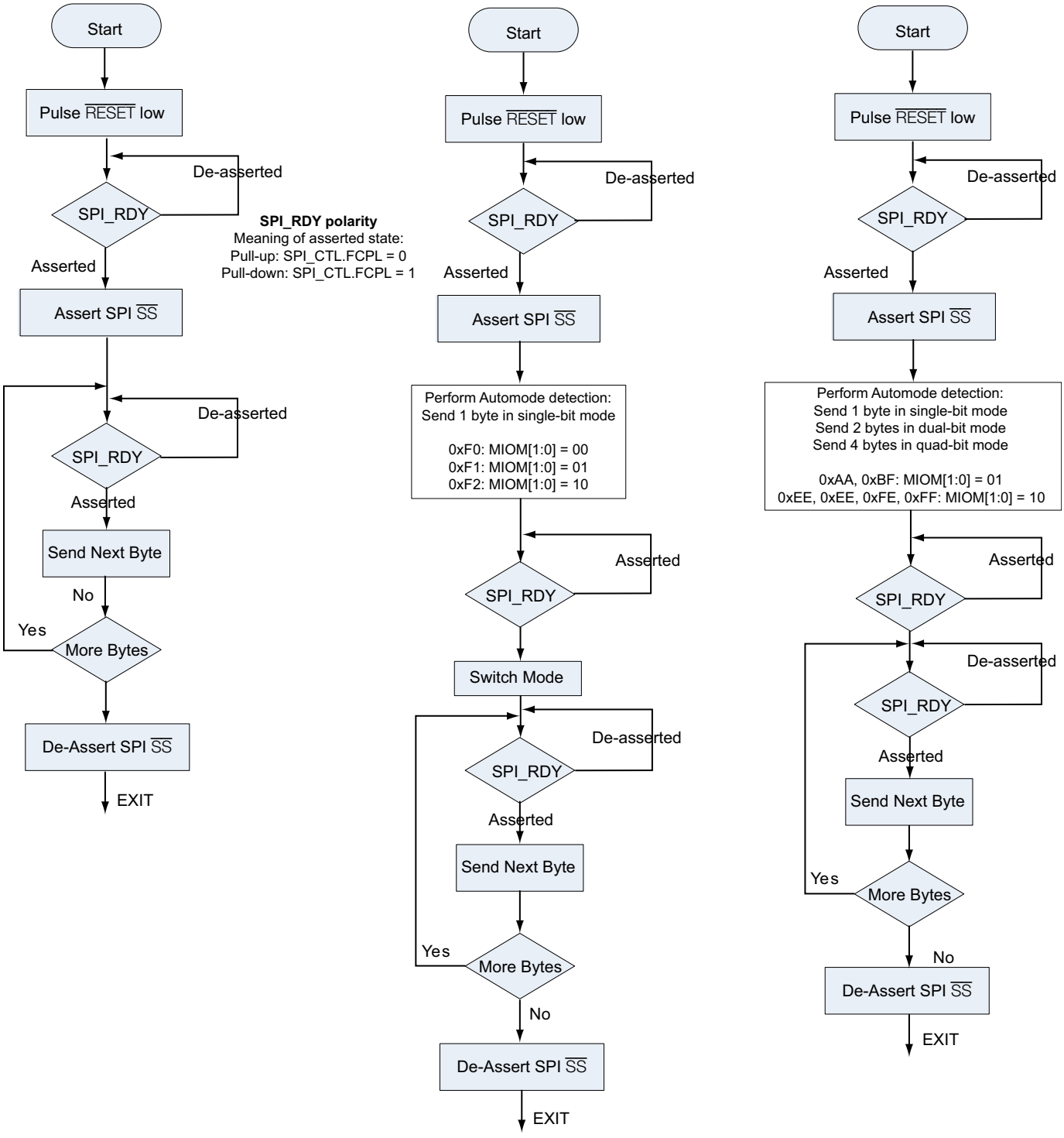


Figure 34-17: SPI Program Flow on the Host Side

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 additional customization such as disabling automatic device configuration or specifying a different SPI device other than SPI0.

When ROM_BCMD_NOCFG flag is specified, it is necessary to program pin muxing and other SPI configuration as required, while keeping the SPI_CTL.EN bit cleared.

The automode detection can be suppressed by the ROM_BCMD_NOAUTO switch. In that case, the desired configuration must be passed through the ROM_BCMD_SPI_CODE bit field, even if the ROM_BCMD_NOCFG flag is set.

Figure *SPI Slave Boot Mode* describes the fields possible for customization through the dBootCmd parameter.

dBootCommand: dBootCommand Parameter for SPI Slave Boot Mode - RW

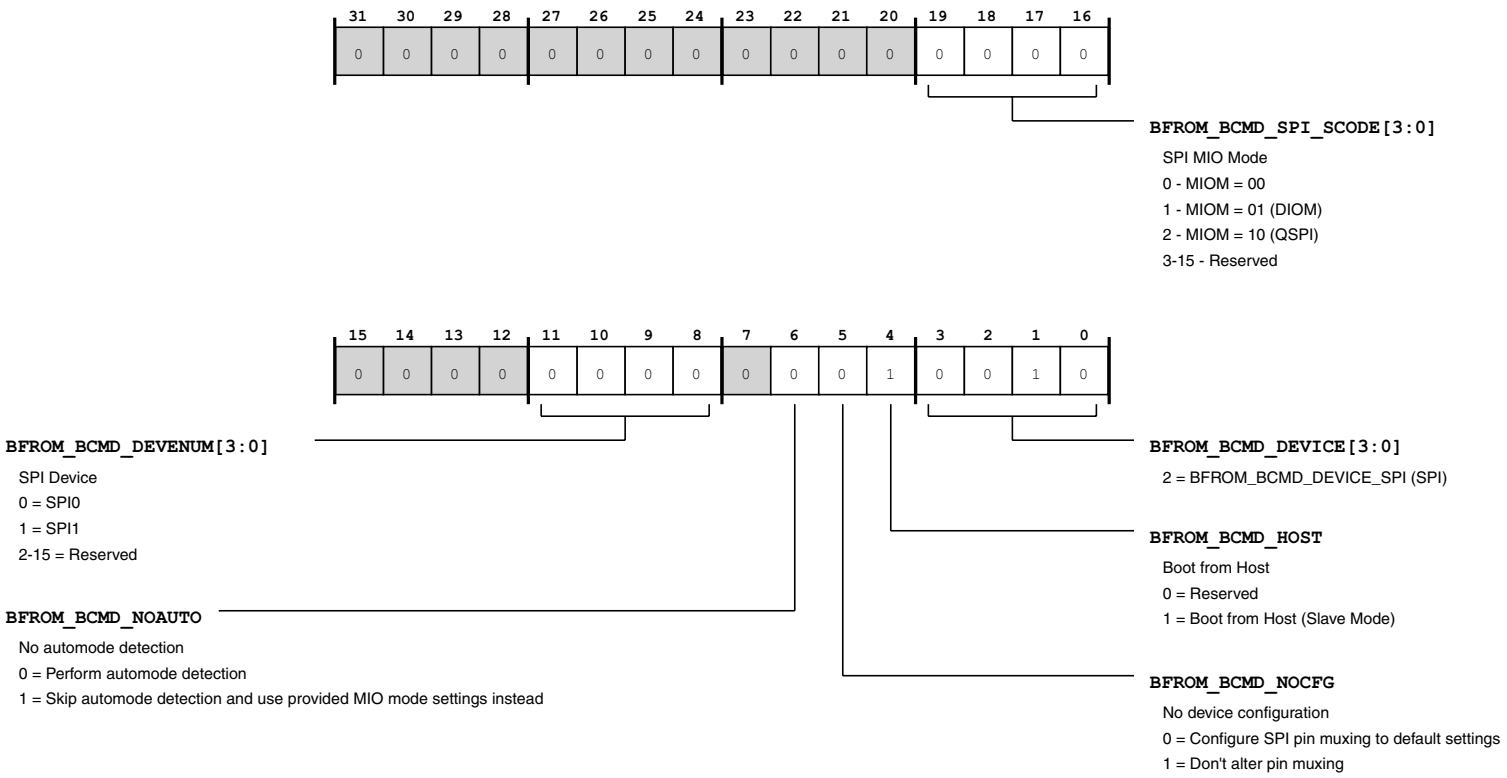


Figure 34-18: dBootCommand

Link Port Slave Boot Mode

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 66MHz 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 means of generating link port transmit and receive service requests. The transmit service request is generated on the processor to transmit the data by the receiver driving the LACKx signal high when the transmitter is disabled. The receive service request is generated on a receiver when it is disabled. This is initiated by the transmitter driving the LCLKx signal high.

As transmitter and receivers may be enabled at different times external pull-down resistors are required on both the LCLKx and LACKx signals in order 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. At no point prior to boot completion is the receiving link port disabled again. Once the link port is enabled all transfers are controlled by the receive DMA channel. The Load function for the link port Receive Boot mode may therefore simply point to the peripheral DMA routine of the main kernel in a similar way to the SPI slave boot mode.

Run-time API

The Boot Routine API function can be used to load a boot stream through the link port peripheral. Using this API the specific LP device may also be chosen, otherwise only link port0 is assumed. See the kernel API section for specifics on how to use this function.

dBootCommand: dBootCommand Parameter for Linkport Boot Mode - RW

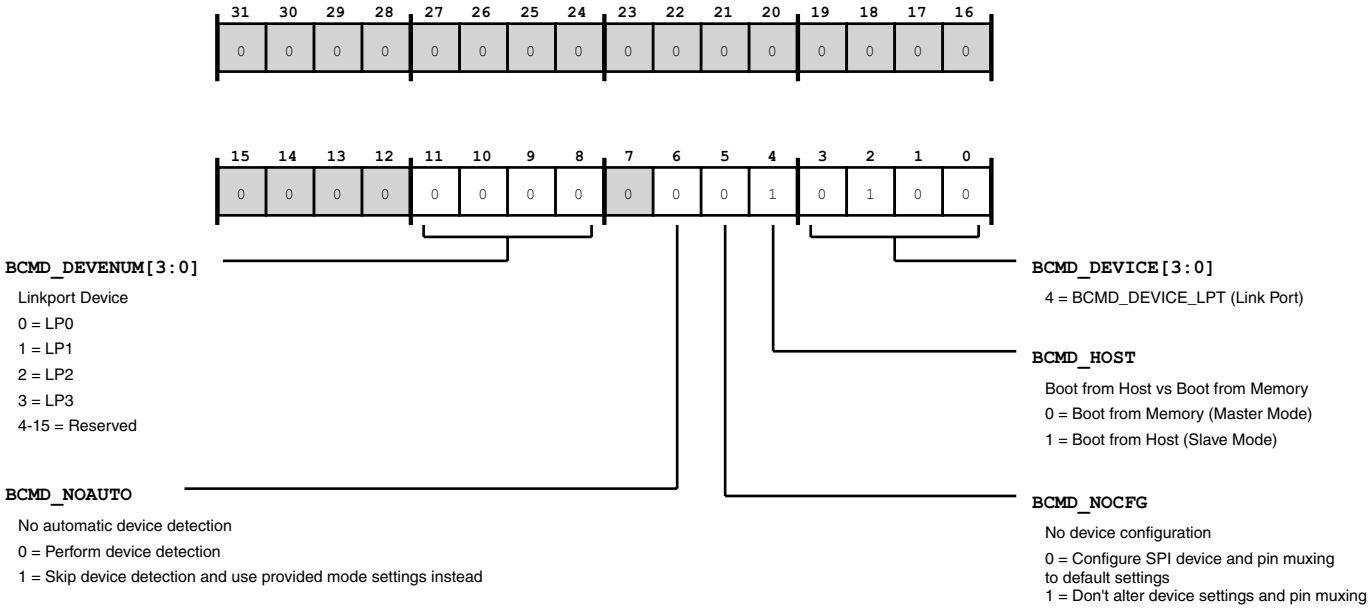


Figure 34-19: dBootCommand for Link Port Boot

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, can be done using the Boot Routine API.

During the 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 host. Since the RTS is in a high impedance state when the processor is in reset, or while executing a preboot, 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.

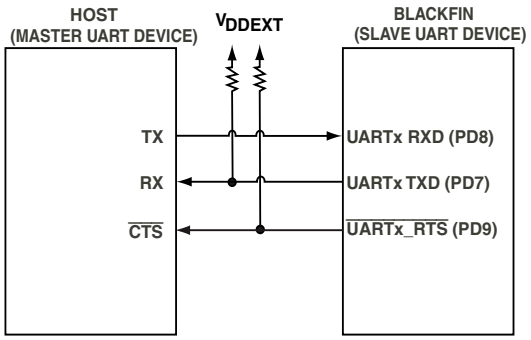


Figure 34-20: Connection Between Host and Processor

When the UART is enabled, the RTS goes immediately low, encouraging the host to send the first boot stream data as shown in the figure. In the case of half-duplex UART connections, this must be avoided. The host should wait until it has received the four bytes from the slave processor, before sending any data.

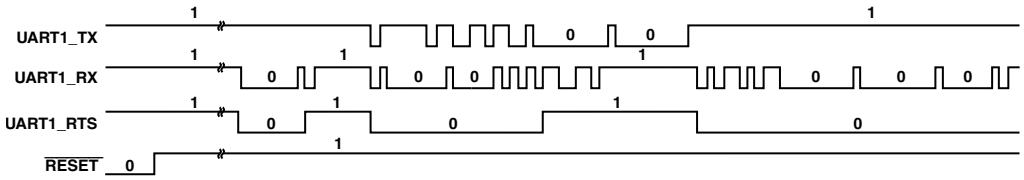


Figure 34-21: Host relying on RTS

When the boot kernel is processing fill or initcode blocks, it might require additional processing time and needs to hold the host off from sending more data. This is signaled using the RTS output.

The figure above shows RTS timing in case an extended initcode routine executes. Since code execution is distracting from the data loading, the host device has to be prevented from sending more data. The timing of the RTS depends on the state of the RFRT bit in the UART Control register (UART_CTL). This bit is cleared in case of the UART Slave Boot mode and RTS is deasserted when the UART receive FIFO contains 4 or more data words and another start bit is detected.

Autobaud Detection

The kernel supports autobaud detection using the '@' character as data. The host is expected to have it's clock set to rate supported in the UART

To determine the bit rate when performing the autobaud:

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 EDBO and UART_CLK register is cleared.
3. The boot kernel acknowledges, and the host then downloads the boot stream. The acknowledgment consists of four 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 has been received, the host can send the entire boot stream.

The host should know the total byte count of the boot stream, but it is not required to have any knowledge about the content of the boot stream.

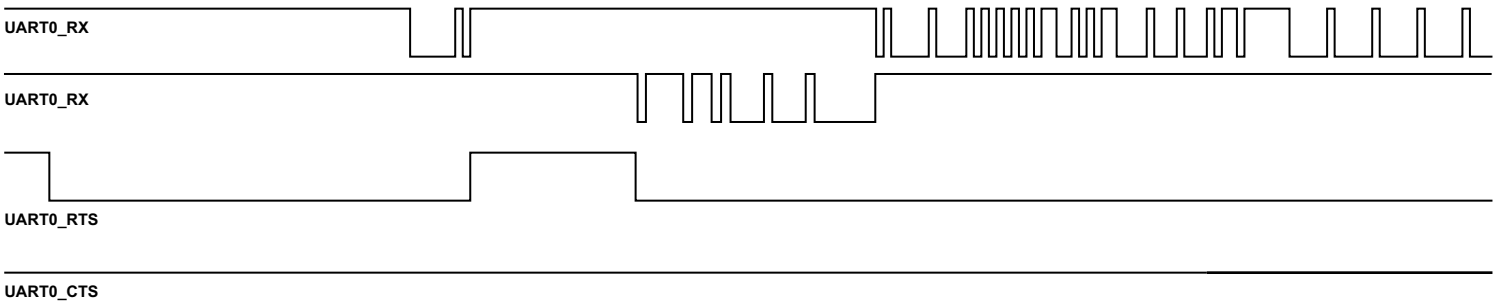


Figure 34-22: UART Autobaud Detection Waveform

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 four bytes.

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 additional customization. both autobaud detection and device configuration can be disabled, and a device other than the default, UART0, may be specified.

If `BFROM_BCMD_NOCFG` flag is specified, it is the programs responsibility to configure pin muxing as required.

Autobaud detection can be suppressed using the `BFROM_BCMD_NOAUTO` switch. In this case, the desired configuration can be passed through the `BFROM_BCMD_UART_CLK` bit field. If the `BFROM_BCMD_UART_CLK` bit field is zero, `UART_CLK` finally is evaluated. If the reset value is detected, 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 dBootCommand figure shows each of the available fields in the dBootCommand parameter in the Boot Routine API function.

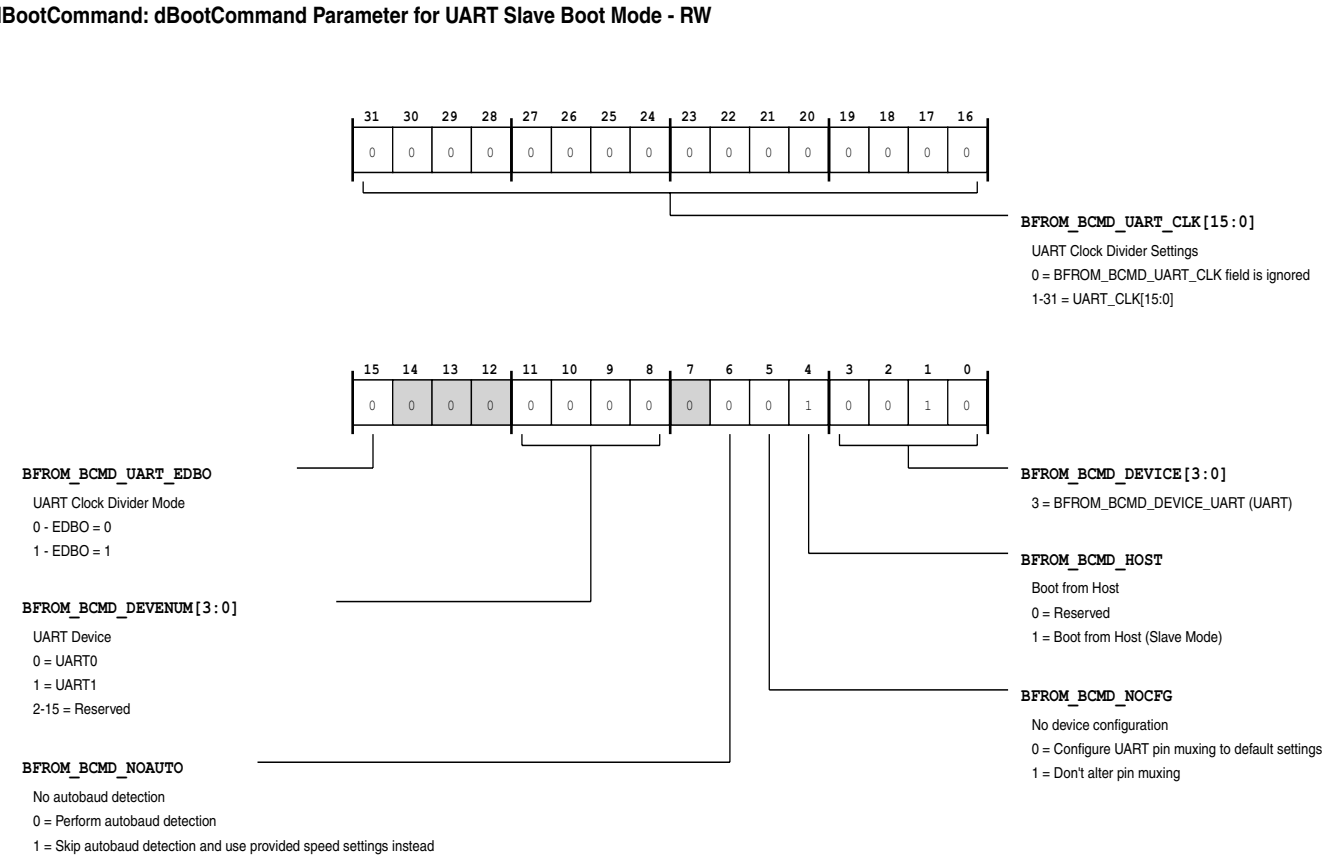


Figure 34-23: dBootCommand

Boot Programming Model

This section describes the programming model for booting the processor. The programming model includes booting functions, API calls, and data structures.

Each boot modefollows implements the same interface to the kernel. This consists of an Init function, a config function, load function, register function and a cleanup function. For an accurate description of the details of each boot mode's implementation, it is recommded to look at each function in the boot source that is provided.

Load Functions

All boot modes are processed by a common boot kernel algorithm. The load function point is exposed by the kernel to allow more complete customization of the booting process. The major customization is done by a subroutine that must be registered to the `pLoadFunction` pointer in the *STRUCT_ROM_BOOT_CONFIG*

structure. Its simple prototype is as follows.

```
void LoadFunction (                    STRUCT_ROM_BOOT_CONFIG* pBootStruct);
```

For some scenarios some of the flags in the dFlags word of the *STRUCT_ROM_BOOT_CONFIG* structure, such as BFLAG_PERIPHERAL and BFLAG_SLAVE, slightly modify the boot kernel algorithm.

The boot ROM contains several load functions. One performs a memory DMA for memory boot, another performs a peripheral DMA. The first is reused for fill operations and indirect booting as well.

In second-stage boot schemes, programs can create customized load functions or reuse the originals and modify the pDmaControlRegister , pControlRegister and nControlValue values in the *STRUCT_ROM_BOOT_CONFIG* structure. The pDmaControlRegister points to the DMAx_CONFIG or MDMA_Dx_CONFIG registers. When BFLAG_SLAVE flag is not set, the pControlRegister and dControlValue variables instruct the peripheral DMA routine to write the control value to the control register every time the DMA is started.

Custom load functions must meet the following requirements.

- Protect against dByteCount values of zero
- Multiple DMA work units are required if the dByteCount value is greater than 65536
- The pSource and pDestination pointers must be properly updated

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 BFLAG_IGNORE blocks. If the load function requires access to the block's ARGUMENT word, it should be read early in the function.

Page Mode

For the benefit of page oriented boot source devices , the boot kernel provides support for page operations. Page mode optimizes memory reads for block organized devices by always reading a page, rather than reading data on demand. The same temporary buffer used by the indirect blocks is used in page mode. The size of the buffer is defined by the dTempByteCount variable in the *STRUCT_ROM_BOOT_CONFIG* structure. The page size of the physical source device is defined by the dPageByteCount variable.

The pTempSourcevariable points to the source address of the data that is currently in the temp buffer. This variable is an internal variable of the boot kernel. However, at any time programs can set this variable to -1 to force the kernel to re fetch source data into the temp buffer. The following items also pertain to Page-mode.

- dPageByteCount must be a power-of-2 value (default is 4)
- dTempByteCount must be the same as or a multiple of dPageByteCount (default is 512)
- pTempBuffer does not have special alignment requirements. However, alignment of 32 may speed up DMA operations.

- In page mode, both block payload data and block headers are loaded through the same mechanism.

Changing Settings at Run Time

Programs can change the `pTempBuffer` , `dTempByteCount` , and `dPageByteCount` variables at any time, even within initcodes or callbacks. Whenever the settings change the kernel continues to operate on the old settings until the content of the former temporary buffer has been entirely processed. The new settings only become active for the next load operation. The kernel can be forced to immediately switch to the new settings by setting `pTempSource` to a value of -1. Note that doing so requires the kernel to re fetch data that had been loaded earlier. Under normal conditions `pTempSource`should not be altered.

The following initcode routine pseudo-code example illustrates on how page mode can be activated in any boot mode. Think of SPI master mode that has significant overhead when fetching data in little chunks, and for block headers or small boot blocks.

```
procedure initcode (BOOT_CONFIG config)
# The config input is the boot config datastructure
# change PLL and initialize DDR controller first */
initPLL()
# enable page mode operation
set config.dFlags.BITM_ROM_BFLAG_PAGEMODE to true;

# set to any unused DDR address,
# make sure it is not overwritten by the boot stream
# ideally pointing to an unitialized section, such as the stack or heap

 set config.pTempBuffer to 0xABCDABCD;
#update the temp byte count
# pSource points to the current source address
# pNextDxe points to the first address after the DXE
set config.dTempByteCount to config.pNextDxe - config.pSource
```

CRC32 Protection

This section describes the CRC32 Protection provisions

The boot kernel provides mechanisms to allow each block to be verified using a 32-bit CRC. To enable this feature, use the kernel initcode routine, `rom_Crc32Initcode` . An init block that provides this function can be created as the target address, and the ARGUMENT field containing the CRC32 checksum polynomial. Once this function is called, CRC verification is enabled for all blocks except forward, ignore, and first blocks. See the Boot Kernel API documentation for the specifics of this Initcode function.

Error Handler

This section describes how to customize the error handler

While the default handler puts the processor into idle mode, an initcode routine can register a customized error function by overwriting the error function pointer to create a customized error handler. The expected prototype is:

```
void ErrorFunction(                    STRUCT_ROM_BOOT_CONFIG                    *
pBootStruct, void *pFailingAddress);
```

Use an Initcode Routine (see *Block Types*) to write the entry address of the error routine to the `pErrorFunction` pointer in the STRUCT_ROM_BOOT_CONFIG structure. The error handler has access to the entire boot structure and receives the instruction address that triggered the error.

The default Error Routine performs the following actions in the following order:

1. Issues `EMUEXCPT` instruction
2. Issues `INTR_SOFT3` software interrupt to SEC
3. Toggles `FAULT` pin
4. Issues an IDLE instruction
5. While (CC) loop;
6. Returns (RTS)

Fault Management

Unless the `RCU_BCODE_NOFAULTS` or `RCU_BCODE_HALT` registers are set, the main routine enables the SEC and configures the following interrupts as faults early in the process:

CGU0_ERR, SEC0_ERR, WDOG0_EXP, DMAC_ERR, CRC0_ERR, SOFT3, C0_DBL_FAULT, C0_HW_ERR:= SCTL_SEN | SCTL_FEN;

For dual-core devices like the ADSP-BF60x family the Core 1 faults are also activated:

WDOG1_EXP, C0_BDL_FAULT, C0_HW_ERR= SCTL_SEN | SCTL_FEN;

After memory initialization, the memory protection channels are enabled as faults:

L2CTL0_ECC_ERR, INTR_C0_L1_PARITY_ERR and in the dual-core case INTR_C1_L1_PARITY_ERR

Events on any of these signals during the boot process will assert the FAULT and FAULTb output pins after a delay. A delay of 256 cycles is used to give the boot code opportunity to self detect the error situation and to execute an `emuexcpt` instruction to alert the emulator if connected.

`SEC0_FDLY` and `SEC_FSRDLY` are set to 0x100 value. SEC0_FCTL= FIEN | FOEN | EN;

Note that no interrupt is forwarded to either core.

If the boot code detects an error by software, its default error handler sets the INTR_SOFT3 software interrupt after having executed an EMUEXCPT and before executing an IDLE.

The boot code does not disable the SEC and fault settings on exit so that a safety hole is not introduced when transitioning to user application.

Callable API Overview

Describes the kernel API available at run-time

The boot code stored in ROM exposes several functions that can be used during run-time or within init-code or callback routines. This section describes the available functions and how they can be used. All functions meet the C runtime calling conventions described in the C/C++ Compiler and Libraries Manual. C prototypes are provided through `cdef_rom.h` header file, and addresses are provided by the `def_rom.h` header file.

System Control

The Systcontrol API provides a consistent reliable method to modify the PLL settings of the processor. It is recommended that this is the only method used to modify the PLL settings.

```
uint32_t rom_SysControl(uint32_t dActionFlags, STRUCT_ROM_SYSCTRL
*pSysCtrlSettings, void *reserved);
```

Functional Description

PP Define	FUNC_ROM_SYSCONTROL
Prototype	uint8_trom_SysControl(uint32_tdActionFlags, ADI_SYSCTRL_VALUES *pSysCtrlSettings, void *reserved);
Arguments	R0: Pointer to Syscontrol flags R1: Pointer to Syscontrol structure R2: reserved word
Return Value	R0: Syscontrol error flags
Stack Requirements	valid stack (SP, FP dividable by 4) required.

The Syscontrol API provide a normalized method to control setting in the CGU and DDR. The following three types of operations are provided:

- CGU Configuration
- DDR initialization from wakeup event
- CGU and DDR context save and restore operations from the `DPM_RESTOREn` registers in order to support entry to hibernate and wakeup from hibernate functionality

Syscontrol does not provide support for:

- Initial configuration of the DDR controller after a power up or reset sequence. The sequence of operations performed for the DDR. Configuration functionality is only compliant with the procedure required in order to restore the DDR configuration in the event of a hibernate/wakeup sequence while the DDR is in the self refresh state.
- Setting the DDR into self refresh mode. The DDR must be set to self refresh mode prior to calling syscontrol in order to save the DDR configuration in preparation of entering hibernate.

Syscontrol has not been designed in such a way that it performs all required operations in a single call. The syscontrol functionality can be broken down into two main categories:

- CGU configuration and limited DDR configuration
- CGU and DDR context save and restore from the DPM_RESTOREn registers

Syscontrol provides a means of configuring the CGU per user requirements. The CGU_CTL and CGU_DIVregisters are used to manipulate the various clock frequencies of the system. The *CGU Configuration Capacity* table defines the order of the sequences when writing to these two registers.

Table 34-20: CGU Configuration Capacity

CGU_CTL Frequency change	CGU_DIV Frequency change	CGU_DIV_UPDT	CGU_DIV_ALGN	Register Written
No	No	NA	NA	None
No	Yes	1	1	CGU_DIV
Yes	No	NA	NA	CGU_CTL
Yes	Yes	0	0	CGU_DIV then CGU_CTL

When the CGU_STAT*register indicates that the PLL is disabled then* CGU_DIV_ALGN *is never set.*

CGU and DDR context save and restore operations from the DPM_RESTOREn registers are enabled with the setting of ROM_SYSCTRL_WUA_EN in the *dActionFlags*parameter. When set any CGU or DDR configuration operations are ignored. Therefore any attempt to reconfigure the PLL while this bit is set will not be performed.

The ROM code calls syscontrol itself only when the device is waking up from hibernate. Nevertheless, syscontrol is an important primitive of the boot concept as programs are expected to call syscontrol from within their initcode to speed-up the boot process.

dActionFlags: Syscontrol Action Flags (R/W) - RW

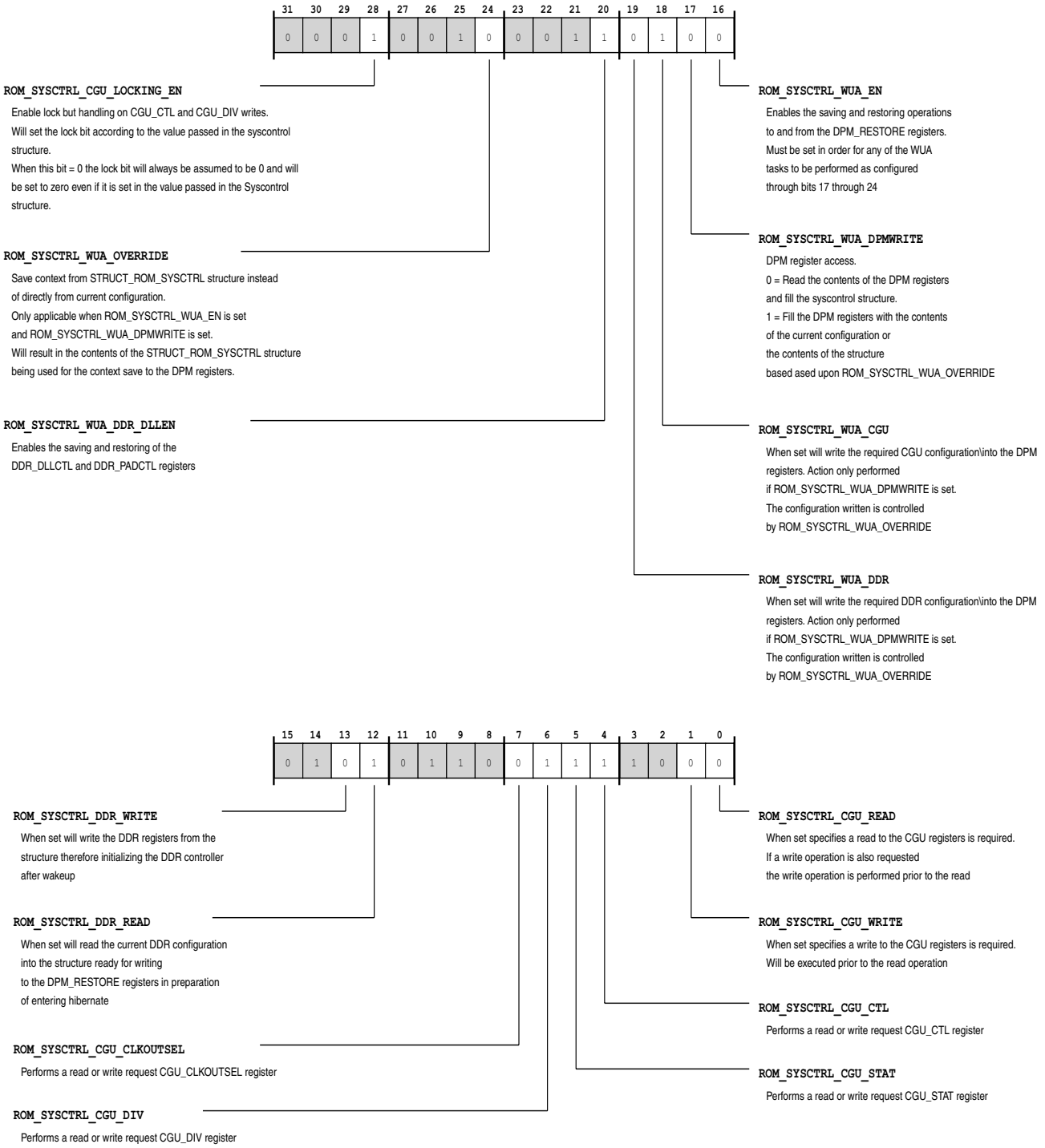


Figure 34-24: dActionFlags

Boot Kernel

The boot kernel API can be used to implement custom boot modes.

```
void * rom_BootKernel(STRUCT_ROM_BOOT_CONFIG *pBootStruct);
```

Boot Kernel

PP Define	FUNC_ROM_BOOTKERNEL
Prototype	void * rom_BootKernel(STRUCT_ROM_BOOT_CONFIG *pBootStruct);
Arguments	R0: Pointer to pBootStruct
Return Value	R0: pSource Pointer to next free source locations Pointer to DXE when invoked by ROM_BFLAG_NEXTDXE switch
Stack Requirements	valid stack (SP, FP dividable by 4) required

Boot Routine

The boot routine provides access to boot an application at run-time through a supported peripheral. The boot routine often also provides the ability for further customization over the equivalent boot mode.

```
rom_Boot(...)
```

The Boot Routine can be used for any kind of second-stage boot for supported boot modes. It provides options to boot from any device and any channel, whereas booting directly limits the choice to the default devices and channels. Often any auto-configuration or detection of the device can also be disabled. Each boot mode defines its down `dBootCommand`, (see individual boot modes for a description of this parameter). The *pCallHook argument is only needed when `BFLAG_HOOK` is set and should otherwise be `NULL`. The *pTargetAddress argument is only needed when `BFLAG_DATAREAD` is set and should otherwise be `NULL`.

PP Define	FUNC_ROM_BOOT
Prototype	void * rom_Boot(void *pBootStream, dFlags, `int32_t` dBlockCount, ROM_BOOT_HOOK_FUNC *pCallHook, `uint32_t` dBootCommand, void *pTargetAddress);
Arguments	R0: Pointer to Boot Stream (memboot) or Start Address of Boot Stream (SPI boot) R1: dFlags R2: dBlockCount [fp+ 0x14] ROM_BOOT_HOOK_FUNC *pCallHook [fp + 0x18] dBootCommand [fp + 0x1C] pTargetAddress
Return Value	R0: pSource Pointer to next free source locations Pointer to DXE when invoked by ROM_BFLAG_NEXTDXE switch

Stack Requirements	valid stack (SP, FP dividable by 4) required. 0x400 bytes of free stack suggested

CRC 32 Polynomial

Generates a CRC look-up table using CRC0.

```
bool rom_Crc32Poly(uint32_t ulPolynomial);
```

PP Define	FUNC_ROM_CRC32POLY
Prototype	bool rom_Crc32Poly(uint32_tulPolynomial);
Arguments	R0: Polynomial
Return Value	0: on success -1 on failure
Stack Requirements	none

CRC Initcode

The CRC Initcode is called by the bootstream. A respective init block is inserted by the loader utility when invoked by the `-crc` switch. The CRC Initcode extracts the polynomial from the `dArgument` field of each block header and calls then the CRC LUT function.

```
void rom_Crc32Initcode( STRUCT_ROM_BOOT_CONFIG *pBootStruct);
```

PP Define	FUNC_ROM_CRC32INITCODE
Prototype	void rom_Crc32Initcode(STRUCT_ROM_BOOT_CONFIG *pBootStruct);
Arguments	R0: Pointer to pBootStruct
Return Value	none
Stack Requirements	valid stack (SP, FP dividable by 4) required

ECC Protection

Protects 64-bit values by 8-bit ECC checksum using Hamming 7264 method. If return values equals ROM_ECC_ERROR_1BIT source data is corrected in place. Cycle count is independent of data and whether correction takes place or not.

```
uint8_t rom_Ecc(u64 *pData, uint8_t bChksum, bool bDecode);
```

PP Define	FUNC_ROM_ECC
Prototype	uint8_t rom_Ecc(u64 *pData, uint8_t bChksum, bool bDecode);

Arguments	R0: Pointer to 64-bit data R1: 8-bit checksum R2: 0=encode, 1=decode
Return Value	R0: 8-bit checksum when bDecode=false error code when dDecode=true
Stack Requirements	valid stack (SP, FP dividable by 4) required

Table 34-21: bDecode Values

PP Define	Description
ROM_ECC_ENCODE	encode mode
ROM_ECC_DECODE	decode mode
ROM_ECC_ERROR_INVS	invalid syndrome/checksum
ROM_ECC_ERROR_2BIT	uncorrectable 2-bit error
ROM_ECC_ERROR_1BIT	1-bt error, has been corrected
ROM_ECC_ERROR_NONE	no error

Execute

Resets the core as specified by dCoreId, stores dStartAddress into respective SVECTx registers and the releases the core.

Name	Execute	-
PP Define	FUNC_ROM_EXEC	-
Prototype	void rom_Exec(uint32_t dCoreId, void* dStartAddress, void *reserved);	-
Argument	R0	0=Core0, 1=Core1
Argument	R1	Pointer to Routine
Argument	R2	reserved, must be zero

Return Value	none	-
Stack Requirements	valid stack (SP, FP dividable by 4) required	-

```
rom_Exec(dCoreID,dStartAddress,0);
```

If dCoreId equals the current core,

- the routine simply issues a call to dStartAddress and returns.
- and the dStartAddress is 0x00000000 or 0xFFFFFFFF, the cycle counter is halted and an idle instruction is executed.

If dCoreId does not equal the current core,

- and the dStartAddress is 0x00000000, the target core is reset and kept in reset.
- and the dStartAddress is 0xFFFFFFFF, the target core vectors to ROM address where the cycle counter is halted and an idle instruction is executed.
- and the dStartAddress is 0xFFFFFFFE, the target core's SVECTx is used without modification

For safety reasons, the rom_Exec function does not unlock write access to the RCU0_SVECT registers. Unlocking and re locking is expected to be managed by the calling function.

PP Define	FUNC_ROM_EXEC
Prototype	void rom_Exec(uint32_t dCoreId, void* dStartAddress, void *reserved);
Arguments	R0: 0=Core0, 1=Core1, ... R1: Pointer to Routine R2: reserved, must be zero
Return Value	none
Stack Requirements	valid stack (SP, FP dividable by 4) required

Forward Config

The Forward Config routine is provided initialize the link port and SPI peripherals for using the boot forward feature.

```
uint32_t* rom_ForwardConfig(STRUCT_ROM_BOOT_CONFIG *pBootStruct,
uint32_t ulForwardPort);
```

The Forward Config routine initializes the link port and SPI peripherals for using the boot forward feature. This routine must be called from an initcode routine before any forward block can be processed. Refer to documentation on forward block (see *Block Types*) documentation for details on `pBootStruct` as the definition varies for each peripheral.

When called with the BFWD_CLEANUP flag, the Forward Config routine clears the `LPx_CTL` and `DMAy_CONFIG` registers.

`PP Define`	FUNC_ROM_FWDCFG
`Prototype`	`uint32_t* rom_ForwardConfig(STRUCT_ROM_BOOT_CONFIG *pBootStruct, uint32_t ulForwardPort);`
`Arguments`	`R0: Pointer to pBootStruct` `R1: Forward Port details`
`Return Value`	R0: pointer to Forward Callback Function on success0 on error
`Stack Requirements`	valid stack (SP, FP dividable by 4) required

Get Address

The Get Address routine can be used to access various look-up tables stored in the ROM. The function returns the address of the lookup table specified by the enum provided. Use this function rather than directly addressing tables to improve compatibility with future parts and silicon revisions.

Functional Description

`PP Define`	FUNC_ROM_GETADDR
`Prototype`	void * rom_GetAddr(enum ETABLES eTable);
`Arguments`	R0: eTable enumerator
`Return Value`	R0: -1 in case of eTable was undefined enum start address of table otherwise
`Stack Requirements`	none

Table 34-22: eTable enumerators Values

Index	Returned Address
0	Global constants
1	boot mode definition table
2	ecc syndrome table
3	MDMACODE look-up table
4	SPIMCODE look-up table
5	RSICODE look-up table

Mem Compare

Mem Compare compares a specified region of memory against a provide 32-bit reference value. The byte count can be any 32-bit value that is divisible by four, including zero. The DMA channel is disabled after completion, but the DMA status is not cleared so that in the event of an error the DMA can be debugged.

```
bool rom_MemCmp(void *pSrc, uint32_t ulChkVal, uint32_t ulByteCnt);
```

PP Define	FUNC_ROM_MEMCMP
Prototype	`bool rom_MemCmp(void *pSrc, uint32_t ulChkVal, uint32_t ulByteCnt);`
Arguments	R0: pSrc Source Address R1: Compare Value R2: Byte Count
Return Value	0: on success -1 on failure
Stack Requirements	none

Memory Copy

The Mem Copy function provides a convenient facility to copy memory using MDMA0 from one location to another. The byte count can be any 32-bit value including zero. The DMA channel is disabled after completion, but the DMA status is not cleared so that in the event of an error the DMA can be debugged.

```
bool rom_MemCpy(void *pDst, void *pSrc, uint32_t ulByteCnt);
```

PP Define	FUNC_ROM_MEMCPY
Prototype	bool rom_MemCpy(void *pDst, void *pSrc, `uint32_t` ulByteCnt);
Arguments	R0: pDst Destination Address R1: pSrc Source Address R2: Byte Count
Return Value	0: on success -1 on failure

Stack Requirements	none

Memory CRC

MemCRC scrubs memory using MDMA0 and CRC0 and builds a CRC checksum based on look-up table as generated by CRC32POLY beforehand. The byte count can be any 32-bit value that is divisible by four, including zero. The DMA channel is disabled after completion, but the DMA status is not cleared so that in the event of an error the DMA can be debugged.

```
bool rom_MemCrc(void *pSrc, uint32_t ulCrcChk, uint32_t ulByteCnt);
```

PP Define	FUNC_ROM_MEMCRC
Prototype	bool rom_MemCrc(void *pSrc, `uint32_t` ulCrcChk, `uint32_t` ulByteCnt);
Arguments	R0: pSrc Source Address R1: Reference Checksum R2: Byte Count
Return Value	0: on success -1 on failure
Stack Requirements	none

Memory Fill

Mem Fill fills a specified region of memory with a 32-bit value provided. The byte count can be any 32-bit value that is divisible by four, including zero. The DMA channel is disabled after completion, but the DMA status is not cleared so that in the event of an error the DMA can be debugged.

```
bool rom_MemFill(void *pDst, uint32_t ulFillVal, uint32_t ulByteCnt);
```

PP Define	FUNC_ROM_MEMFILL
Prototype	bool rom_MemFill(void *pDst, `uint32_t` ulFillVal, `uint32_t` ulByteCnt);
Arguments	R0: pDst DestinationAddress R1: Fill Value R2: Byte Count
Return Value	0: on success, -1 on failure
Stack Requirements	none

Software Built-in Self Test

Provides entry to multiple Software-Based Functional Self-Testing routines.

```
ADI_SBST_RESULT rom_SBST(TEST_FUNC_ID, argument1, argument2, argument3);
```

PP Define	FUNC_ROM_SBST

Address	0xC8000084
Prototype	ADI_SBST_RESULT rom_SBST(TEST_FUNC_ID, argument1, argument2, argument3);
Arguments	R0: ID of function to be executed R1: Argument 1 to be passed to function R2: Argument 2 to be passed to function [fp+ 0x14]: argument 3 to be passed to function
Return Value	struct ADI_SBST_RESULTR0: ADI_SBST_RESULT.dResultR1: ADI_SBST_RESULT.dErrorAddrress
Stack Requirements	valid stack (SP, FP dividable by 4) required

Booting Data Structures

The programming model for booting the processor uses the data structures defined in this section.

The programming model for booting the processor uses the data structures defined in this section.

STRUCT_ROM_SYSCTRL

Syscontrol configuration parameter

```
struct STRUCT_ROM_SYSCTRL;
```

uint32_t	ulCGU_CTL	Content of Clock Control Register
uint32_t	ulCGU_DIV	Content of Clock Divide Register
uint32_t	ulWUA_Flags	Flags used during wakeup
uint32_t	ulWUA_BootAddr	Memory boot address
uint32_t	ulWUA_User	User variable
uint32_t	ulDMC_CTL	Content of the DMC Control Register
uint32_t	ulDMC_CFG	Content of the DMC Config Register
uint32_t	ulDMC_TR0	Content of the DMC Timing Register 0
uint32_t	ulDMC_TR1	Content of the DMC Timing Register 1
uint32_t	ulDMC_TR2	Content of the DMC Timing Register 2
uint32_t	ulDMC_MR	Content of the DMC MR Shadow Register
uint32_t	ulDMC_EMR1	Content of the DMC EMR1 Shadow Register
uint32_t	ulDMC_EMR2	Content of the DMC EMR2 Shadow Register
uint32_t	ulDMC_PADCTL	Content of the DMC PADCTL register
uint32_t	ulDMC_DLLCTL	Content of the DMC DLLCTL register
uint32_t	ulReserved	Reserved entry

STRUCT_ROM_BOOT_BUFFER

```
struct STRUCT_ROM_BOOT_BUFFER
```

void	*pBuffer
int32_t	dByteCount

STRUCT_ROM_BOOT_CONFIG

```
struct STRUCT_ROM_BOOT_CONFIG
```

void	*pSource
void	*pDestination
uint32_t	*pControlRegister
uint32_t	*pAuxControlRegister
uint32_t	*pDmaControlRegister
uint32_t	*pSecControlRegister
int32_t	dControlValue
int32_t	dByteCount
int32_t	dFlags
uint16_t	uwDataWidth
uint16_t	uwSrcModifyMult
uint16_t	uwDstModifyMult
uint16_t	uwUserShort
int32_t	dUserLong
int32_t	dReserved
void	*pModeData
int32_t	dBootCommand
void	*pNextDxe
ROM_BOOT_ERROR_FUNC	*pErrorFunction
ROM_BOOT_LOAD_FUNC	*pLoadFunction
ROM_BOOT_CALLBACK_FUNC	*pCallBackFunction

ROM_BOOT_CALLBACK_FUNC	*pCrcFunction
ROM_BOOT_CALLBACK_FUNC	*pForwardFunction
STRUCT_ROM_BOOT_HEADER	*pHeader
void	*pTempBuffer
int32_t	dTempByteCount
void	*pTempSource
int32_t	dPageByteCount
uint32_t	ulBlockCount
uint32_t	ulBlockCurrent
int32_t	dClock
void	*pLogBuffer
void	*pLogCurrent
int32_t	dLogByteCount

STRUCT_ROM_BOOT_HEADER

```
typedef struct STRUCT_ROM_BOOT_HEADER
```

int32_t	dBlockCode
void	*pTargetAddress
int32_t	dByteCount
int32_t	dArgument

STRUCT_ROM_BOOT_SPI

```
struct STRUCT_ROM_BOOT_SPI
```

uint8_t	ubReadCommand
uint8_t	ubDummyBytes
uint8_t	ubAddressBytes
uint8_t	ubDataBits
uint16_t	uwClkLower
uint16_t	uwTxCtlUpper
uint16_t	uwRxCtlUpper
uint16_t	uReserved

Wakeup From Hibernate

When the boot code detects that the chip is waking up from hibernate it can take some special actions. These include:

- Configuration of PLL (CGU_DIV and CGU_CTL registers)
- Initialization of DDR controller

- Memboot from any memory address

The special sequence executes when the RCU_HBTRES hibernate reset identification bit is set. In case of a warm boot the RCU_HBTRES bit can be overruled by the BCODE_HBTOVW bit. When either of these bits is detected as a one, the boot code inspects the DPM_RESTORE0 register for further instructions. At least two bits must be set in DPM_RESTORE0 for the special action to take place.

Note the importance of this feature in case of slave boot modes. The Blackfin enters hibernate on its own decision. There is no direct path for this action to be directly controlled by the host (it might indirect thought). Therefore, do not assume that the host was aware of the processor being in the hibernate state.

On wakeup, the host may not know that the processor is expecting boot data. To prevent the processor from starving at wakeup, this special functionality enables the processor to boot from a LDR stream stored in off-chip memory (DDR or SMC SRAM). The stream may consist of one single 16-byte boot block header, which efficiently terminates the boot process and enables Core 0 to execute the code directly without further involvement of the boot code. Since even this one boot header is protected by XOR checksum the processor is prevented from booting illegal opcodes when the memory content was not in integer form.

When the WUA_MEMBOOT and WUA_ENA bits are set, after optional PLL and DDR initialization, the boot code does not continue with boot mode processing as per the BMODE pin configuration. Instead, the boot code performs a memory boot from the address specified by the WUA_BOOT_ADDR field. In case the address points to SMC space the memory boot mode automatically enables the respective SMC bank and required pin muxing.

CGU Initialization after Wakeup

When the processor enters the hibernate state, the CGU current configuration is lost and at a wakeup event the default reset values become applicable. In order to provide more optimal processor configuration at wakeup, programs may save the required CGU configuration to the DPM0_RESTORE registers. Upon detecting a wakeup event the boot code can optionally re-configure the CGU based on the contents of the DPM0_RESTORE registers.

The WUA_CLOCK bit of the DPM0_RESTORE0 register defines whether the boot code should reconfigure the CGU or not. If the CGU is to be reconfigured then the CGU0_DIV and the CGU0_CTL registers are programmed based upon the contents of two of the DPM0_RESTORE registers. See the registers section below for details of which registers are used during the configuration.

DDR Controller Initialization after Wakeup

Before entering the hibernate state, the external SDRAM is placed in self-refresh mode in order to preserve the contents of memory. During hibernate the power to the DDR is cut-off, to lower power consumption. When the device exits the hibernate state the DDR is required to be re-initialized. In order for the DDR to be re-initialized, programs are required to save the current DDR configuration to the DPM_RESTORE registers prior to the processor entering the hibernate state. The Syscontrol routine provides a means for programs to restore the context prior to entering hibernate in a manner that is compatible with the required context restoration.

BFLAG_WAKEUP and BFLAG_QUICKBOOT

Whenever either the RCU_BCODE_HBTOVW flag or the RCU_STAT_HBRST status bit is set, the main routine passes the BFLAG_WAKEUP flag when calling the Boot routine. When called that way the underlying boot kernel watches for block headers for the BFLAG_QUICKBOOT flag set. In the event of both flags being set, the boot kernel takes special action in the scope of the current block by:

- Toggling the BFLAG_IGNORE flag
- Clearing the BFLAG_INIT, BFLAG_CALLBACK, BFLAG_FORWARD, BFLAG_AUX and BFLAG_FINAL flags (as with any ignore block)
- Prevents fill blocks from processing (as with any ignore block)

Blocks that have BFLAG_IGNORE=0 and BFLAG_QUICKBOOT=1 are normally processed for a regular (non-wakeup) boot and are not processed for a wakeup boot. This way unnecessary rebooting of non-volatile memory cells can be avoided (speak external SRAM or SDRAM in self-refresh mode).

Blocks that have BFLAG_IGNORE=1 and BFLAG_QUICKBOOT=1 are ignored during regular boot and become active in case of wakeup. This supports complete arbitrate boot processing. Note that for FIRST blocks the IGNORE flag only controls the data load. The handling of the start vector and the next dxe pointer is not conditional. To make the initcode processing conditional, keep the BFLAG_QUICKBOOT flag clear for the init block and interrogate the BFLAG_WAKEUP flag in the initcode itself.

The BFLAG_SAVE flag is the BFLAG_QUICKBOOT's counterpart and enables programs to implement special context savings strategies before entering hibernate. However, the boot code does not directly support this functionality.

The wakeup/quickboot mechanism can be re-purposed by controlling the BFLAG_WAKEUP bit in the rom_Boot() routine's dFlags argument, or by altering the flag from within hook or initcode routines.

ADSP-BF60x DPM Register List

The dynamic power management (DPM) unit includes the phase locked loop (PLL) enable/disable features, deep sleep and hibernate mode controls, and clock domain enable/disable features. The combination of these features and controls provide selective and flexible power management. A set of registers govern DPM operations. For more information on DPM functionality, see the DPM register descriptions.

Table 34-23: ADSP-BF60x DPM Register List

Name	Description
DPM_CTL	Control Register
DPM_STAT	Status Register
DPM_CCBF_DIS	Core Clock Buffer Disable Register

Table 34-23: ADSP-BF60x DPM Register List (Continued)

Name	Description
DPM_CCBF_EN	Core Clock Buffer Enable Register
DPM_CCBF_STAT	Core Clock Buffer Status Register
DPM_CCBF_STAT_STKY	Core Clock Buffer Status Sticky Register
DPM_SCBF_DIS	System Clock Buffer Disable Register
DPM_WAKE_EN	Wakeup Enable Register
DPM_WAKE_POL	Wakeup Polarity Register
DPM_WAKE_STAT	Wakeup Status Register
DPM_HIB_DIS	Hibernate Disable Register
DPM_PGCNTR	Power Good Counter Register
DPM_RESTOREn	Restore Registers

DPM Restore

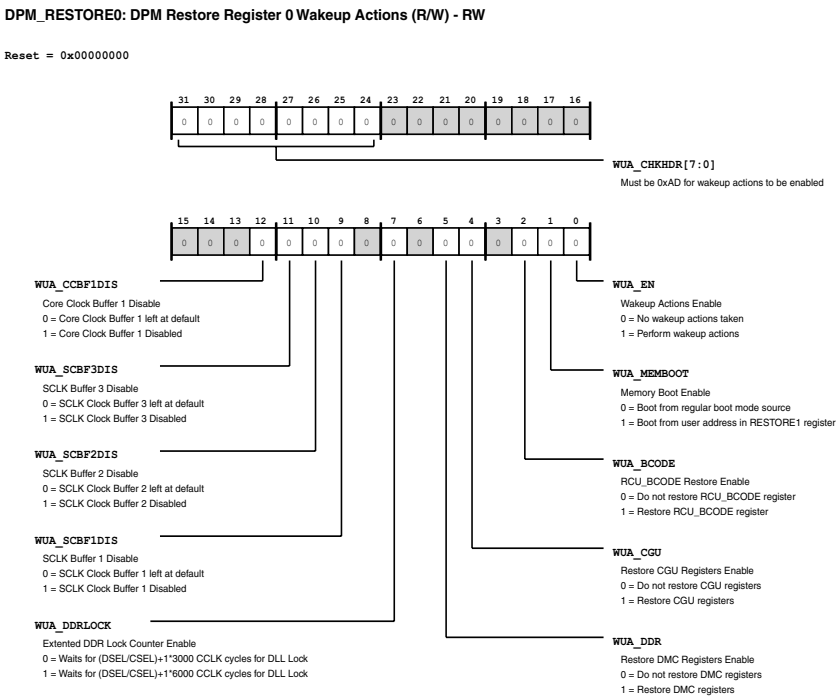


Figure 34-25: DPM Restore 0 Registers

The following table describes the function of each of the restore registers. RESTORE0 and RESTORE1 are only registers that do not directly restore other registers. RESTORE0 serves as the control register, enabling the function of the rest, and RESTORE1 stores a boot address rather than data to restore a particular register.

Table 34-24: DPM Restore Values

DPM Register	Restored Register	Enable Bit
RESTORE1	Address booted from	WUA_MEMBOOT
RESTORE2	CGU_DIV	WUA_CLOCK
RESTORE3	CGU_CTL	WUA_CLOCK
RESTORE4	RCU_BCODE	WUA_BCODE
RESTORE5[31:16]	DMC_CFG[15:0]	WUA_DDR
RESTORE5[15:0]	DMC_CTL[15:0]	WUA_DDR
RESTORE6[31:16]	DMC_MR[15:0]	WUA_DDR
RESTORE6[15:0]	DMC_EMR1[15:0]	WUA_DDR
RESTORE7	DMC_TR0	WUA_DDR
RESTORE8	DMC_TR1	WUA_DDR
RESTORE9	DMC_TR2	WUA_DDR
RESTORE10[31:16]	DMC_EMR2[15:0]	WUA_DDR
RESTORE10[15:0]	DMC_DLLCTL[15:0]	WUA_DDR
RESTORE11	DMC_PADCTL	WUA_DDR
RESTORE12	DMC_PHYCTL1	WUA_DDR
RESTORE13	DMC_PHYCTL3	WUA_DDR
RESTORE14	DMC_PHYCTL0	WUA_DDR
RESTORE15	DMC_PHYCTL2	WUA_DDR

Reset and Power-up

The processor implements several different power-up and reset scenarios ranging from a full power-on to software triggered resets. The Boot Kernel plays a specific role depending on the scenario.

Reset Vector

When reset releases, the processor starts fetching and executing instructions at the on-chip boot ROM.

On a hardware reset the boot kernel initializes the RCU_SVECTregister to 0xFFA00000. When the booting process completes, the boot kernel jumps the location provided by the RCU_SVECTvector register. With the

exception of the host DMA boot modes, the content of the RCU_SVECTregister is overwritten by the target address field of the first block of the loaded boot stream. If the BCODE field of the RCU_STATregister is set to No-boot, the RCU_SVECTregister is not modified by the boot kernel on software resets. Therefore, programs can control the reset vector for software resets through the RCU_SVECT register. The content of the register may be RCU_SVECT undefined in emulator sessions. For more information, see Software Vector Register 0 and Software Vector Register 1 .

Servicing Reset Interrupts

The reset interrupt has top priority and the processor services a reset event like other interrupts. Only emulation events have higher priority. When it comes out of reset, the processor is in supervisor mode and has full access to all system resources. The boot kernel can be seen as part of the reset service routine as it runs at top interrupt priority level.

Even when the boot process has finished and the boot kernel passes control to the user application, the processor is still in the reset interrupt. To enter the user mode, the reset service routine must initialize the RETI register and terminate by an RTI instruction.

Systems that do not work in an OS environment may not enter user mode. Typically, the interrupt level needs to be degraded down to IVG15.

NOTE: As the boot kernel is running at the reset interrupt priority, $\overline{\text{NMI}}$ events, hardware errors, and exceptions are not served at boot time. As soon as the reset service routine returns, the processor may service the events that occurred during the boot sequence. It is recommended that programs install $\overline{\text{NMI}}$, hardware error and exception handlers before leaving the reset service routine. This includes proper initialization of the respective event vector registers, RCU_SVECT

NMI and RESOUT

The $\overline{\text{NMI}}$ and $\overline{\text{RESOUT}}$ signals share the same pin. When the part is in reset, the pin is in $\overline{\text{RESOUT}}$ mode, and $\overline{\text{RESOUT}}$ is asserted. The $\overline{\text{NMI}}$ signal is disabled for the duration of this reset event. After the $\overline{\text{RESOUT}}$ goes low the pin is automatically placed into $\overline{\text{NMI}}$ mode and the $\overline{\text{NMI}}$ signal is enabled.

This pin should be connected to an external pull-up. The state of the RESOUT can be controlled by the RCU_CTL.RSTOUTDSRT and RCU_CTL.RSTOUTASRTcontrol bits.

Program Flow - Main Routine

On startup or reset the ROM's main routine is executed, the ROM is responsible for inspecting the state of the processor and taking appropriate action. This invovles inspecting the BMODE, BCODE, DPM_RESTORE resigisters, as well as what specifically caused the initiation of the booting process.

Core 0 begins the boot process, Core 1 is released during the early stages of the boot process and executes a default application as described. Core 1 can begin execution anytime during the application boot using an initcode (see *Block Types*). For more information, see *Execute*. If Core 1 will be used, it may be useful to set the ROM_BFLAG_NORESET flag.

The following sections give an overview of the procedures followed by both cores during a boot.

Core 0

1. Enable Cycle Counter and latch BMODE/ BCODE from RCU.
2. Set Lx, Bx, LCx registers to zero, setup stack registers, and initialize SVECT registers (including Core1 registers).
3. Release Core 1.
4. Install Fault Management, Initialize Memory and Release
5. Wakeup Management - potentially a direct memory boot depending on DPM wakeup registers.
6. Branch to address in SVECT0 if NoKernel option is set
7. Boot according to selected boot mode

Core 1

1. Enable Cycle Counter and latch BMODE/BCODE from RCU
2. Branch to address in SVECT1 register (normally set by Core 0)

Core 1 Default Application

After Core1 is released by Core0 during the initial stages of boot,Core1 executes a default application unless BCODE_NOVECTINIT flag is set. The entry address for the routine is written by Core0.

The default application sets SVECT1 to FF600000 (unless NOVECTINIT is set), Initializes its core's memory and become idle.

Memory Initialization

The first stage of booting initializes and validates all memory. This is performed before the actual boot process begins.

Memory Initialization occurs early in the boot process, before any stream information is read, and before the boot code has any influence. The boot kernel initializes all memory and cache tags and ensures all parity and ECC checksum bits are consistent with data. All memory is filled using a fill value.

Both cores are initialized. This infers that both cores will be woken up during this initial boot process. When booting from Core 0, upon completion of memory initialization, core 1 is left in the idle state. Core 1 can be used during the boot process using the API function.

The fill value for the processor is 0x00A700A7 or except 7

Boot Debug

This section describes some common techniques that can be applied for debugging the boot procedure.

If the boot process fails, very little information can be gained by watching the chip from outside. In master boot modes, the interface signals can be observed. In slave boot modes, only the RESOUT and RTS signals tell about the progress of the boot process.

The $\overline{\text{RESOUT}}$ signal is cleared after memory initializations are completed. The RESOUT and $\overline{\text{NMI}}$ signals share the same pin. During the boot process the $\overline{\text{NMI}}$ signal is disabled, once the RESOUT signal is asserted high upon completion of the boot process, the $\overline{\text{NMI}}$ signal is enable. Note that the RESOUT and $\overline{\text{NMI}}$ signals are active low and therefore require a pull-up resistor to be connected.

Using the emulator, however, there are many possibilities to debug the boot process. The entire source code of the boot kernel is provided with the tools installation. This includes the DSP executable (DXE) file. The *load symbols* feature of the tools environment helps to navigate the program. Note that the content of the ROM might differ between silicon revisions. Hardware breakpoints and single-stepping capabilities are also available. Since the content of the L1 instruction ROM cannot be read out by the emulator, as this ROM is not supported by the ITEST feature, these instructions are not displayed in the disassembly window.

The boot kernel also generates a circular log file in scratch pad memory. While the pLogBuffer and the dLogByteCount variables describe the location and dimension of the log buffer, the pLogCurrent points to the next free location in the buffer. The log file is updated whenever the kernel passes the _bootrom.bootkernel.breakpointlabel.

At each pass, nine 32-bit words are written to the log file, as follows.

- the block code word (dBlockCode) of the block header
- the target address (dTargetAddress) of the block header
- the byte count (dByteCount) of the block header
- the argument word (dArgument) of the block header
- the source pointer (dSource) of the boot stream
- the block count (dBlockCount)
- an internal copy of the dBlockCode word OR'ed with dFlags
- the content of the SEQSTAT register
- a 0xFFFFFFFA constant

The ninth word is overwritten by the next entry set, so that 0xFFFFFFFA always marks the last entry in the log file. Most of the data structures used by the boot kernel reside of the stack in scratch pad memory.

While executing the boot kernel routine (excluding subroutines) the P5 point to the *STRUCT_ROM_BOOT_CONFIG* structure. Type `(STRUCT_ROM_BOOT_CONFIG*) $P5` in the IDDE expression window to see the structure content.

The *Error Handler*can also give indications of errors. See the API documentation for more information.

Boot ROM Revision Control

Describes the provisions for reading ROM version on the processor

Boot ROM Revision Control

The boot ROM reserves the 32-bit location at `REG_ROM_REVISION` for a version code consisting of four bytes as shown in the *ROM Revision Control* .

Revision: Memory location with ROM revision information -

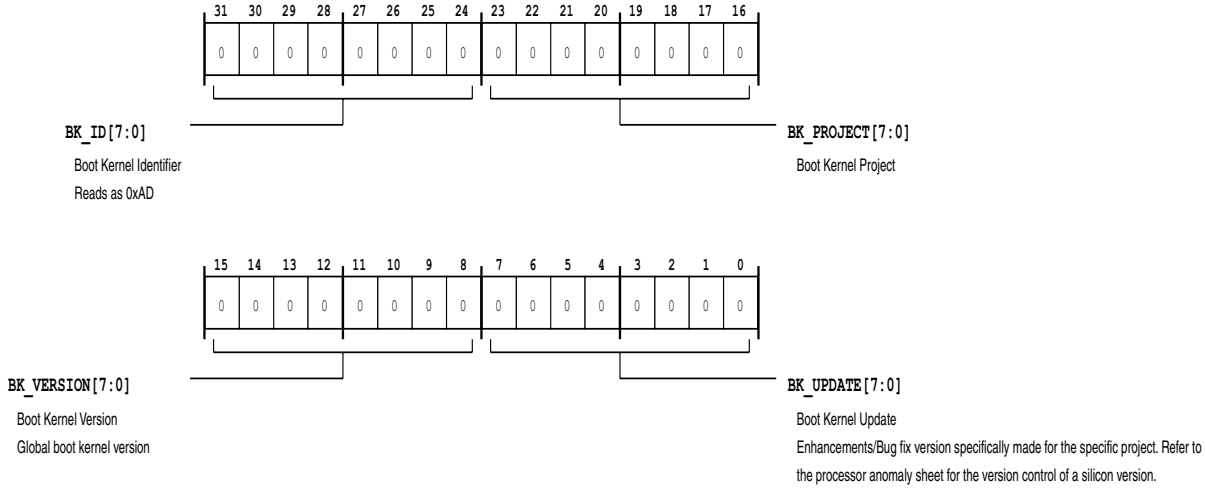


Figure 34-26: ROM Revision Control

Booting Register Reference

This section provides reference information regarding the registers used during the booting process.

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 cleared by software. Error status bits are cleared by any reset event.

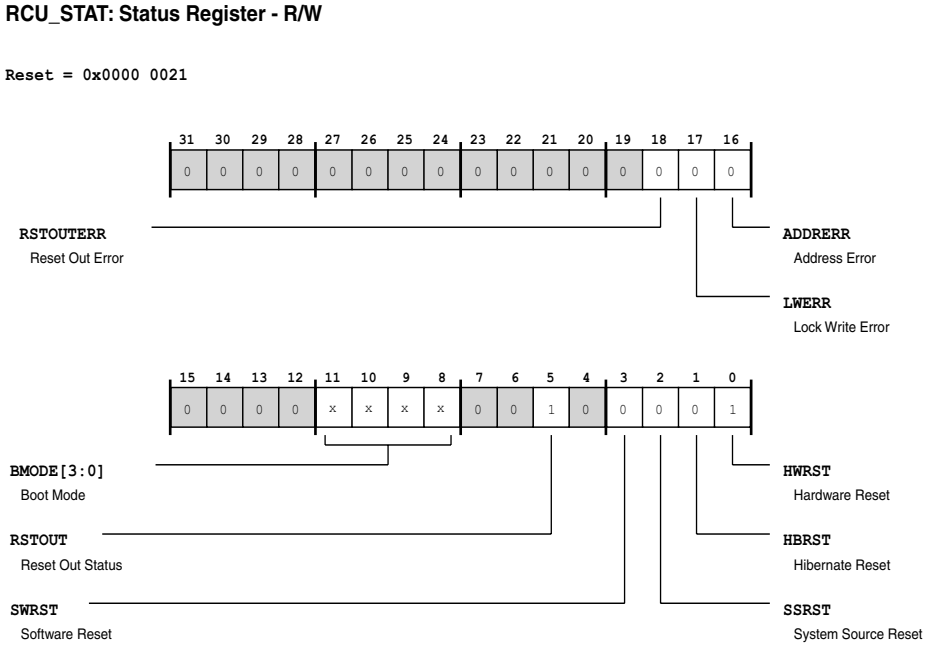


Figure 34-27: RCU_STAT Register Diagram

Table 34-25: RCU_STAT Register Fields

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

Table 34-25: 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
1 (R/W1C)	HBRST	Hibernate Reset. The RCU_STAT.HBRST bit indicates that a hibernate reset has occurred since the last time a hardware reset occurred or since the RCU_STAT.HWRST bit was cleared by software.	
		0	Inactive
		1	Reset Occurred

Table 34-25: RCU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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

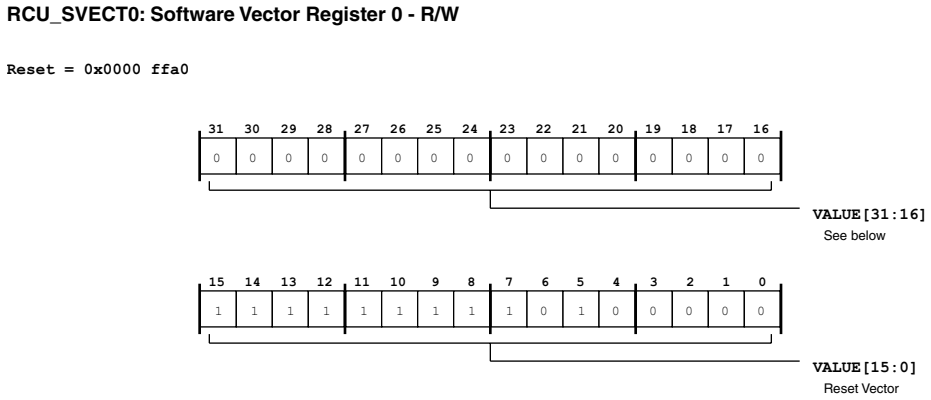


Figure 34-28: RCU_SVECT0 Register Diagram

Table 34-26: RCU_SVECT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Reset Vector.

Software Vector Register 1

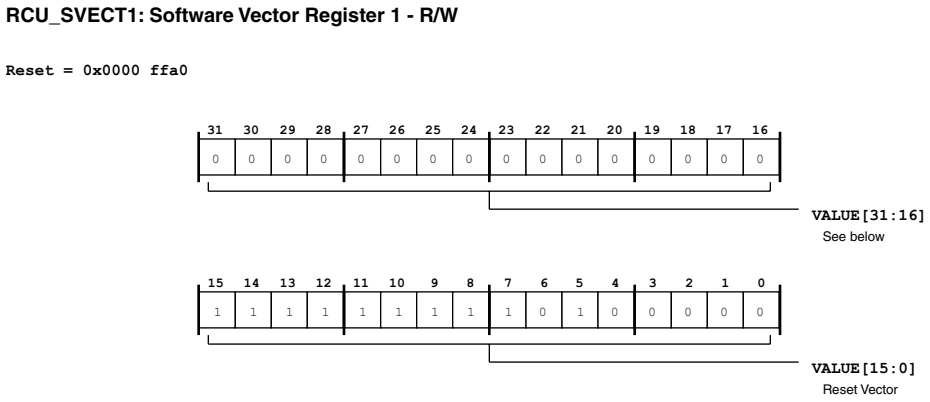


Figure 34-29: RCU_SVECT1 Register Diagram

Table 34-27: RCU_SVECT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Reset Vector.

Boot Code Register

The RCU software vector lock register (RCU_BCODE) provides a register lock and software vector n enable bits for each processor core on the product. For a processor-specific definition of the RCU_BCODE register, see the Booting Register Reference in the Boot ROM chapter.

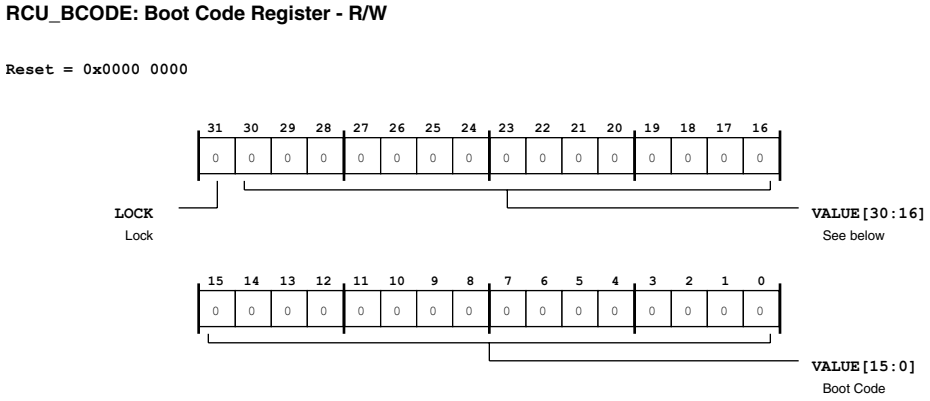


Figure 34-30: RCU_BCODE Register Diagram

Table 34-28: RCU_BCODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the RCU_BCODE.LOCK bit is set, the RCU_BCODE register is read only (locked).	
		0	Unlock
		1	Lock
30:0 (R/W)	VALUE	Boot Code. The RCU_BCODE.VALUE bits contain a boot code for the processor. For more information, see the RCU functional description.	

RCU BCODE Register Definition

The RCU BCODE register control a variety of options provided by the internal ROM.

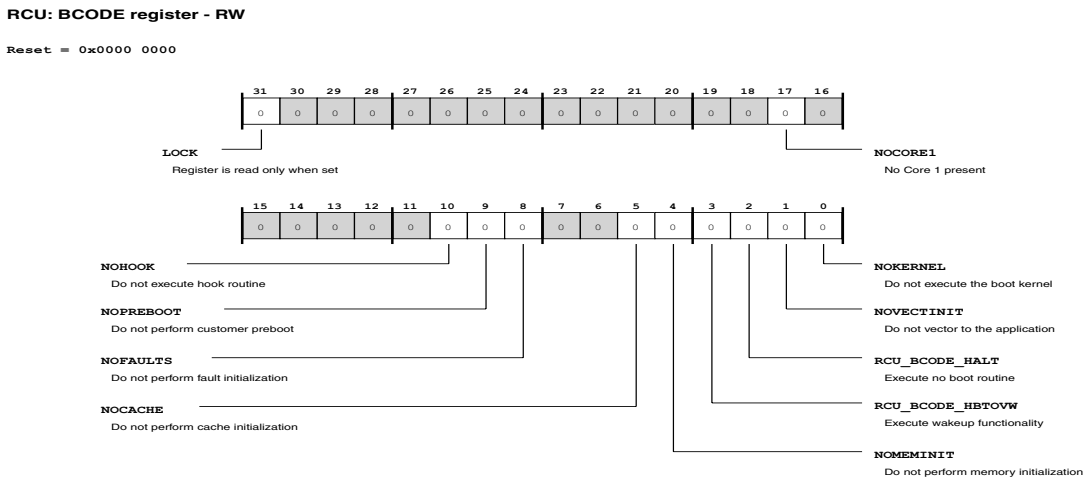


Figure 34-31: RCU BCODE Register Diagram

Table 34-29: RCU BCODE Register Fields

Bit No.	Bit Name	Description/Enumeration	
31	LOCK	Lock. If the global lock bit is set (SPU_CTL_GLCK bit =1) and the RCU_BCODE.LOCK bit is set, the RCU_BCODE register is read only (locked).	
		0	Unlock
		1	Lock
17	NOCORE1	No Core 1 present	

Table 34-29: RCU BCODE Register Fields (Continued)

Bit No.	Bit Name	Description/Enumeration
10	NOHOOK	Do not execute hook routine
9	NOPREBOOT	Always 0
8	NOFAULTS	Do not perform fault initialization
5	NOCACHE	Do not perform cache initialization
4	NOMEMINIT	Do not perform memory initialization
3	RCU_BCODE_ HBTOVW	Execute wakeup functionality
2	RCU_BCODE_ HALT	Execute no boot routine
1	NOVECTINIT	Do not vector to the application
0	NOKERNEL	Do not execute the boot kernel

35 System Debug Unit (SDU)

The System Debug Unit (SDU) provides debug access to the connected system. It provides debug host interface support through a JTAG (IEEE-1149.1) interface. In addition to traditional JTAG features, the SDU provides direct access to the processor's system resources to allow highly flexible and non intrusive debug support.

The SDU can control and communicate with processor cores through the JTAG or (if supported by the core) through system transactions. The SDU can perform system transactions as a system master through the memory access controller interface (MAC) and its memory-mapped registers can be accessed by system masters through the slave port.

SDU Features

- System JTAG TAP controller for system debug features, boundary scan, and public JTAG features
- Provides debug interface to core(s), and other system resources
- Provides direct and runtime access to memory system and system MMRs
- Provides DMA and non-DMA access to memory spaces
- SDU registers accessible to debug host and system (system MMRs)
- Support for system watchpoint and other event signaling
- Direct control over system reset
- Provides support for debug immediately after reset (boot debug)
- Supports group halt (debug event immediately halts all specified endpoints)

SDU Functional Description

The following sections provide a functional description of the SDU.

- *ADSP-BF60x SDU Register List*
- *ADSP-BF60x SDU Interrupt List*
- *ADSP-BF60x SDU Trigger List*
- *ADSP-BF60x SDU DMA List*

ADSP-BF60x SDU Register List

The system debug unit (SDU) provides IEEE-1149.1 support through its JTAG interface. The registers listed in the SDU register summary table govern SDU operations. For more information on SDU functionality, see the SDU register descriptions.

Table 35-1: ADSP-BF60x SDU Register List

Name	Description
SDU_IDCODE	ID Code Register
SDU_CTL	Control Register
SDU_STAT	Status Register
SDU_MACCTL	Memory Access Control Register
SDU_MACADDR	Memory Access Address Register
SDU_MACDATA	Memory Access Data Register
SDU_DMARD	DMA Read Data Register
SDU_DMAWD	DMA Write Data Register
SDU_MSG	Message Register
SDU_MSG_SET	Message Set Register
SDU_MSG_CLR	Message Clear Register
SDU_GHLT	Group Halt Register

ADSP-BF60x SDU Interrupt List

Table 35-2: ADSP-BF60x SDU Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
SDU0 DMA	64	11	LEVEL

ADSP-BF60x SDU Trigger List

Table 35-3: ADSP-BF60x SDU Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
SDU0 DMA	31	PULSE/EDGE

Table 35-4: ADSP-BF60x SDU Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
SDU0 DMA	31	
SDU0 Slave Trigger	69	

ADSP-BF60x SDU DMA List

Table 35-5: ADSP-BF60x SDU DMA List DMA Channel List

Description	DMA Channel
SDU0 DMA	DMA11

Definitions

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

JTAG 1149.1

IEEE Standard Test Access Port and Boundary-Scan Architecture.

Group Halt

Synchronous halt of distributed bus masters and various peripherals.

MAC

Memory Access Controller, system master interface controlled by the SDU

System Bus

In the context of the SDU definition this refers to the system interconnect fabric of the given product.

Block Diagram

The SDU architectural model is illustrated in the following figure. The figure is a block diagram of the SDU and the associated debug connections of the system.

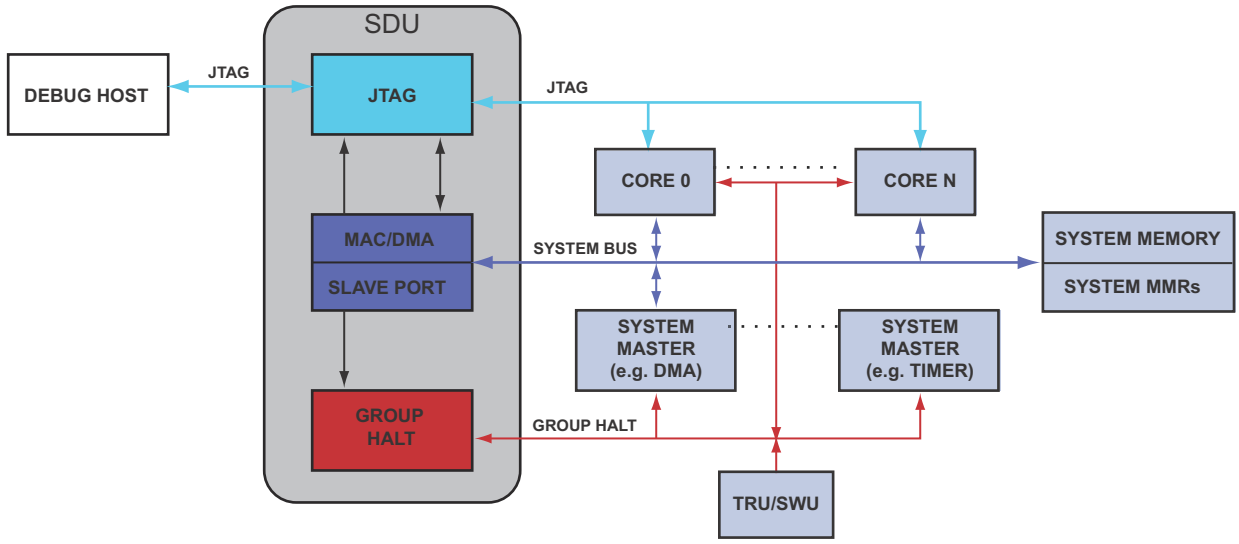


Figure 35-1: SDU Block Diagram

JTAG TAP Controller (JTC) Block Diagram

The SDU includes a JTAG test access port controller to support the IEEE-1149.1 standard. The JTAG TAP controller implements the traditional state machine for the TAP controller and the necessary support for the implemented scan chains. The following figure shows the JTAG (IEEE1149.1) TAP controller state machine.

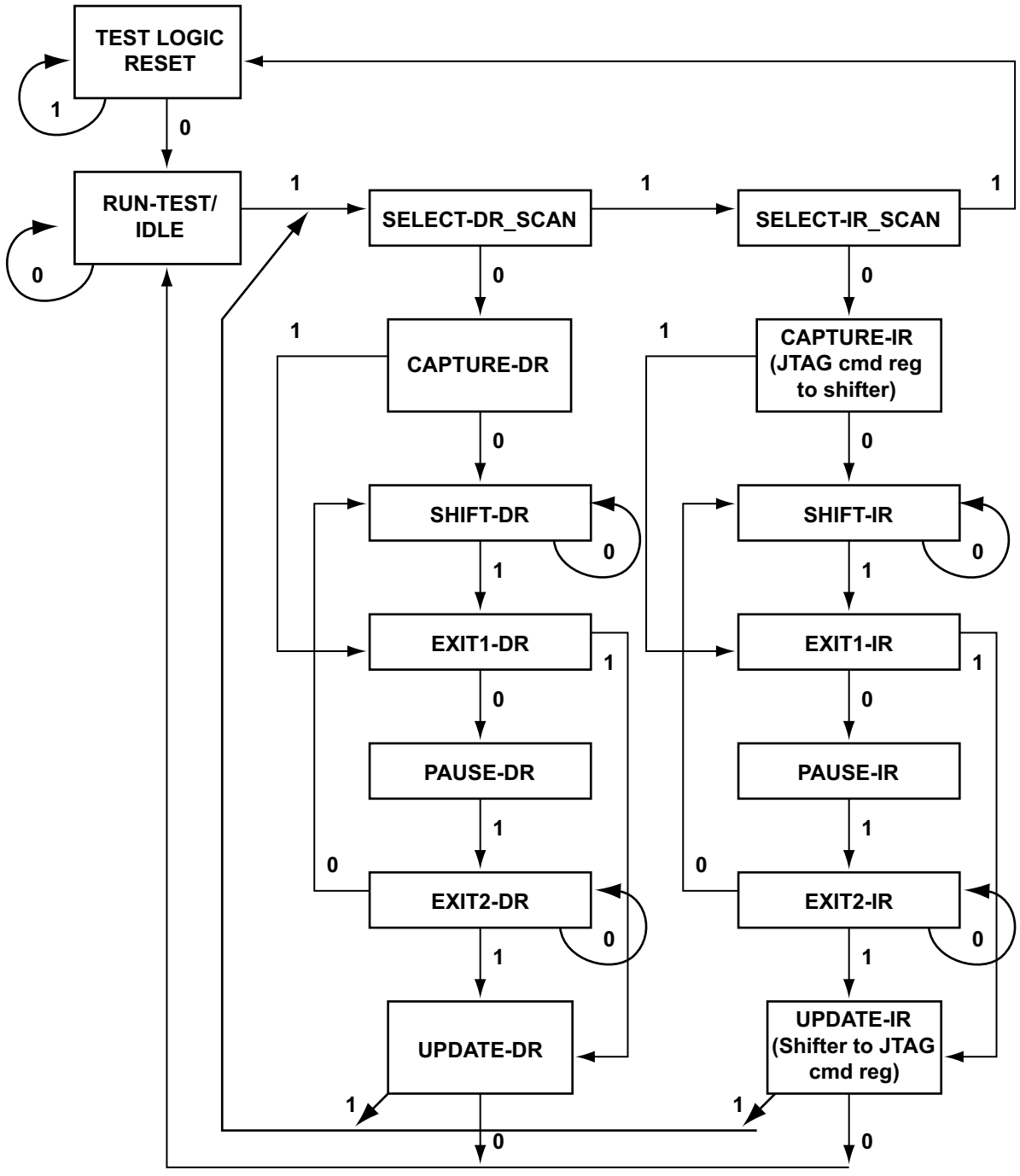


Figure 35-2: JTAG TAP Controller

JTC Core Emulation

Each core has its own JTAG TAP controller in addition to the JTC. The core JTAG TAP controllers are bypassed (not in the scan path) after reset. They can be added to the scan path under control of the SDU_

CTLregister. Each core JTAG controller provides support for the core only feature set (step, core register access and others). The following figure illustrates the connection model.

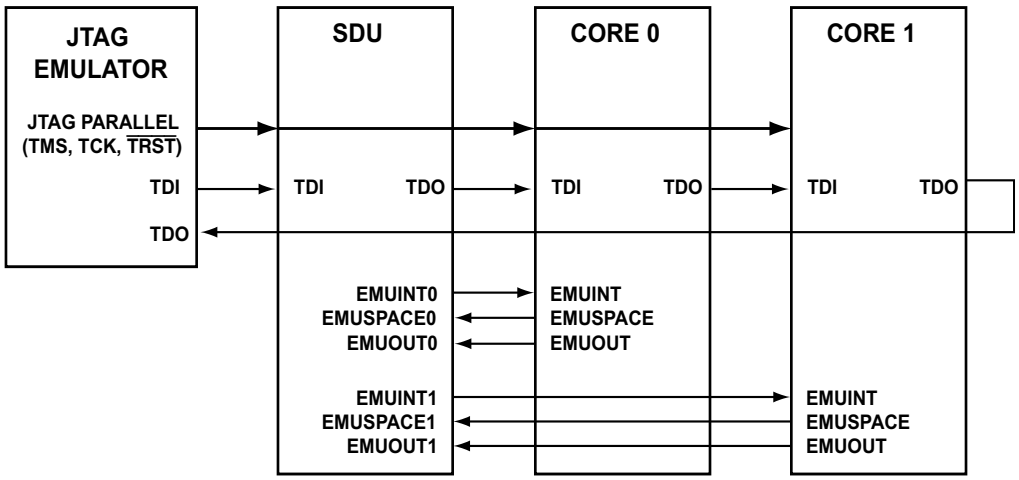


Figure 35-3: JTAG Signal Connections

JTC Instruction Register (IR)

The instruction register of the JTC is five bits wide. Decoding the instruction register selects which of the SDU scan registers is connected between the JTG_TDI and JTG_TDO pins during scan operations. The SDU scan registers provide support for JTAG public instructions and private (SDU) registers for debug control and status.

The following tables list the JTAG scan registers with their corresponding instruction decode. Instruction decode values not listed are reserved and must not be accessed.

Table 35-6: JTAG Public Scan Registers

Instruction Decode (TDI->TDO)	Instruction Names	Register Name	Width
00000	Extest	Boundary Scan	Product Specific
10000	Sample/Preload	Boundary Scan	Product Specific
00010	IDCODE	CHIPID - Constant (Product Specific) value	32
11111	Bypass	Bypass	1

Table 35-7: SDU Scan Registers

Instruction Decode (TDI->TDO)	Instruction Names	Register Name	Width
00101	SDU_CTL	SDU Control Register	16

Table 35-7: SDU Scan Registers (Continued)

Instruction Decode (TDI->TDO)	Instruction Names	Register Name	Width
10101	SDU_STAT	SDU Status Register	32
01101	SDU_MACCTL	Memory Access Control Register	5
11101	SDU_MACADDR	Memory Access Address Register	32
00011	SDU_MACDATA	Memory Access Data Register	32
10011	SDU_DMARD	DMA Read Data Register	32
01011	SDU_DMAWD	DMA Write Data Register	32

Memory Access Controller (MAC)

The MAC implements control functions to provide support for access to system memory space (for example L1, L2, L3 and System MMRs). These functions support debug host features like Background Telemetry Channel (BTC) and code and data download. The MAC supports direct and DMA accesses which are described in the following sections.

MAC Direct Access

The MAC provides direct access to system memory and system MMRs. This function provides access to individual addresses through address (`SDU_MACADDR`), data (`SDU_MACDATA`), and control (`SDU_MACCTL`) registers. When the debug host needs to perform a memory access, it loads these three registers with address, data, and control values. The act of writing to the control register triggers the MAC state machine to initiate a transfer request on the MAC system interface.

The `SDU_STAT.MACRDY`bit indicates when the MAC is ready for a new transfer. In the case of the previous operation being a read transfer, The `SDU_STAT.MACRDY`bit also indicates that read data has been returned in the `SDU_MACDATA`register.

MAC DMA Access

The MAC provides DMA access to system memory to improve the efficiency of debug host communication. The MAC DMA interface has the following elements.

- DMA FIFO
- DMA data registers

– DMA write data register (SDU_DMAWD)

– DMA read data register (SDU_DMARD)

The DMA FIFO is 4-bytes wide and connected to a system DMA channel on one side and to the SDU_DMARD and SDU_DMAWDregisters on the other.

The DMA interface registers act as the interface points to the DMA FIFO. Depending on the direction of a DMA transfer, the SDU_DMAWDregister is used to fill the DMA FIFO, or the SDU_DMARDregister is used to empty the DMA FIFO.

Group Halt

The SDU provides support for a user defined group halt model. The model allows for the masters and slaves of the group halt assertion to be selectable.

- Each master provides support for event signaling to the SDU for group halt assertion
- Each slave provides support for responding to a group halt assertion

The programming model consists primarily of a system MMR in the SDU group halt register (SDU_GHLT) to configure the masters and slaves of group halt. An overview of the signaling involved for group halt is shown in the following figure (example signal names are shown for illustration only).

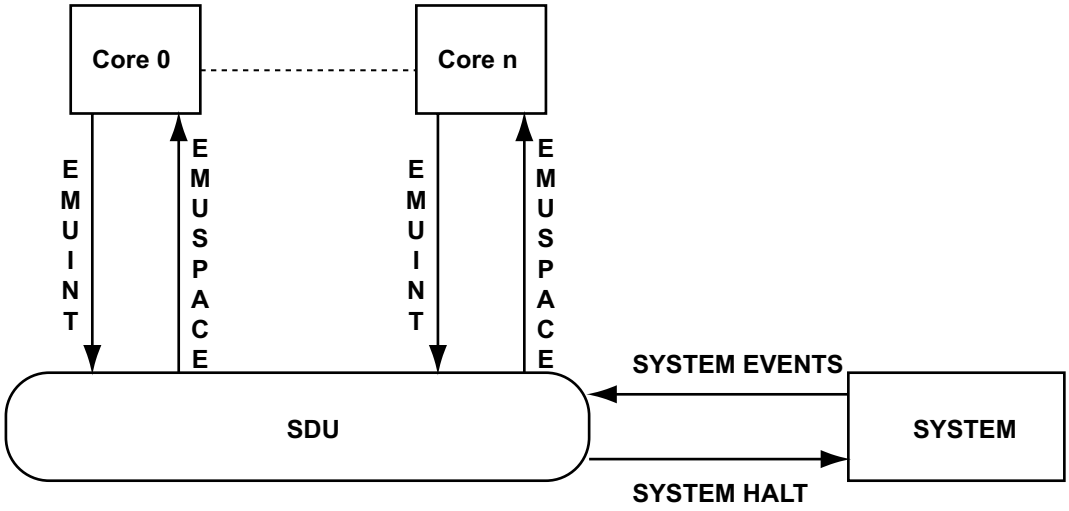


Figure 35-4: Group Halt Overview

Group Halt Status

Group halt status and group halt cause (master) are captured in the SDU_STAT register. The cause is captured when the status bit is set. This information is used to indicate to the debug host that the group halt was asserted and which master generated the group halt.

NOTE: The first cause of the halt is the only one that is captured.

The system halt signal (from the **Group Halt Overview** figure) is the indication for non-core system masters to halt if configured to do so. The SDU_STAT.GHLT bit is the source of the system halt if the system is selected as a slave. For the system to resume (deassertion of system halt), SDU_STAT.GHLT bit must be cleared by a W1C operation.

Group Halt Masters

If *core n* generates a debug event (for example software breakpoint) it asserts its debug mode indication signal (for example EMUSPACE) to the SDU. If core n is enabled as a master of group halt, the SDU sets the SDU_STAT.GHLTbit and capture the cause in the SDU_STAT.GHLTCbit.

If the system generates a debug event (for example SDU trigger input, system watchpoint event) it asserts the corresponding signal to the SDU. If the system is enabled as a master of group halt, the SDU sets SDU_STAT.GHLTbit and capture the cause in the SDU_STAT.GHLTCbit.

System debug events may be disabled at the source as necessary (for example a system watchpoint event indication to the SDU may be disabled in the System Watchpoint Unit).

Group Halt Slaves

When the SDU_STAT.GHLT bit is set, the SDU asserts the debug request signal (for example EMUINT) to selected cores according to the settings in the SDU_GHLT register. The debug request to a core is asserted based on the transition of SDU_STAT.GHLT from 0 to 1).

When the SDU_STAT.GHLT bit is set, the SDU asserts the system halt signal if system is selected as a slave in the SDU_GHLT register.

Each system master has a dedicated control bit to specify its response to a system halt signal assertion. Typically this response is to halt the system which may have the following local implications.

1. Stop issuing system transactions
2. Pause the entire block
3. Both

SDU Programming Concepts

The following sections provide information on debug programming concepts.

Core Control

Cores are directly controlled through emulation space. Core emulation space is entered by generating the core emulation interrupt. To do this, emulation and the emulation interrupt must be enabled in the appropriate core control register.

The SDU generates an emulator interrupt with the rising edge of the JTG_TMS pin when the JTAG TAP controller in the SDU is in the Run-Test-Idle state. The processor core(s) enter emulation space in response to the emulation interrupt from SDU. The SDU generates an emulation interrupt to a core if the specific core is selected using the SDU_CTL.EHLTbit field.

Once in emulation space, the core executes operations as directed by the debug host. Processor instructions are scanned in and executed using the core emulation instruction register (for example EMUIR) and the core emulation data register (for example EMUDAT). The JTAG TAP controller is used to perform the scan operations.

Memory and Register Access

Memory and register access is performed using one of the following two methods.

- **Direct.**Memory and registers that are mapped into the system address space may be accessed directly by the SDU system interfaces (for example the MAC).
- **Indirect.**Memory and registers that are not mapped into the system address space may only be accessed indirectly through core instruction execution while in emulation space (see *Core Control*).

Statistical Profiling

Cores may support runtime statistical profiling by providing a register that samples the core PC in a pseudo-random fashion. The debug host accesses the core register (for example EMUPC) through JTAG scan operations.

Power Management Support

A processor may support various power saving modes which can limit the ability of a debug host to communicate with the processor resources because the debug host may not be able to detect when the processor enters or exits such modes. The SDU provides the following support for the debug host to determine the power mode.

- **Core Clock(s) Stopped.** Power management may support a mode (for example sleep) which stops the core clock or clocks. In this mode the SDU is unable to communicate with some system resources (for example core registers or core memory). The power management resource (for example Dynamic

Power Management or DPM) typically remains accessible. The debug host, through the SDU MAC, may access the status registers in the DPM to determine the state of the processor.

- **System Clock(s) Stopped.** Power management may support a mode (for example deep sleep) which stops the system clock or clocks. In this mode the SDU is unable to communicate with all system resources including the power management resource (for example dynamic power management). The SDU provides an input and associated status bit for the DPM to provide an indication of this power mode. The debug host may scan the SDU status register (`SDU_STAT`) through the JTAG.

Security Support

A processor may support various security modes which can limit debug host communications with processor resources in that the debug host may not detect when the processor enters or exits such security modes.

When security is enabled, the SDU provides an input and an associated status bit to provide indication of a secure mode. The debug host may scan the SDU status register (`SDU_STAT`) through the JTAG.

System Reset Support

The SDU may indirectly generate a system reset. When set, the system reset bit (`SDU_CTL.SYSRST`) signals a system reset request to the Reset Control Unit (RCU). In response, the RCU asserts the system reset signal. Other events in the system can also cause the RCU to issue a system reset. In response to the system reset signal from RCU, all registers in the SDU are reset, except for the `SDU_CTL`register. See the *Register Descriptions* section for more detail.

ADSP-BF60x SDU Register Descriptions

System Debug Unit (SDU) contains the following registers.

Table 35-8: ADSP-BF60x SDU Register List

Name	Description
`SDU_IDCODE`	ID Code Register
`SDU_CTL`	Control Register
`SDU_STAT`	Status Register
`SDU_MACCTL`	Memory Access Control Register
`SDU_MACADDR`	Memory Access Address Register

Table 35-8: ADSP-BF60x SDU Register List (Continued)

Name	Description
SDU_MACDATA	Memory Access Data Register
SDU_DMARD	DMA Read Data Register
SDU_DMAWD	DMA Write Data Register
SDU_MSG	Message Register
SDU_MSG_SET	Message Set Register
SDU_MSG_CLR	Message Clear Register
SDU_GHLT	Group Halt Register

ID Code Register

The SDU ID code register (SDU_IDCODE) contains bit fields describing the processor silicon revision, product identification, and manufacturer identification.

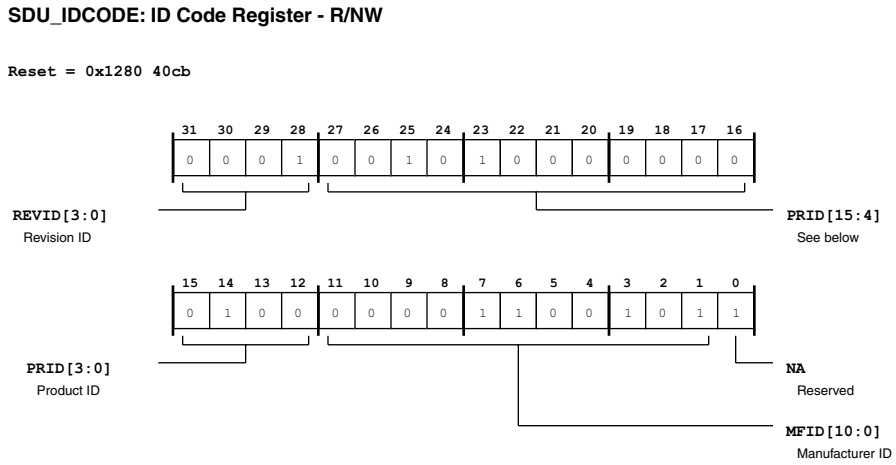


Figure 35-5: SDU_IDCODE Register Diagram

Table 35-9: SDU_IDCODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:28 (R/NW)	REVID	Revision ID.

Table 35-9: SDU_IDCODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
27:12 (R/NW)	PRID	Product ID.
11:1 (R/NW)	MFID	Manufacturer ID.
0 (R/NW)	NA	Reserved.

Control Register

The SDU control register (SDU_CTL) contains bit fields enabling or disabling emulator features. This register may be reset only by asserting the $\overline{\text{JTG\_TRST}}$ pin.

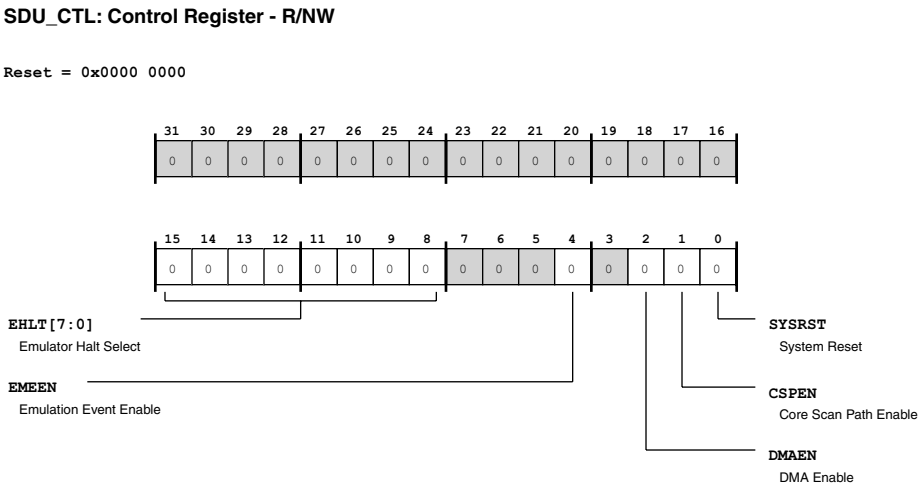


Figure 35-6: SDU_CTL Register Diagram

Table 35-10: SDU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	EHLT	Emulator Halt Select. The SDU_CTL.EHLT selects system components to halt on assertion of the SDU_STAT.EME bit.

Table 35-10: SDU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	EMEEN	Emulation Event Enable. The SDU_CTL.EMEEN enables the assertion of the SDU_STAT.EME bit on the rising edge of the JTG_TMS pin input.	
		0	Disable
		1	Enable
2 (R/W)	DMAEN	DMA Enable. The SDU_CTL.DMAEN enables the operation of the SDU_DMARD and SDU_DMAWD registers. When the SDU_CTL.DMAEN bit is cleared, the SDU_DMARD and SDU_DMAWD registers are disabled, and the SDU_STAT.DMARDRDY and SDU_STAT.DMAWDRDY bit (associated status bits) are cleared.	
		0	Disable
		1	Enable
1 (R/W)	CSPEN	Core Scan Path Enable. The SDU_CTL.CSPEN enables the JTAG TAP controllers, adding them to the JTAG scan path.	
		0	Disable Signals the (RCU) module to assert system reset, resetting all resources in the SDU SCLK domain except for the SDU_CTL register.
		1	Enable
0 (R/W)	SYSRST	System Reset. The SDU_CTL.SYSRST forces a hardware reset when set.	
		0	Inactive
		1	Active

Status Register

The SDU status register (SDU_STAT) contains bit fields describing debug-related processor status.

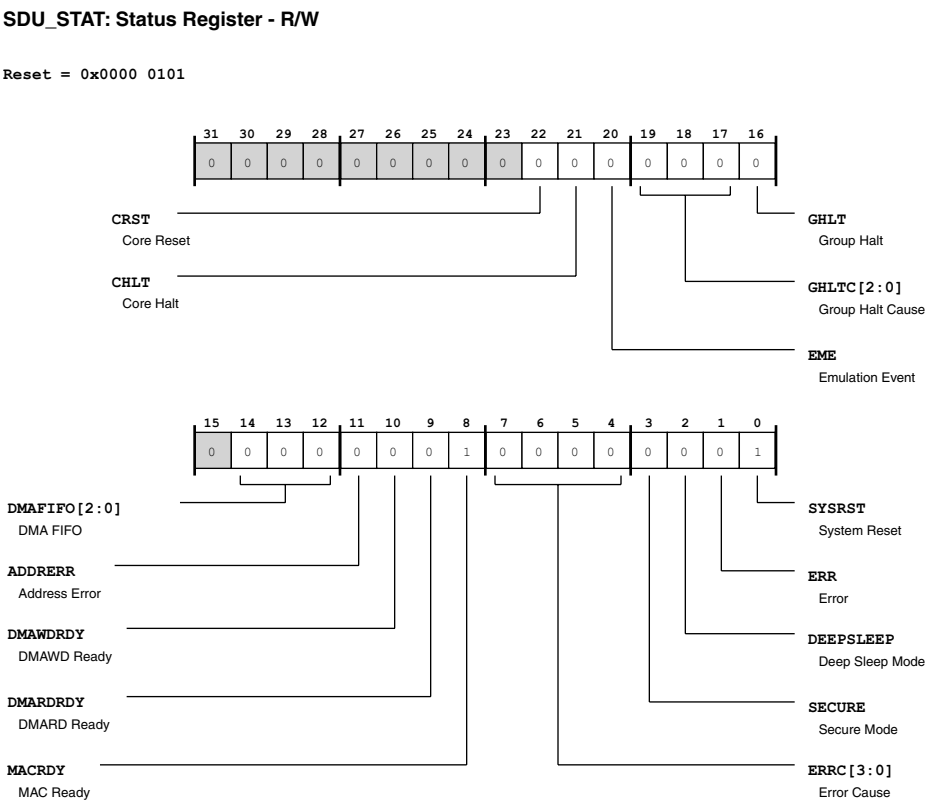


Figure 35-7: SDU_STAT Register Diagram

Table 35-11: SDU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/W1C)	CRST	Core Reset. The SDU_STAT.CRST indicates the core reset has been asserted to one or more of the cores in the system. The core debug status register indicates which core or cores have been reset.	
		0	Inactive
		1	Active
21 (R/NW)	CHLT	Core Halt. The SDU_STAT.CHLT indicates one or more of the cores in the system are halted. The core debug status register indicates which core or cores have halted.	
		0	Inactive
		1	Active

Table 35-11: SDU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W1C)	EME	Emulation Event. The SDU_STAT.EME indicates an emulation event is active and is set when the JTG_TMS pin transitions from 0-to-1 while the SDU_CTL.EMEEN bit is set. The SDU_STAT.EME bit transition to 1 serves as the source for core halt assertion as part of SDU_CTL.EMEEN selection. The emulator clears SDU_STAT.EME with a W1C operation, clearing SDU_STAT.EME for the next core halt by emulation. SDU_STAT.EME is the source for system halt assertion as part of SDU_CTL.EHLT selection. So, event response software must clear SDU_STAT.EME to resume system operation.	
		0	Inactive
		1	Active
19:17 (R/NW)	GHLTC	Group Halt Cause. The SDU_STAT.GHLTC identifies the group halt master. The SDU updates the SDU_STAT.GHLTC bits on SDU_STAT.GHLT assertion.	
		0	System
		...	Reserved
		1	Core 0
		2	Core 1
		3	Reserved
		7	Reserved
16 (R/W1C)	GHLT	Group Halt. The SDU_STAT.GHLT indicates that group halt has been asserted by one of the group halt masters. On SDU_STAT.GHLT assertion, the SDU updates SDU_STAT.GHLTC with the value identifying the group halt cause. SDU_STAT.GHLT is the source for system halt assertion as part of group halt slave selection with the SDU_GHLT.SS0-SDU_GHLT.SS2 bits. So, halt response software must clear SDU_STAT.GHLT to resume system operation.	
		0	Inactive
		1	Active

Table 35-11: SDU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14:12 (R/NW)	DMAFIFO	DMA FIFO. The SDU_STAT.DMAFIFO indicates DMA FIFO full/empty status.	
		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
		6	Reserved
		7	Full
11 (R/W1C)	ADDRERR	Address Error. The SDU_STAT.ADDRERR indicates an attempted write to a read-only register or an access an invalid address.	
		0	Inactive
		1	Active
10 (R/NW)	DMAWDRDY	DMAWD Ready. The SDU_STAT.DMAWDRDY indicates whether the SDU_DMAWD register is ready for the next transfer. When new data is written to the SDU_DMAWD register, the SDU_STAT.DMAWDRDY bit is cleared. When the write data is pushed into the DMA FIFO from the SDU_DMAWD register, the SDU_STAT.DMAWDRDY bit is set, indicating that it is safe to bring in next piece of write data for the DMA transfer. If SDU_DMAWD is written while SDU_STAT.DMAWDRDY is not set, the SDU_STAT register's SDU_STAT.ERR bit in is set (indicating an overflow condition) and the SDU_STAT.ERRC field is updated (indicating the overflow).	
		0	Not Ready
		1	Ready

Table 35-11: SDU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/NW)	DMARDRDY	DMARD Ready. The SDU_STAT.DMARDRDY indicates whether the SDU_DMARD register is ready for the next transfer. When data is read from the SDU_DMARD register, the SDU_STAT.DMARDRDY bit is cleared. When the new read data is loaded into the SDU_DMARD register from the DMA FIFO, the SDU_STAT.DMARDRDY bit is set, indicating that it is safe read (scan out) the next piece of read data for the DMA transfer. If SDU_DMARD is read (scanned out) while SDU_STAT.DMARDRDY is not set, the SDU_STAT register's SDU_STAT.ERR bit in is set (indicating an underflow condition) and the SDU_STAT.ERRC field is updated (indicating the underflow). The SDU_STAT.DMARDRDY bit is cleared when the read data is placed on the scan chain during JTAG scan operation.	
		0	Not Ready
		1	Ready
8 (R/NW)	MACRDY	MAC Ready. The SDU_STAT.MACRDY indicates whether the SDU_MACADDR and SDU_MACDATA registers are ready for the next transfer.	
		0	Not Ready
		1	Ready
7:4 (R/NW)	ERRC	Error Cause. The SDU_STAT.ERRC identifies the error type. The SDU updates SDU_STAT.ERRC on assertion of the SDU_STAT.ERR bit.	
		...	Reserved
		0	MAC over/underflow
		1	DMARD underflow
		2	DMAWD overflow
		3	DMA Error
		4	MAC Bus Error
		5	Reserved
		15	Reserved
3 (R/W1C)	SECURE	Secure Mode. The SDU_STAT.SECURE indicates whether the processor is operating in secure mode.	
		0	Inactive
		1	Active

Table 35-11: SDU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	DEEPSLEEP	Deep Sleep Mode. The SDU_STAT.DEEPSLEEP indicates whether the processor is operating in DEEPSLEEP power mode. The processor's transition to DEEPSLEEP mode sets SDU_STAT.DEEPSLEEP.	
		0	Inactive
		1	Active
1 (R/W1C)	ERR	Error. The SDU_STAT.ERR indicates an SDU related error. The SDU updates SDU_STAT.ERRC on assertion SDU_STAT.ERR with the value identifying the error cause. SDU_STAT.ERRC only captures the first error on SDU_STAT.ERR assertion.	
		0	Inactive
		1	Active
0 (R/W1C)	SYSRST	System Reset. The SDU_STAT.SYSRST indicates that system reset has been asserted. The assertion of system reset sets SDU_STAT.SYSRST.	
		0	Inactive
		1	Active

Memory Access Control Register

The SDU memory access control register (SDU_MACCTL) contains bit fields configuring memory access features.

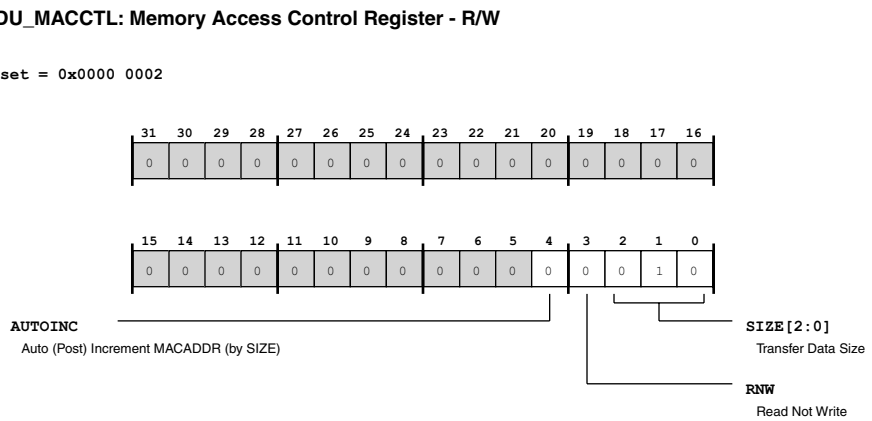


Figure 35-8: SDU_MACCTL Register Diagram

Table 35-12: SDU_MACCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	AUTOINC	Auto (Post) Increment MACADDR (by SIZE). The SDU_MACCTL.AUTOINC enables auto (post) increment of the SDU_MACADDR register value by the SDU_MACCTL.SIZE selection.	
		0	Inactive
		1	Active
3 (R/W)	RNW	Read Not Write. The SDU_MACCTL.RNW selects read or write operation for the MAC transfer.	
		0	Write
		1	Read
2:0 (R/W)	SIZE	Transfer Data Size. The SDU_MACCTL.SIZE selects the size of the data in the SDU_MACDATA register for the MAC transfer.	
		0	8b
		...	Reserved
		1	16b
		2	32b
		3	Reserved
		7	Reserved

Memory Access Address Register

The SDU memory access address register (SDU_MACADDR) contains the address of the MAC transfer.

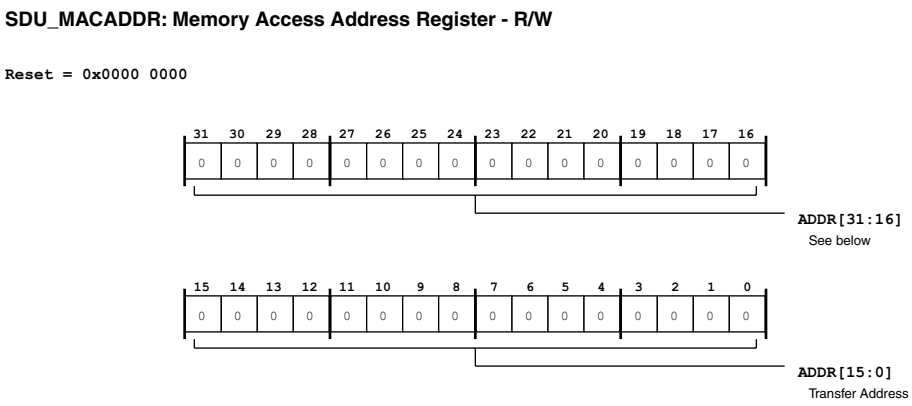


Figure 35-9: SDU_MACADDR Register Diagram

Table 35-13: SDU_MACADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Transfer Address. The SDU_MACADDR.ADDR holds the address for the MAC transfer.

Memory Access Data Register

The SDU memory access data register (SDU_MACDATA) contains the data for the MAC transfer.

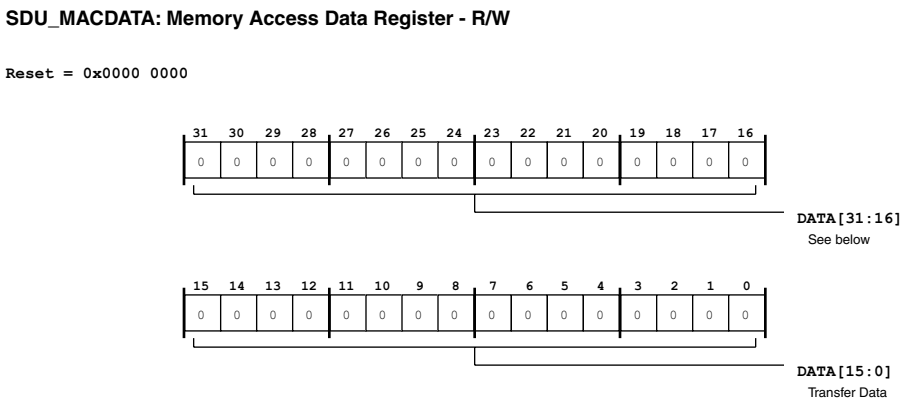


Figure 35-10: SDU_MACDATA Register Diagram

Table 35-14: SDU_MACDATA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Transfer Data. The SDU_MACDATA.DATA holds the data for the MAC transfer.

DMA Read Data Register

The SDU DMA read data register (SDU_DMARD) contains the data for the DMA read transfer.

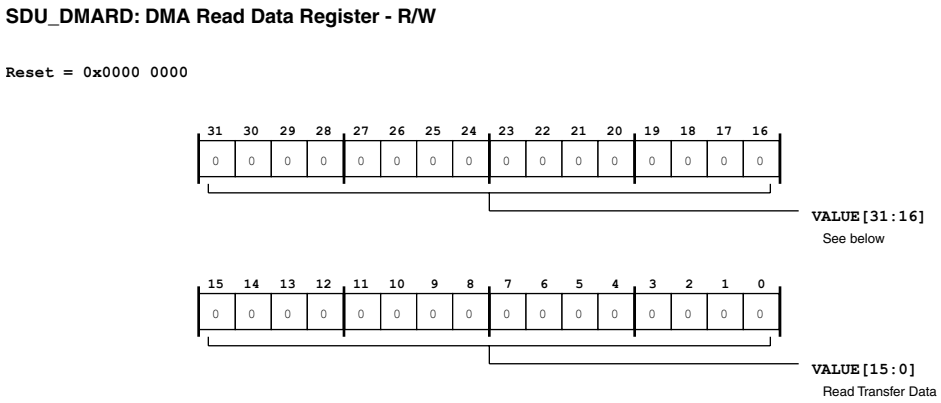


Figure 35-11: SDU_DMARD Register Diagram

Table 35-15: SDU_DMARD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Read Transfer Data. The SDU_DMARD.VALUE holds the data for the DMA read transfer.

DMA Write Data Register

The SDU DMA write data register (SDU_DMAWD) contains the data for the DMA write transfer.

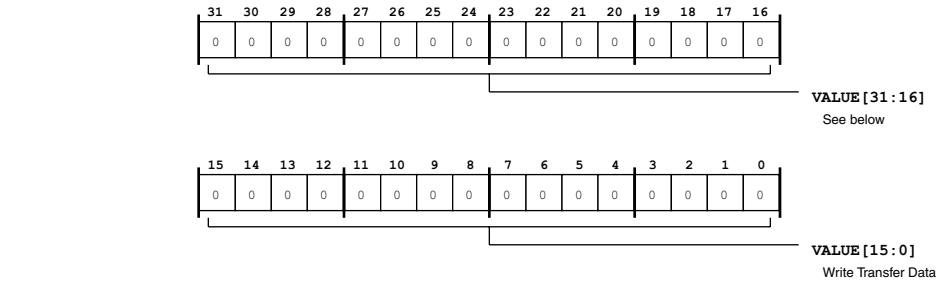


Figure 35-12: SDU_DMAWD Register Diagram

Table 35-16: SDU_DMAWD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Write Transfer Data. The SDU_DMAWD.VALUE holds the data for the DMA write transfer.

Message Register

The SDU message register (SDU_MSG) holds user defined messages to pass to or to get from the debugger.

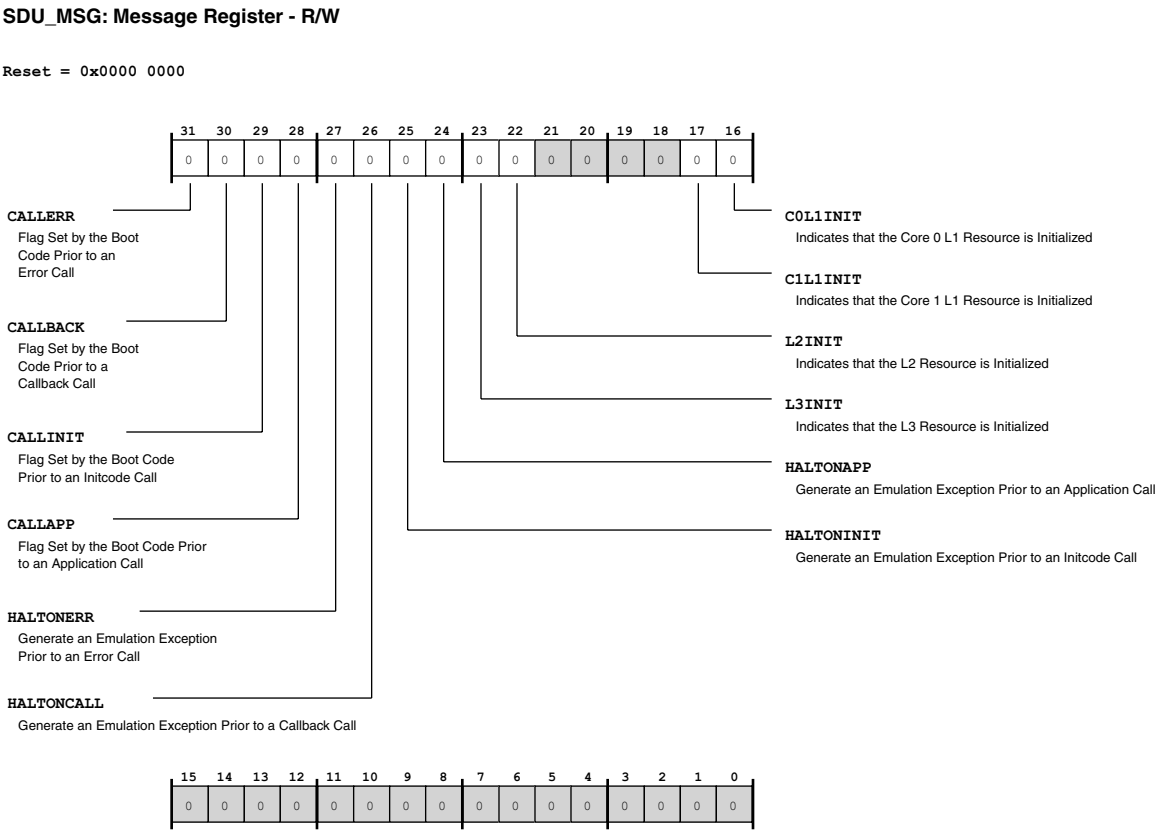


Figure 35-13: SDU_MSG Register Diagram

Table 35-17: SDU_MSG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	CALLERR	Flag Set by the Boot Code Prior to an Error Call.

Table 35-17: SDU_MSG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/W)	CALLBACK	Flag Set by the Boot Code Prior to a Callback Call.
29 (R/W)	CALLINIT	Flag Set by the Boot Code Prior to an Initcode Call.
28 (R/W)	CALLAPP	Flag Set by the Boot Code Prior to an Application Call.
27 (R/W)	HALTONERR	Generate an Emulation Exception Prior to an Error Call.
26 (R/W)	HALTONCALL	Generate an Emulation Exception Prior to a Callback Call.
25 (R/W)	HALTONINIT	Generate an Emulation Exception Prior to an Initcode Call.
24 (R/W)	HALTONAPP	Generate an Emulation Exception Prior to an Application Call.
23 (R/W)	L3INIT	Indicates that the L3 Resource is Initialized.
22 (R/W)	L2INIT	Indicates that the L2 Resource is Initialized.
17 (R/W)	C1L1INIT	Indicates that the Core 1 L1 Resource is Initialized.
16 (R/W)	C0L1INIT	Indicates that the Core 0 L1 Resource is Initialized.

Message Set Register

The SDU message set register (`SDU_MSG_SET`) is a write-1-set alias for the `SDU_MSG` register.

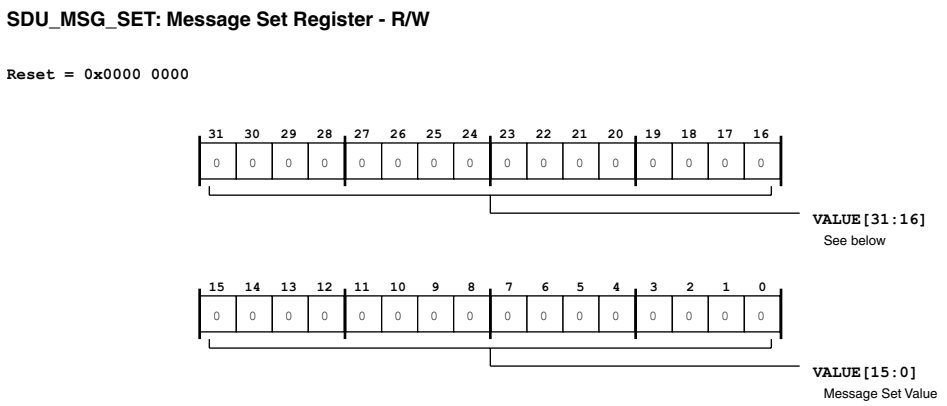


Figure 35-14: SDU_MSG_SET Register Diagram

Table 35-18: SDU_MSG_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W1A)	VALUE	Message Set Value. The SDU_MSG_SET.VALUE acts as a write-1-set alias for the SDU_MSG register. Writing a 1 to any bit in SDU_MSG_SET sets the corresponding bit in SDU_MSG.

Message Clear Register

The SDU message clear register (SDU_MSG_CLR) is a write-1-clear alias for the SDU_MSG register.

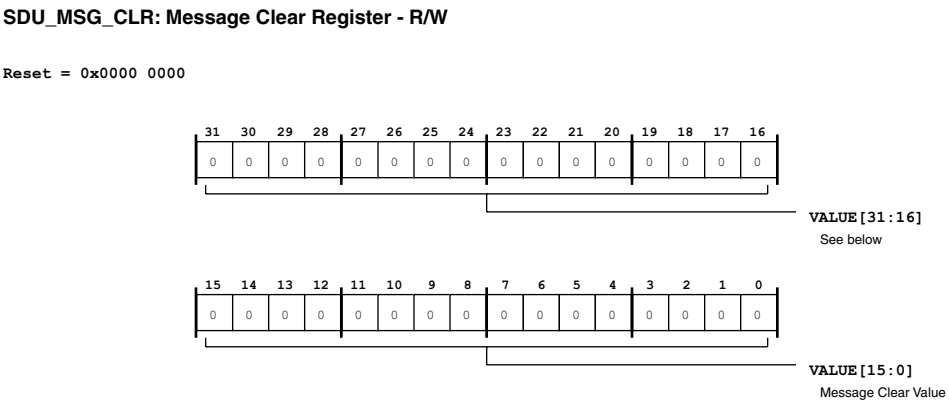


Figure 35-15: SDU_MSG_CLR Register Diagram

Table 35-19: SDU_MSG_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W1A)	VALUE	Message Clear Value. The SDU_MSG_CLR.VALUE acts as a write-1-clear alias for the SDU_MSG register. Writing a 1 to any bit in SDU_MSG_CLR clears the corresponding bit in SDU_MSG.

Group Halt Register

The SDU group halt register (SDU_GHLT) configures the master and slave selections for group halt.

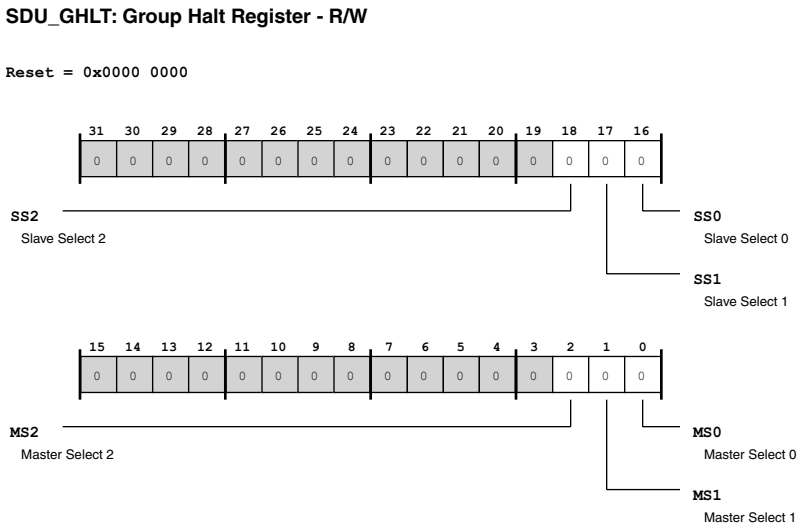


Figure 35-16: SDU_GHLT Register Diagram

Table 35-20: SDU_GHLT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	SS2	Slave Select 2. The SDU_GHLT.SS2 selects core 1 slave for group halt. On SDU_STAT.GHLT assertion, a halt command is issued to the slave.	
		0	Not Selected
		1	Selected

Table 35-20: SDU_GHLT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	SS1	Slave Select 1. The SDU_GHLT.SS1 selects core 0 slave for group halt. On SDU_STAT.GHLT assertion, a halt command is issued to the slave.	
		0	Not Selected
		1	Selected
16 (R/W)	SS0	Slave Select 0. The SDU_GHLT.SS0 selects system slave for group halt. On SDU_STAT.GHLT assertion, a halt command is issued to the slave.	
		0	Not Selected
		1	Selected
2 (R/W)	MS2	Master Select 2. The SDU_GHLT.MS2 selects core 1 master as a source for group halt (SDU_STAT.GHLT) assertion.	
		0	Not Selected
		1	Selected
1 (R/W)	MS1	Master Select 1. The SDU_GHLT.MS1 selects core 0 master as a source for group halt (SDU_STAT.GHLT) assertion.	
		0	Not Selected
		1	Selected
0 (R/W)	MS0	Master Select 0. The SDU_GHLT.MS0 selects system debug events master as a source for group halt (SDU_STAT.GHLT) assertion.	
		0	Not Selected
		1	Selected

36 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 system crossbar address channel signals. 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-BF60x 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 generates an events based on monitoring transactions at the system slaves using four watchpoint match groups. The transaction monitoring within each match group may be filtered by address, ID, direction, and other watchpoint attributes. An event may be a trace message, trigger, or interrupt. The desired event behavior is programmed into the appropriate system watchpoint controls and attributes. The SWU also provides watchpoint event status reporting, a global lock, and processor reset. A set of registers govern SWU operations. For more information on SWU functionality, see the SWU register descriptions.

Table 36-1: ADSP-BF60x SWU Register List

Name	Description
SWU_GCTL	Global Control Register
SWU_GSTAT	Global Status Register
SWU_CTLn	Control Register n
SWU_LAn	Lower Address Register n
SWU_UAn	Upper Address Register n
SWU_IDn	ID Register n
SWU_CNTn	Count Register n
SWU_TARGn	Target Register n
SWU_HISTn	Bandwidth History Register n
SWU_CURn	Current Register n

ADSP-BF60x SWU Interrupt List

Table 36-2: ADSP-BF60x SWU Interrupt List Interrupt List

Description	Interrupt ID	DMA Channel	Sensitivity
SWU0 Event	133		LEVEL
SWU1 Event	134		LEVEL
SWU2 Event	135		LEVEL
SWU3 Event	136		LEVEL
SWU4 Event	137		LEVEL
SWU5 Event	138		LEVEL
SWU6 Event	139		LEVEL

ADSP-BF60x SWU Trigger List

Table 36-3: ADSP-BF60x SWU Trigger List Trigger Masters

Description	Trigger ID	Sensitivity
SWU0 Event	80	PULSE/EDGE
SWU1 Event	81	PULSE/EDGE
SWU2 Event	82	PULSE/EDGE
SWU3 Event	83	PULSE/EDGE
SWU4 Event	84	PULSE/EDGE
SWU5 Event	85	PULSE/EDGE
SWU6 Event	86	PULSE/EDGE

Table 36-4: ADSP-BF60x SWU Trigger List Trigger Slaves

Description	Trigger ID	Sensitivity
SWU0 Event	80	
SWU1 Event	81	
SWU2 Event	82	
SWU3 Event	83	
SWU4 Event	84	
SWU5 Event	85	
SWU6 Event	86	

SWU Definitions

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 Flow Diagram

The following diagram shows the logical program flow of the SWU.

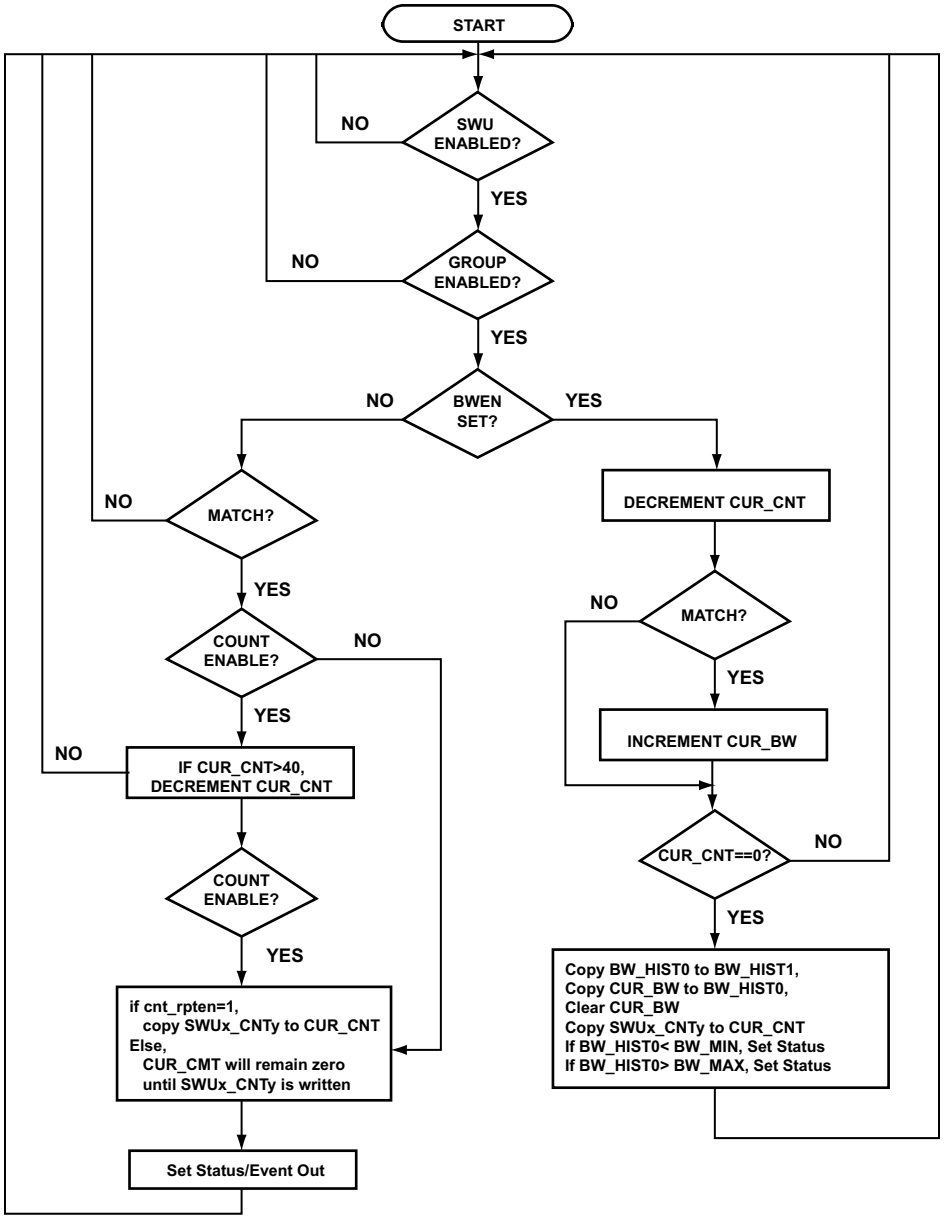


Figure 36-1: SWU Logical Flow

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 following figure shows the SWU block diagram.

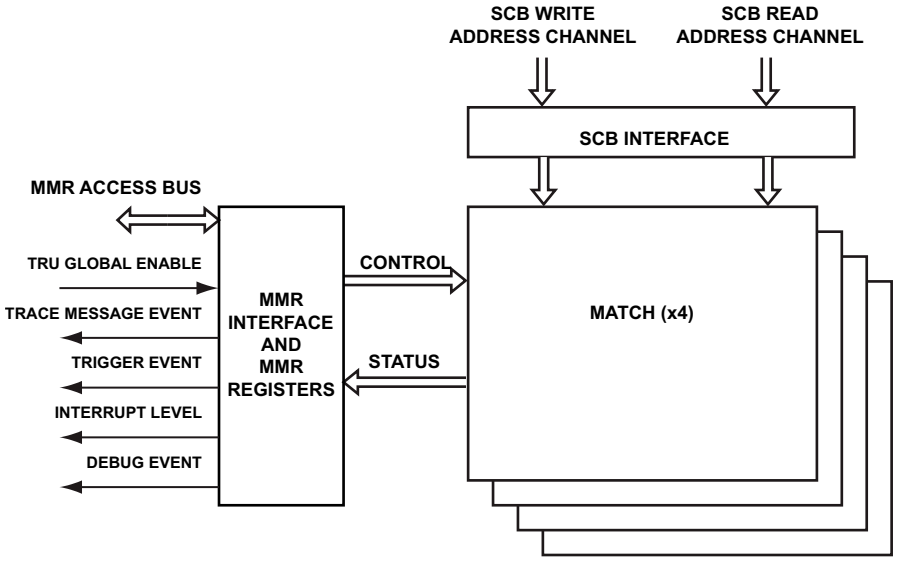


Figure 36-2: System Watchpoint Unit Top Level Block Diagram

System Crossbar 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 Block

The SWU MMR block contains the peripheral bus interface and the SWU MMR registers. It also merges all interrupts 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, transactions which match the properties specified in the `SWU_CTLn` register are counted during a sampling window determined by the respective `SWU_CNTn` register. At the end of the sampling window, results are stored in the `SWU_HISTn` register. If the sampled bandwidth falls outside a programmed range, then the programmed action is taken.

Watchpoint Mode

In watchpoint mode, if the SWU_CTLn.CNTENbit is set, the SWU_CURnregister is decremented for each match, until it equals zero, at which point any programmed actions are taken. The SWU_CURnregister is then reloaded from the SWU_CNTnregister (if the SWU_CTLn.CNTRPTEN bit is set), and the cycle repeats. If the SWU_CTLn.CNTRPTEN bit is not set, any programmed actions are taken on every match.

Match Block

There are four match blocks for each SWU. Each SWU match block can monitor either the read or write address channel, selected by the SWU_CTLn.DIR bit, and can operate in either watchpoint or bandwidth mode, selected by the SWU_CTLn.BWEN bit.

In either mode, the SWU match block can be programmed to match based on address (exact, inclusive/exclusive range), ID (with masking), security, and lock type. All enabled matches are AND'ed together to determine a match.

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
4. Debug

SWU Interrupts

All interrupts 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 only error that the SWU records is an address error when a write or read attempt is made to the SWU's MMR address space and the register does not exist. 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 or not a watchpoint match occurred for each match group.

- Indicate whether or not an interrupt 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 how the trigger routing unit (TRU) is configured. As a trigger master, programs must set the SWU_CTLn.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 if 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, even though the SWU can be enabled by a trigger event as a slave, to disable the SWU, programs must manually clear the SWU_GCTL.EN bit.

SWU Programming Model

The SWU is used by programming the appropriate registers. Each control register is used to configure various aspects such as the direction of monitoring (reads or writes), whether Bandwidth Mode or Watchpoint Mode is used, setting up which events are triggered when a condition is met while monitoring using the SWU, and other parameters. Supplemental registers such as the lower and upper address boundaries are also configured before enabling.

Once the SWU has been enabled and monitoring conditions are met, events are generated if the SWU was configured to do so.

The global status register can be read to observe the current status of the units.

SWU Mode Configuration

The following sections show the steps for configuring the SWU into bandwidth mode and watchpoint mode.

Configuring the SWU for Bandwidth Mode

In bandwidth mode, transactions which match are counted during a sampling window. At the end of the sampling window, results are stored and an action can be taken if the sampled bandwidth goes above and/or falls below a programmed range.

1. Configure the SWU_CTLn.DIR bit to test the match on writes or reads.
2. Configure the SWU_CTLn.ACMPM bits to address whether comparisons are made, exact match, matches inside a range or matches outside a range.
3. If ID If comparison ID is desired, set the SWU_CTLn.IDCMPEN bit.

4. Set the SWU_CTLn.BLENINC bit to increment by burst length or clear it to increment by 1.
5. Configure the SWU_CTLn.MAXACT and SWU_CTLn.MINACT bits to enable actions taken when the bandwidth goes above the maximum, or falls below the minimum, respectively.
6. Set the SWU_CTLn.BWEN=1 to enable bandwidth mode.
7. Program the lower address register, SWU_LAn, and upper address register, SWU_UAn, to define the memory range for comparison.
8. If ID comparison is enabled, program the ID register, SWU_IDn.
9. Program the count register, SWU_CNTn, with the number of clock cycles for which the SWU will be counting the number of matches.
10. If the SWU is set to take action when the bandwidth measurement underflows or overflows, the min and max values should be programmed into the SWU_TARGn register.
11. Enable the SWU

RESULT:

The SWU counts the number of matches in a pre-defined amount of clock cycles programmed by the user. Lower and upper limits can optionally be defined. 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_CTLn.DIR bit to test the match on writes or reads.
2. Configure the SWU_CTLn.ACMPM bits for address comparisons, exact match, matches inside a range or matches outside a range are desired.
3. Optionally set the SWU_CTLn.IDCMPEN if ID comparison is desired.
4. Set the SWU_CTLn.CNTEN bit to enable the events to be triggered when the count decrements to zero.
5. Optionally set the SWU_CTLn.CNTRPTEN bit to automatically reload the counter after it has decremented to zero to start another match sequence.
6. Clear the SWU_CTLn.BWEN = 0 to configure watchpoint mode.
7. Configure the lower address register, SWU_LAn, and upper address register, SWU_UAn, to define the memory range for comparison.
8. If ID comparison is enabled, configure the ID register, SWU_IDn.

9. Configure the count register, SWU_CNTn, to determine how many matches occur before the watchpoint group takes action.
10. Enable the SWU.

RESULT:

The SWU detects and counts down the number of match occurrences. When the counter expires, an action is taken.

ADSP-BF60x SWU Register Descriptions

System Watchpoint Unit (SWU) contains the following registers.

Table 36-5: ADSP-BF60x SWU Register List

Name	Description
SWU_GCTL	Global Control Register
SWU_GSTAT	Global Status Register
SWU_CTLn	Control Register n
SWU_LAn	Lower Address Register n
SWU_UAn	Upper Address Register n
SWU_IDn	ID Register n
SWU_CNTn	Count Register n
SWU_TARGn	Target Register n
SWU_HISTn	Bandwidth History Register n
SWU_CURn	Current Register n

Global Control Register

The SWU global control register (SWU_GCTL) provides SWU reset and enable.

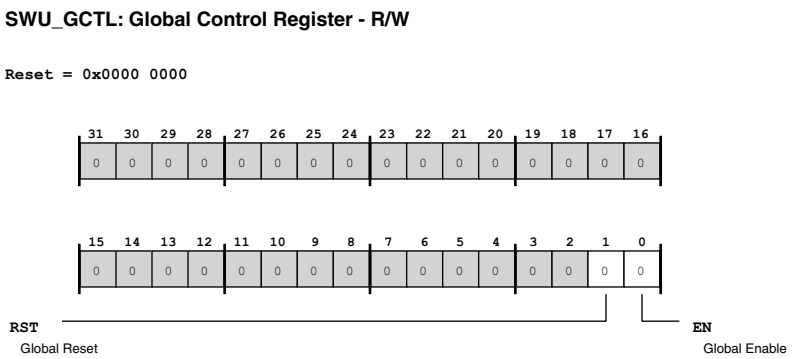


Figure 36-3: SWU_GCTL Register Diagram

Table 36-6: SWU_GCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R0/W1A)	RST	Global Reset. The SWU_GCTL.RST) is write-1-action/read zero and controls the SWU operational state. Setting SWU_GCTL.RST 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 SWU_GCTL.EN controls the SWU operational state. Clearing SWU_GCTL.EN halts the execution of all watchpoint and bandwidth tracking operations without resetting status registers or associated signals. Setting SWU_GCTL.EN 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.

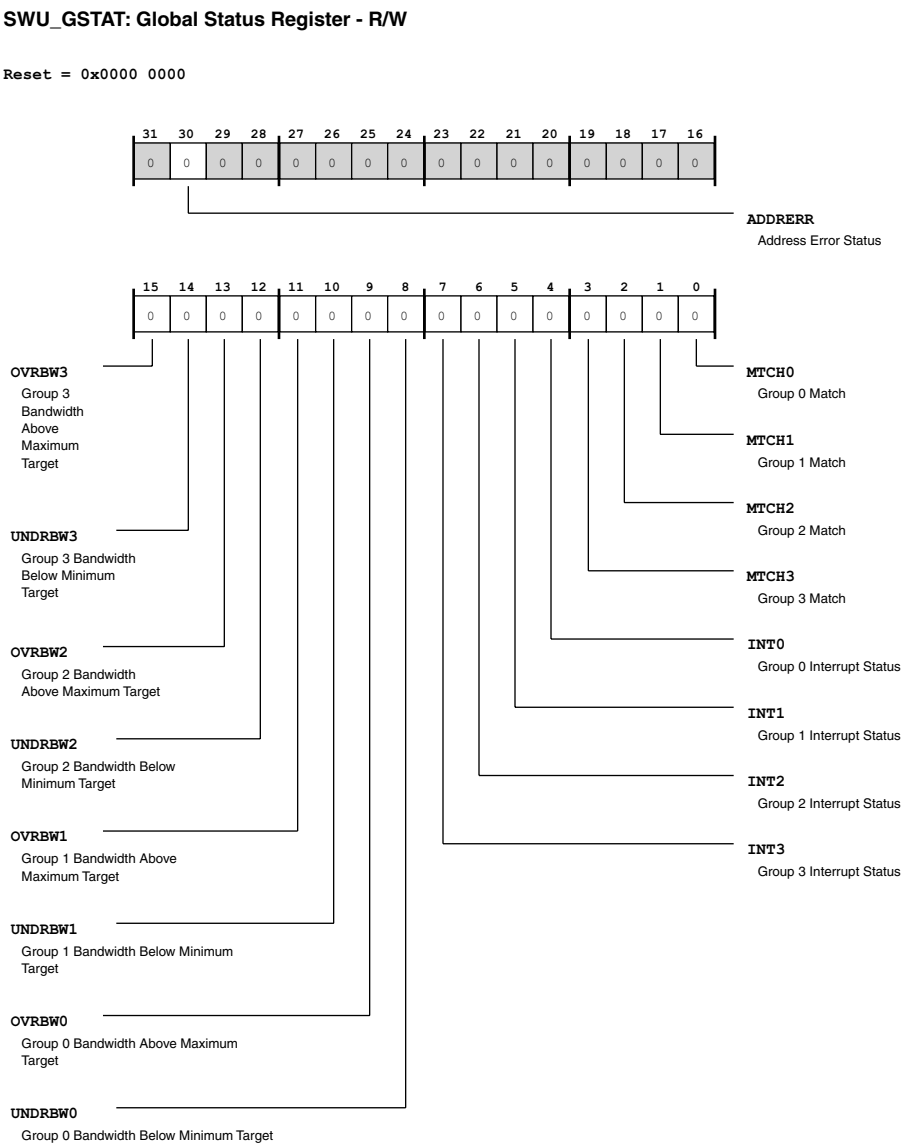


Figure 36-4: SWU_GSTAT Register Diagram

Table 36-7: SWU_GSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W1C)	ADDRERR	Address Error Status. The SWU_GSTAT.ADDRERR indicates that the SWU generated an address error. This status bit is sticky; write-1-to-clear it.	
		0	Inactive
		1	Active

Table 36-7: 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.	
		1	Group 3 was above maximum bandwidth
14 (R/W1C)	UNDRBW3	Group 3 Bandwidth Below Minimum Target. See SWU_GSTAT.UNDRBW0 description.	
		1	Group 3 was below minimum bandwitdth
13 (R/W1C)	OVRBW2	Group 2 Bandwidth Above Maximum Target. See SWU_GSTAT.OVRBW0 description.	
		1	Group 2 was above maximum bandwidth
12 (R/W1C)	UNDRBW2	Group 2 Bandwidth Below Minimum Target. See SWU_GSTAT.UNDRBW0 description.	
		1	Group 2 was below minimum bandwidth
11 (R/W1C)	OVRBW1	Group 1 Bandwidth Above Maximum Target. See SWU_GSTAT.OVRBW0 description.	
		1	Group 1 was above maximum bandwidth
10 (R/W1C)	UNDRBW1	Group 1 Bandwidth Below Minimum Target. See SWU_GSTAT.UNDRBW0 description.	
		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 watchpoint 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_CNTn register was over the maximum target. This status bit is sticky; write-1-to-clear it.	
		1	Group 0 was above maximum bandwidth
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_CNTn register was below the minimum target. This status bit is sticky; write-1-to-clear it.	
		1	Group 0 was below minimum bandwidth

Table 36-7: SWU_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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

Table 36-7: SWU_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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 occured in a SWU watchpoint group, as controlled by the group's related watchpoint control register (SWU_CTLn). This status bit is sticky; write-1-to-clear it.	
		0	No match
		1	Group 0 Watchpoint Match

Control Register n

The SWU control registers (SWU_CTLn) 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 locked 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.

SWU_CTLn: Control Register n - R/W

Reset = 0x0000 0000

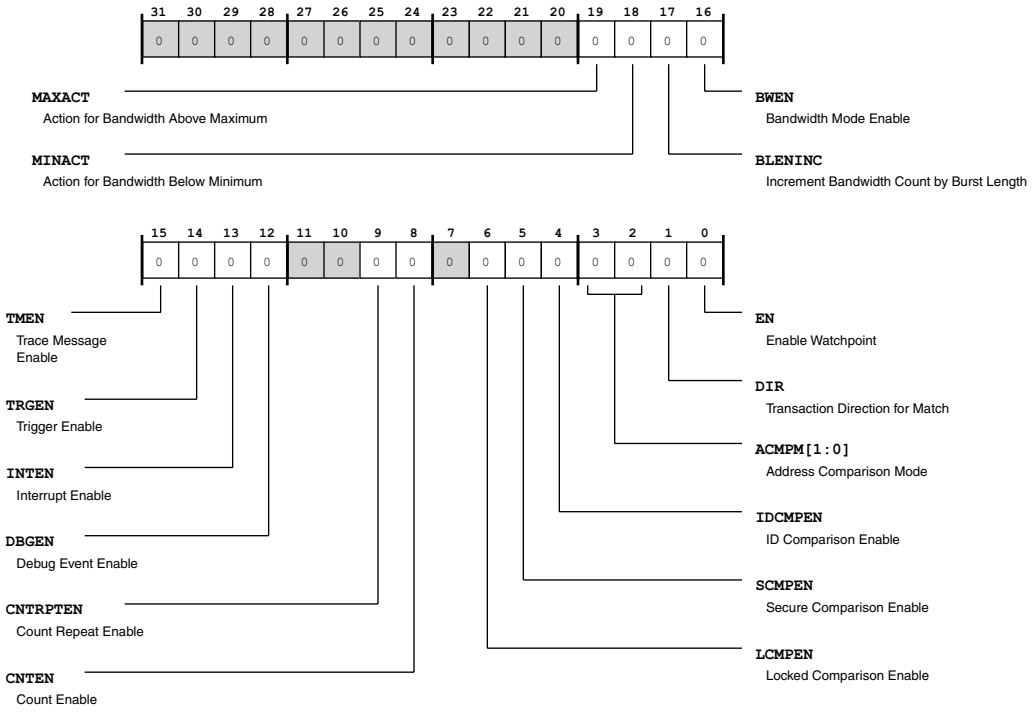


Figure 36-5: SWU_CTLn Register Diagram

Table 36-8: SWU_CTLn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	MAXACT	Action for Bandwidth Above Maximum. Each SWU_CTLn.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_CTLn.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 36-8: SWU_CTLn Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	BLENINC	Increment Bandwidth Count by Burst Length. Each SWU_CTLn.BLENINC bit controls how a watchpoint group's bandwidth count is incremented in the SWU_CURn register's SWU_CURn.CURBW field. If the SWU_CTLn.BLENINC bit is cleared (= 0), the SWU increments the bandwidth count by 1 for each matching transaction. If the SWU_CTLn.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_CTLn.BWEN bit == 1). Note that if the address range match is enabled (SWU_CTLn.ACMPM bits) and if any address of a burst falls within the address range, the SWU_CURn.CURBW field is incremented by the burst length even if some of the burst address fall outside of the range. 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.	
		0	Increment by 1
		1	Burst Length Increment for Bandwidth Count
16 (R/W)	BWEN	Bandwidth Mode Enable. Each SWU_CTLn.BWEN bit controls whether a watchpoint group operates in watchpoint mode or bandwidth mode. In watchpoint mode, the SWU_CTLn.CNTEN and (optionally) SWU_CTLn.CNTRPTEN registers control usage of the cycle count for watchpoint group operations. In bandwidth mode, the SWU_CTLn.BLENINC, SWU_TARGn, and SWU_HISTn registers control usage of watchpoint matches for watchpoint group operations.	
		0	Watchpoint Mode
		1	Bandwidth Mode
15 (R/W)	TMEN	Trace Message Enable. Each SWU_CTLn.TMEN bit controls whether a match for a watchpoint group generates a trace message event. This feature is valid in both bandwidth and watchpoint modes.	
		0	Disable
		1	Enable

Table 36-8: SWU_CTLn Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	TRGEN	Trigger Enable. Each SWU_CTLn.TRGEN bit controls whether a match for a watchpoint group generates a trigger event. This feature is valid in both bandwidth and watchpoint modes.	
		0	Disable
		1	Enable
13 (R/W)	INTEN	Interrupt Enable. Each SWU_CTLn.INTEN bit controls whether a match for a watchpoint group generates an interrupt. This feature is valid in both bandwidth and watchpoint modes.	
		0	Disable
		1	Enable
12 (R/W)	DBGEN	Debug Event Enable. Each SWU_CTLn.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_CTLn.CNTRPTEN bit controls whether the watchpoint group's cycle count is reloaded and repeated after cycle countdown. If the SWU_CTLn register's SWU_CTLn.CNTRPTEN bit is set, the SWU_CURn register's SWU_CURn.CURCNT field is reloaded from SWU_CNTn register's SWU_CNTn.COUNT field, and the countdown starts again. If SWU_CTLn.CNTRPTEN bit is cleared, the expired count remains zero, and no further events are signalled. (See the SWU_CTLn.CNTEN bit description for information regarding the countdown setup.)	
		0	Disable
		1	Enable

Table 36-8: SWU_CTLn Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	CNTEN	Count Enable. Each SWU_CTLn.CNTEN bit controls whether the cycle count in the watchpoint group's SWU_CNTn register is decremented each cycle until it reaches zero. This feature is only valid in watchpoint mode (SWU_CTLn.BWEN bit == 0).When the count reaches zero, any enabled watchpoint events are triggered. (See the SWU_CTLn.CNTRPTEN bit description for optional actions at that may occur at the end of the countdown.)	
		0	Disable
		1	Enable
6 (R/W)	LCMPEN	Locked Comparison Enable. Each SWU_CTLn.LCMPEN bit controls locked comparison operation of an SWU watchpoint group, permitting matches based on lock status.	
		0	Match on all transaction
		1	Match only locked transactions
5 (R/W)	SCMPEN	Secure Comparison Enable. Each SWU_CTLn.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_CTLn.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_IDn register.	
3:2 (R/W)	ACMPM	Address Comparison Mode. Each set of SWU_CTLn.ACMPM bits control the address comparison operation of an SWU watchpoint group. The address within range for comparison is defined as (SWU_LAn register <= address < SWU_UAn register). The address outside range for comparison is defined as (address < SWU_LAn) or (SWU_UAn<= address).	
		0	No address comparison
		1	Exact match on LAn
		2	Match on address within range
		3	Match on address outside range

Table 36-8: SWU_CTLn Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	DIR	Transaction Direction for Match. Each SWU_CTLn.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_CTLn.EN bit controls the operation of one SWU watchpoint group. Clearing the SWU_CTLn.EN bit halts the execution of watchpoint or bandwidth tracking operations in the watchpoint group without resetting status or configuration registers. Setting the SWU_CTLn.EN bit enables the SWU watchpoint group to begin or resume operation with the current configuration and status.	
		0	Disable
		1	Enable

Lower Address Register n

The SWU lower address registers (SWU_LAn) contain each watchpoint group's lower address for address match comparison. In exact match on SWU_LAn address mode (SWU_CTLn.ACMPM bits =01), the watchpoint group uses only this address for match comparison.

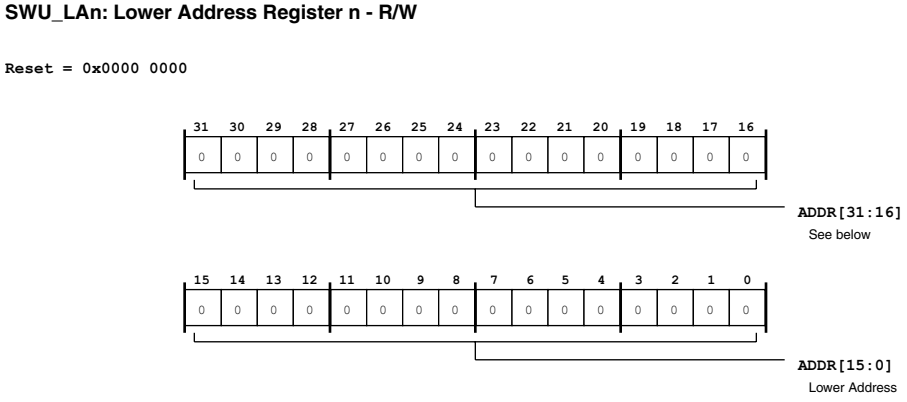


Figure 36-6: SWU_LAn Register Diagram

Table 36-9: SWU_LAn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Lower Address.

Upper Address Register n

The SWU upper address registers (SWU_UAn) contain each watchpoint group's upper address for address match comparison. In exact match on SWU_LAn address mode (SWU_CTLn.ACMPM bits =01), the SWU_UAn is not used for match comparison.

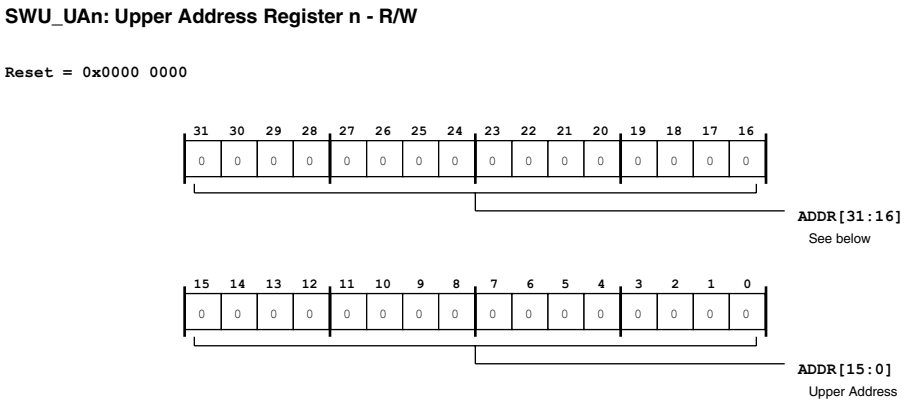


Figure 36-7: SWU_UAn Register Diagram

Table 36-10: SWU_UAn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Upper Address.

ID Register n

The SWU ID registers (SWU_IDn) contain a 16-bit ID field (SWU_IDn.ID) and a 16-bit ID mask field (SWU_IDn.IDMASK) that watchpoint groups use for ID comparison. The ID on the bus is AND'ed with the SWU_IDn.IDMASK field, then the watchpoint group compares the result against the SWU_IDn.ID field.

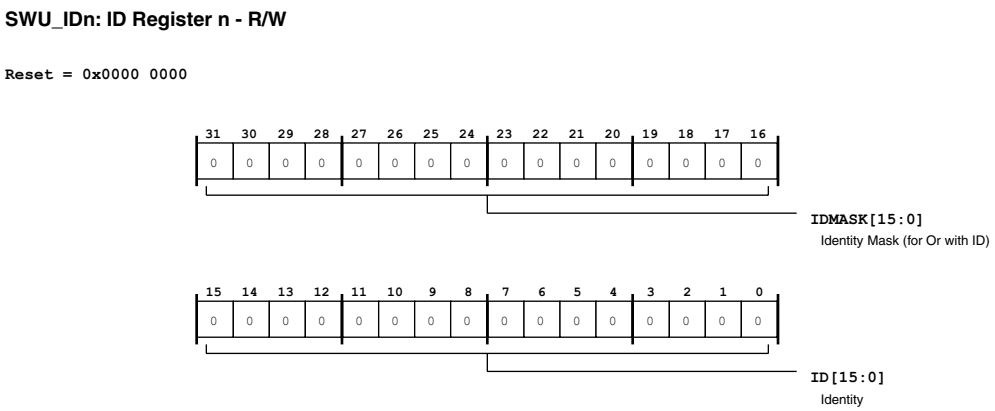


Figure 36-8: SWU_IDn Register Diagram

Table 36-11: SWU_IDn 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.

Count Register n

The SWU count registers (SWU_CNTn) contain a 16-bit count field (SWU_CNTn.COUNT) whose usage differs depending on the mode of the watchpoint group. In bandwidth mode, the SWU_CNTn.COUNT field value defines the number of clock cycles in a bandwidth period. In watchpoint mode, when the cycle count is enabled, the SWU_CNTn.COUNT field value determines how many matches occur before the watchpoint group takes action.

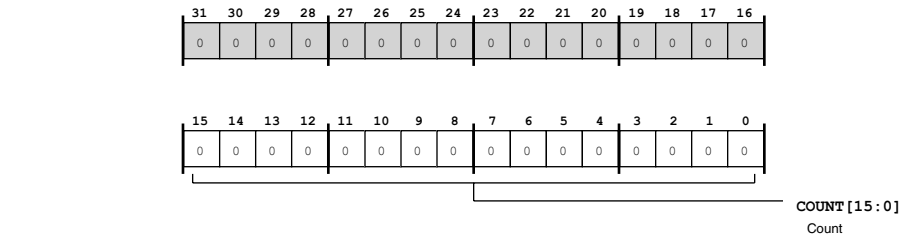


Figure 36-9: SWU_CNTn Register Diagram

Table 36-12: SWU_CNTn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	COUNT	Count.

Target Register n

The SWU target registers (SWU_TARGn) contain a minimum value field (SWU_TARGn.BWMIN) and maximum value field (SWU_TARGn.BWMAX) of bandwidth targets used by watchpoint groups in bandwidth mode. When the bandwidth period expires, if the current bandwidth value (SWU_CURn register, SWU_CURn.CURBW bits) is below the minimum target or above the maximum target, the watchpoint group takes action as enabled by the SWU_CTLn register's SWU_CTLn.MINACT or SWU_CTLn.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_CTLn.BLENINC bit. Keep this mode selection in mind when programming the bandwidth target values.

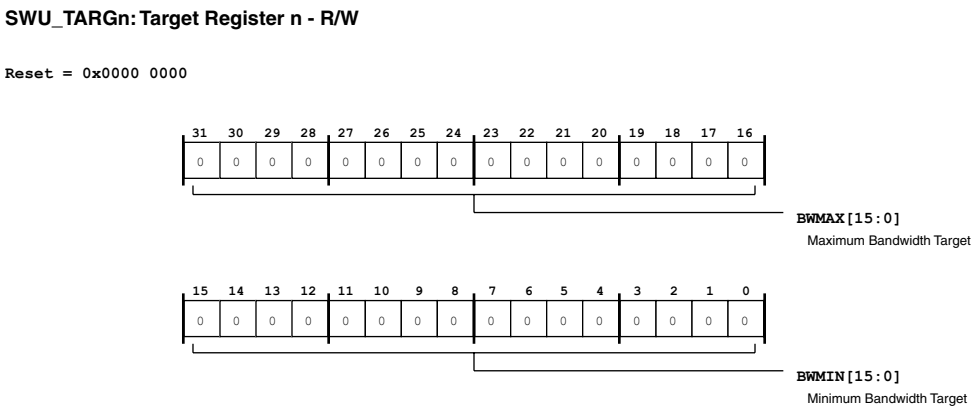


Figure 36-10: SWU_TARGn Register Diagram

Table 36-13: SWU_TARGn 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.

Bandwidth History Register n

The SWU bandwidth history registers (SWU_HISTn) contain data copied from a watchpoint group's current bandwidth value (SWU_CURn register, SWU_CURn.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_HISTn.BWHIST0 field to the SWU_HISTn.BWHIST1 field and copies the new bandwidth value from the SWU_CURn.CURBW field to the SWU_HISTn.BWHIST0 field.

SWU_HISTn: Bandwidth History Register n - R/W

Reset = 0x0000 0000

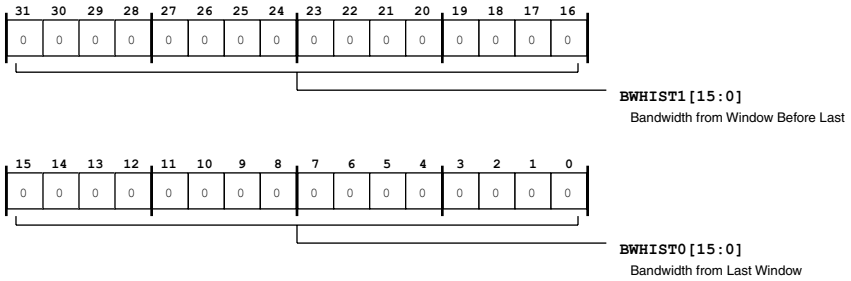


Figure 36-11: SWU_HISTn Register Diagram

Table 36-14: SWU_HISTn 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.

Current Register n

The SWU current register (SWU_CURn) 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_CURn.CURCNT field from the SWU_CNTn register's SWU_CNTn.COUNT field when the watchpoint count is enabled (SWU_CTLn register, SWU_CTLn.CNTEN bit =1).

In bandwidth mode, the current count field (SWU_CURn.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_CURn.CURCNT field from SWU_CNTn register's SWU_CNTn.COUNT field. In bandwidth mode, the current bandwidth field (SWU_CURn.CURBW) contains the count of watchpoint matches (bandwidth) accummulated in the current watchpoint period.

In watchpoint mode, the current count field (SWU_CURn.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_CURn.CURCNT field from SWU_CNTn register's SWU_CNTn.COUNT field if the SWU_CTLn register's SWU_CTLn.CNTRPTEN bit is set (=1). In watchpoint mode, the current bandwidth field (SWU_CURn.CURBW) is undefined.

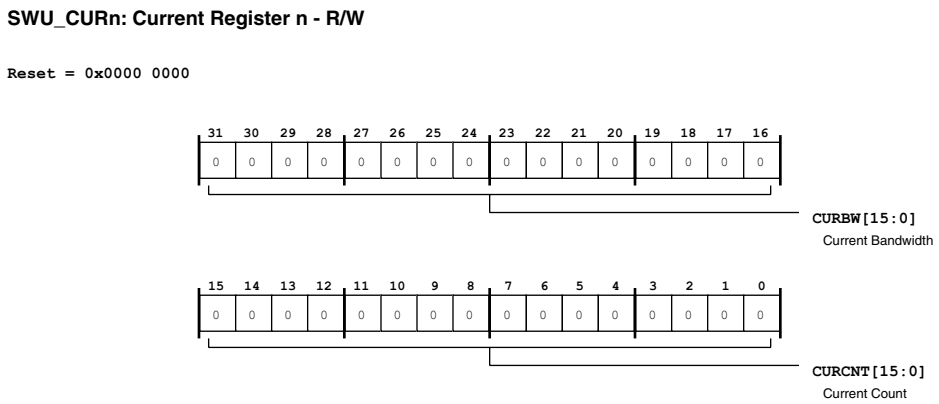


Figure 36-12: SWU_CURn Register Diagram

Table 36-15: SWU_CURn Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	CURBW	Current Bandwidth.
15:0 (R/NW)	CURCNT	Current Count.

I Index

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Y