ADSP-214xx SHARC® Processor Hardware Reference

Includes ADSP-2146x, ADSP-2147x, ADSP-2148x

Revision 1.1, April 2013

Part Number
82-000469-01

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

ANALOG DEVICES

CONTENTS

I/O PROCESSOR

EXTERNAL PORT

PULSE WIDTH MODULATION

MEDIA LOCAL BUS

PERIPHERAL TIMERS

SYSTEM DESIGN

REGISTER LISTING

AUDIO FRAME FORMATS

INDEX

PREFACE

Thank you for purchasing and developing systems using SHARC® processors from Analog Devices.

Purpose of This Manual

ADSP-214xx SHARC Processor Hardware Reference contains information about the peripherals associated with ADSP-214xx processors. These are 32-bit, fixed- and floating-point digital signal processors from Analog Devices for use in computing, communications, and consumer applications.

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 hardware and programming reference manuals that describe their target architecture.

Manual Contents

This manual provides detailed information about the ADSP-214xx processor peripherals in the following chapters:

- Chapter 1, "Introduction"
  Provides an architectural overview of the SHARC processors.
- Chapter2 "Interrupt Control"
  Provides a functional description of the system interrupt controller including a complete listing of the registers that are used to configure and control interrupts.
- Chapter 3, "I/O Processor"
  Describes input/output processor architecture, and provides direct memory access (DMA) procedures for the processor peripherals.
- Chapter 4, "External Port"
  Describes how the processor's connect to external memories. These include DDR2 (ADSP-2146x) and SDRAM (ADSP-2147x, ADSP-2148x).
- Chapter 5, "Link Ports – ADSP-2146x"
  Describes the two bidirectional 8-bit wide link ports, which can connect to other processor or peripheral link ports.
- Chapter 6, "Memory-to-Memory Port DMA"
  Describes on-chip memory-to-memory DMA.
- Chapter 7 "FFT/FIR/IIR Hardware Modules"
  Describes the dedicated hardware accelerators used to reduce the instruction load on the core, freeing it up for other tasks, effectively adding more bandwidth.

- Chapter 8, "Pulse Width Modulation"
Describes the implementation and use of the pulse width modulation module which provides a technique for controlling analog circuits with the microprocessor's digital outputs.

- Chapter 9, "Media Local Bus"
Details the Media Local Bus port (MLB), an on-PCB or inter-chip communication bus, which allows an application to access MOST network data via an INIC (intelligent network interface controller).

- Chapter 10, "Digital Application/ Digital Peripheral Interfaces"
Provides information about the digital audio/digital peripheral interface (DAI/DPI) which allows you to attach an arbitrary number and variety of peripherals to the SHARC processor while retaining high levels of compatibility.

- Chapter 11, "Serial Ports (SPORTs)"
Describes the data line serial ports. Each SPORT contains a clock, a frame sync, and two data lines that can be configured as either a receiver or transmitter pair.

- Chapter 12, "Input Data Port (SIP, PDAP)"
Discusses the function of the input data port (IDP) which provides a low overhead method of routing signal routing unit (SRU) signals back to the core's memory.

- Chapter 13, "Asynchronous Sample Rate Converter"
Provides information on the sample rate converter (SRC) module. This module performs synchronous or asynchronous sample rate conversion across independent stereo channels, without using any internal processor resources.

- Chapter 14, "Sony/Philips Digital Interface"
  Provides information on the use of the Sony/Philips Digital Interface which is a standard audio file transfer format that allows the transfer of digital audio signals from one device to another without having to be converted to an analog signal.
- Chapter 15, "Precision Clock Generator"
  Details the precision clock generators (PCG), each of which generates a pair of signals derived from a low jitter based off-chip clock input signal.
- Chapter 16, "Serial Peripheral Interface Ports"
  Describes the operation of the serial peripheral interface (SPI) port. SPI devices communicate using a master-slave relationship and can achieve high data transfer rate because they can operate in full-duplex mode.
- Chapter 17, "Peripheral Timers"
  Describes the 32-bit timers that can be used to interface with external devices.
- Chapter 18, "Shift Register – ADSP-2147x"
  Describes the 18 stage serial in, serial/parallel out shift register.
- Chapter 19, "Real-Time Clock – ADSP-2147x/ADSP-2148x"
  Describes the real time clock which operates independent of the processor clocks.
- Chapter 20, "WatchDog Timer – ADSP-2147x, ADSP-2148x"
  Describes software watchdog function which can improve system reliability by forcing the processor to a known state.
- Chapter 21, "UART Port Controller"
  Describes the operation of the Universal Asynchronous Receiver/Transmitter (UART) which is a full-duplex peripheral compatible with PC-style industry-standard UART.

- Chapter 22, "Two-Wire Interface Controller"
  The two-wire interface is fully compatible with the widely used I^2C bus standard. It is designed with a high level of functionality and is compatible with multi-master, multi-slave bus configurations.

- Chapter 23, "System Design"
  Describes system design features of the ADSP-214xx processors. These include power, reset, clock, JTAG, and booting, as well as pin multiplexing schemes and other system-level information.

- Chapter 24, "Power Management"
  Describes system design features as they relate to power management.

- Appendix A, "Register Reference"
  Provides a graphical presentation of all registers and describes the bit usage in each register.

- Appendix B, "Register Listing"
  Provides the register mnemonic, address, brief description, and state at reset of all registers.

- Appendix C "Audio Frame Formats"
  Provides descriptions on the standard audio formats used by many of the peripherals.

This hardware reference is a companion document to *SHARC Processor Programming Reference*.

What's New in This Manual

This manual is Revision 1.1 of *ADSP-214xx SHARC Processor Hardware Reference*. This revision corrects minor typographical errors and the following issues:

- IOP throughput in Chapter 3, "I/O Processor".
- LACK signal sampling edge in Chapter 5, "Link Ports – ADSP-2146x".
- Descriptions of Save Biquad State mode and IIR throughput in Chapter 7, "FFT/FIR/IIR Hardware Modules".
- Number of SRU groups in Chapter 10, "Digital Application/ Digital Peripheral Interfaces".
- Timer period equation in Chapter 17, "Peripheral Timers".
- ESD/EOS protection circuits in Chapter 24, "System Design".
- PLLM, DIVEN, and IIR_DMASVDk bit descriptions in Appendix A, "Register Reference".

Technical Support

You can reach Analog Devices processors and DSP technical support 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 directly at:
  `http://www.analog.com/support`
- E-mail your questions about processors, DSPs, and tools development software from **CrossCore® Embedded Studio** or **VisualDSP++®**:

Choose **Help > Email Support**. This creates an e-mail to `processor.tools.support@analog.com` and automatically attaches your **CrossCore Embedded Studio** or **VisualDSP++** version information and `license.dat` file.

- E-mail your questions about processors and processor applications to:
  `processor.support@analog.com` or
  `processor.china@analog.com` (Greater China support)
- In the **USA only**, call **1-800-ANALOGD** (1-800-262-5643)
- Contact your Analog Devices sales office or authorized distributor. Locate one at:
  `www.analog.com/adi-sales`
- Send questions by mail to:
  Processors and DSP Technical Support
  Analog Devices, Inc.
  Three Technology Way
  P.O. Box 9106
  Norwood, MA 02062-9106
  USA

Supported Processors

The name "*SHARC*" refers to a family of high-performance, floating-point embedded processors. Refer to the CCES or VisualDSP++ online help for a complete list of supported processors.

Product Information

Product information can be obtained from the Analog Devices Web site and the CCES or VisualDSP++ online help.

Analog Devices Web Site

The Analog Devices Web site, 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 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 provides access to books, application notes, data sheets, code examples, and more.

Visit myAnalog 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, Inc. 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 in this manual are identified and described as follows.

Example	Description
File > Close	Titles in reference sections indicate the location of an item within the IDE environment'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 ellipsis; 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.
	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: 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.
	Warning: Injury to device users may result if ... A Warning identifies conditions or inappropriate usage of the product that could lead to conditions that are potentially hazardous for devices users. In the online version of this book, the word **Warning** appears instead of this symbol.

Register Diagram Conventions

Register diagrams use the following conventions:

- The descriptive name of the register appears at the top, followed by the short form of the name in parentheses.
- If the register is read-only (RO), write-1-to-set (W1S), or write-1-to-clear (W1C), this information appears under the name. Read/write is the default and is not noted. Additional descriptive text may follow.
- If any bits in the register do not follow the overall read/write convention, this is noted in the bit description after the bit name.
- If a bit has a short name, the short name appears first in the bit description, followed by the long name in parentheses.
- The reset value appears in binary in the individual bits and in hexadecimal to the right 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.

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

The following figure shows an example of these conventions.

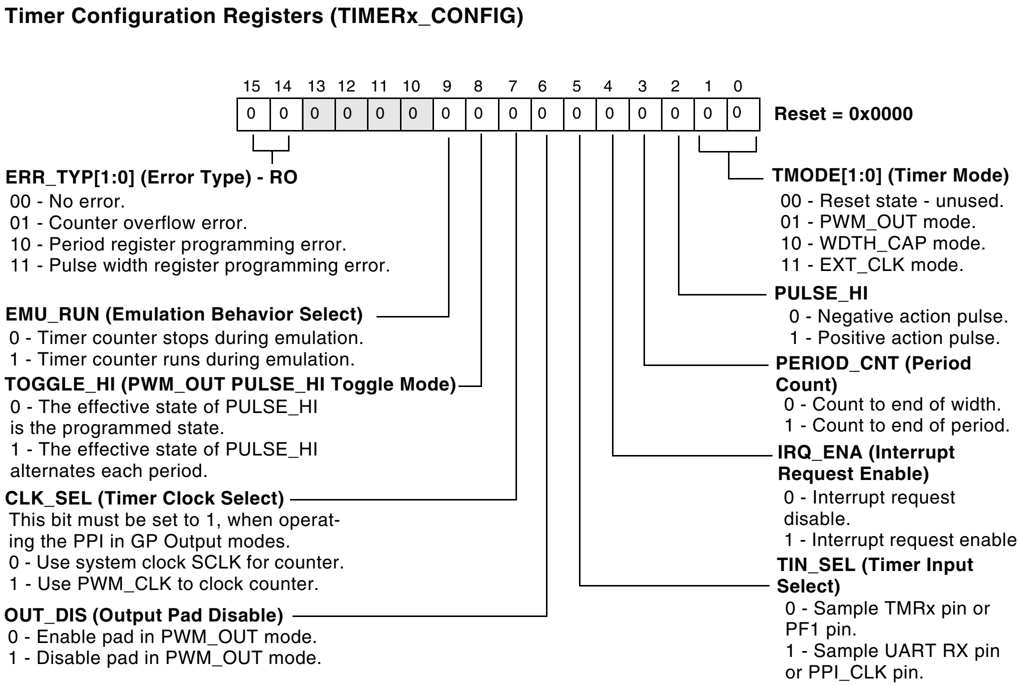

Figure 1. Register Diagram Example

1 INTRODUCTION

The ADSP-214xx SHARC processors are high performance 32-bit processors used for high quality audio, medical imaging, communications, military, test equipment, 3D graphics, speech recognition, motor control, imaging, and other applications. By adding on-chip SRAM, integrated I/O peripherals, and an additional processing element for single-instruction multiple-data (SIMD) support, this processor builds on the ADSP-21xxx family DSP core to form a complete system-on-a-chip.

Design Advantages

A digital signal processor's data format determines its ability to handle signals of differing precision, dynamic range, and signal-to-noise ratios. Because floating-point DSP math reduces the need for scaling and the probability of overflow, using a floating-point processor can simplify algorithm and software development. The extent to which this is true depends on the floating-point processor's architecture. Consistency with IEEE workstation simulations and the elimination of scaling are clearly two ease-of-use advantages. High level language programmability, large address spaces, and wide dynamic range allow system development time to be spent on algorithms and signal processing concerns, rather than assembly language coding, code paging, and error handling. The SHARC processors described in this manual are highly integrated, 32-bit/40-bit floating-point processor's which provide all of these design advantages.

SHARC Family Product Offerings

The products described in this manual offer a variety of features and performance. A complete list of features and specifications can be found in the product-specific data sheet.

Some models of these products are available with controlled manufacturing to support the quality and reliability requirements of automotive applications. Contact your local ADI account representative for specific product ordering information and to obtain the specific Automotive Reliability reports for these models.

Processor Architectural Overview

The ADSP-214xx processors form a complete system-on-a-chip, integrating a large, high speed SRAM and I/O peripherals supported by a dedicated I/O bus. The following sections summarize the features of each functional block in the SHARC architecture.

Processor Core

The processor core consists of two processing elements (each with three computation units and data register file), a program sequencer, two data address generators, a timer, and an instruction cache. Digital signal processing occurs primarily in the processor core.

I/O Peripherals

These peripherals are coupled with the external port and therefore independent from the routing units.

- Asynchronous Memory Interface (AMI)
- SDRAM controller (ADSP-2147x, ADSP-2148x)

- DDR2 controller (ADSP-2146x)
- 4 PWM modules
- The FFT, FIR, and IIR accelerators each contain dedicated signal processing units to off load core processing for these units.
- Link ports for inter-chip communication

I/O Processor

The input/output processor (IOP) manages the off-chip data I/O to free the core from this burden. Up to 67 peripheral DMA channels are multi-stage arbitrated into internal or external memory. For model specific information, see the product-specific data sheet.

Digital Audio Interface (DAI)

The digital audio interface (DAI) unit consists of an interrupt controller, a signal routing unit, and many peripherals:

- 8 serial ports (SPORT)
- Input Data Port (IDP)
- 4 precision clock generators (PCG)
- Some family members have an S/PDIF receiver/transmitter
- 4 asynchronous sample rate converters (ASRC)
- DTCP encryption

DAI System Interrupt Controller

The DAI contains its own interrupt controller that indicates to the core when DAI audio events have occurred. This interrupt controller offers 32 independently configurable channels.

Signal Routing Unit

Conceptually similar to a “patch-bay” or multiplexer, the SRU provides a group of registers that define the interconnection of the DAI peripherals to the DAI pins or to other DAI peripherals.

Digital Peripheral Interface (DPI)

The digital peripheral interface (DPI) unit consists of an interrupt controller, a signal routing unit, and many peripherals:

- 2 serial peripheral interface ports (SPI)
- 2 peripheral timers
- 1 UART
- 1 TWI controller (I^2C compatible)

DPI System Interrupt Controller

The DPI contains its own interrupt controller that indicates to the core when DPI audio events have occurred. This interrupt controller offers 12 independently configurable channels.

Signal Routing Unit 2

Conceptually similar to a “patch-bay” or multiplexer, the SRU2 provides a group of registers that define the interconnection of the DPI peripherals to the DPI pins or to other DPI peripherals.

Differences from Previous Processors

This section identifies differences between the ADSP-214xx processors and previous SHARC processors: ADSP-21161, ADSP-21160, ADSP-21060, ADSP-21061, ADSP-21062, and ADSP-21065L. Like the

ADSP-2116x family, the ADSP-214xx SHARC processor family is based on the original ADSP-2106x SHARC family. The ADSP-214xx processors preserve much of the ADSP-2106x architecture and is code compatible to the ADSP-21160, while extending performance and functionality. For background information on previous generations of SHARC processors and the ADSP-2106x family DSPs, see *ADSP-2106x SHARC User's Manual* or *ADSP-21065L SHARC DSP Technical Reference*.

I/O Architecture Enhancements

The I/O processor provides much greater throughput than the ADSP-2116x processors. This architecture incorporates two independent DMA buses versus the previous SHARC DMA controllers:

- One peripheral DMA bus (IOD0)
- One external port DMA bus (IOD1)

This allows operation of all external port DMA accesses independently from the peripheral buses since up to four internal memory blocks are addressable without bus conflicts.

The SPORT modules can perform DMA accesses directly into the SDRAM/DDR2 memory space without the need to split the memory space into I/O-to- internal memory and internal-to-external memory.

The core access bus to the external port has been increased in I/O size from 48 bits (previous SHARC processors) to 64 bits. This enhancement allows external memory access in SIMD mode operation by accessing data directly between external memory and the PEY unit.

Moreover, external instruction fetch has also been enhanced. The application decides (depending on the address space) for normal word to fetch traditional ISA instructions or in the short word space to fetch VISA instructions.

Development Tools

The processor is supported by a complete set of software and hardware development tools, including Analog Devices' emulators and the CrossCore Embedded Studio or VisualDSP++ development environment. (The emulator hardware that supports other Analog Devices processors also emulates the processor.)

The development environments support advanced application code development and debug with features such as:

- Create, compile, assemble, and link application programs written in C++, C, and assembly
- Load, run, step, halt, and set breakpoints in application programs
- Read and write data and program memory
- Read and write core and peripheral registers
- Plot memory

Analog Devices DSP emulators use the IEEE 1149.1 JTAG test access port to monitor and control the target board processor during emulation. The emulator provides full speed emulation, allowing inspection and modification of memory, registers, and processor stacks. Nonintrusive in-circuit emulation is assured by the use of the processor JTAG interface—the emulator does not affect target system loading or timing.

Software tools also include Board Support Packages (BSPs). Hardware tools also include standalone evaluation systems (boards and extenders). In addition to the software and hardware development tools available from Analog Devices, third parties provide a wide range of tools supporting the Blackfin processors. Third party software tools include DSP libraries, real-time operating systems, and block diagram design tools.

2 INTERRUPT CONTROL

This chapter provides information about controlling interrupts as well as a complete listing of the registers that are used to configure and control programmable interrupts. For information on the `IRPTL`, `LIRPTL`, and `IMASK` registers, see *SHARC Processor Programming Reference*.

Table 2-1. Link Port Specifications

Feature	DAI	DPI
Total Channels	32	12
Peripheral Channels	22	3
Miscellaneous Channels	10	9
Local Priorities	Yes (high, low)	No
Rising Falling Edge	Yes	Yes
Interrupt to Core	2	1
Clock Operation	$f_{PCLK}/4$	$f_{PCLK}/4$

Features

Features include the following:

- Two system interrupt controllers (DAI SIC, DPI SIC) are connected to the core interrupt controller.
- The DAI SIC allows high or low interrupt priority configuration options.

- The DAI interrupt controller offers up to 32 independently configurable channels.
- The DPI interrupt controller offers up to 12 independently configurable channels.
- Both controllers allow latching on rising or/and falling edges of waveform events.
- Same interrupt latency as core latched interrupts.

Clocking

The fundamental timing clock of the system interrupt controllers is peripheral clock/4. ($f_{PCLK}/4$). All interrupt requests are acknowledged and responded to with up to $f_{PCLK}/4$ speed.

Register Overview

Programmable Interrupt Control Registers (PICR3–0). Nineteen peripherals can be routed to the programmable interrupt inputs with the purpose to assign individual priorities to each peripheral channel.

DAI Interrupt Mask Registers (DAI_IMASK_RE/FE). Masks interrupt for rising and/or falling edge waveform events.

DAI Interrupt Mask Priority Register (DAI_IMASK_PRI). Masks interrupt for DAI high or DAI low interrupt priority.

DAI Interrupt Latch Registers (DAI_IRPTL_L/H). Latches interrupt for the DAI high or DAI low interrupt.

DPI Interrupt Mask Registers (DPI_IMASK_RE/FE). Masks interrupts for rising and/or falling edge waveform events.

DPI Interrupt Latch Registers (DPI_IRPTL). Latches interrupt for DPI interrupt.

Functional Description

The following sections provide information on function of the interrupt controller.

Programmable Interrupt Priority Control

The processor core supports 19 programmable prioritized interrupts, which are shown in an example routing in Figure 2-1. The highest priority interrupt is P0I while the lowest priority is P18I. Any peripheral interrupt output may be connected to any programmable priority interrupt input.

All peripheral interrupt output signals are considered as source signals. The 19 prioritized peripheral interrupts (P0I–P18I) of the core are considered destination interrupts. The `PICR` register controls the connectivity between the source and destination.

The interrupt output of every peripheral can be programmed to connect to any one of the 19 peripheral interrupts. Moreover, the peripherals are grouped in two broad categories—DAI or DPI, each having its own interrupt controller. These interrupt controllers program the polarity, priority and the destination of each peripheral interrupt output. Therefore, all peripheral interrupts can also be connected to the core as DAI or DPI interrupts.

The PICR controls all peripheral's interrupts including DAI or DPI unit.

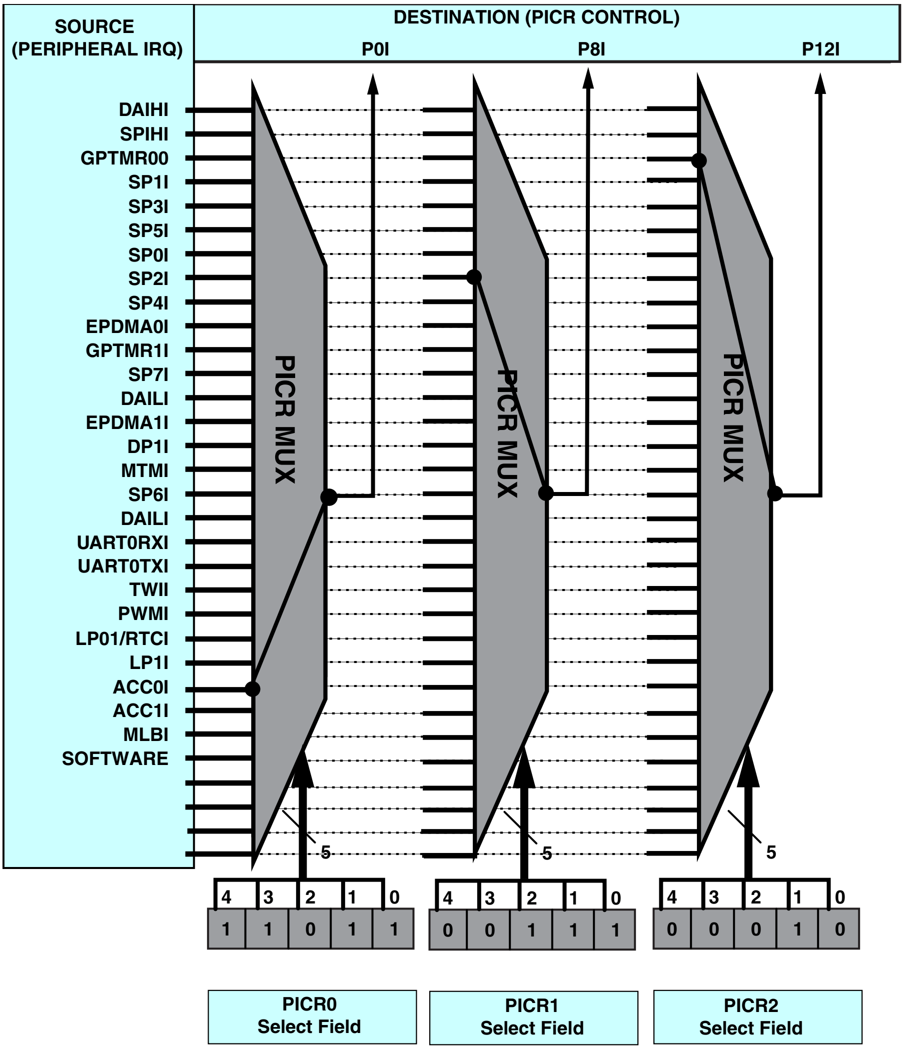

Figure 2-1. Programmable Prioritized Interrupts

Peripheral Interrupt

All input field encodings from Table A-6 on page A-14 assign a peripheral to trigger an interrupt as shown in the example below.

```
ustat1=dm(PICR2);
bit set ustat1 P17I4|P17I3;
bit clr ustat1 P17I2|P17I1|P17I0;
dm(PICR2)=ustat1;          /* write 11000 to route ACC1I to P17I */
```

Software Interrupt

Using the selection code 11111 (High) in Table A-6 on page A-14 allows programs to use software based interrupts (see example below). Unlike the core (four software interrupts) these software interrupts can be changed in priority.

```
ustat1=dm(PICR0);
bit set ustat1 P2I4|P2I3|P2I2|P2I1|P2I0;
dm(PICR0)=ustat1;         /* write 11111 to route SWI to P2I */
......
P2I_ISR:
ustat1=dm(PICR0);
bit clr ustat1 P2I4|P2I3|P2I2|P2I1|P2I0;
dm(PICR0)=ustat1;         /* clear 00000 for P2I acknowledge */
rti;
```

Peripherals with Multiple Interrupt Request Signals

The TWI and the UART have separate interrupt outputs. Both peripherals are already connected via the P14I (DPI) by default. However both peripherals allow separate connectivity into the PICR that are not routed by default. This provides more flexibility for priority change across the DAI/DPI interrupts.

System Interrupt Controller

The DAI and DPI modules each incorporate a system interrupt controller (SIC) which is connected to the core interrupt controller as seen in Figure 2-2.

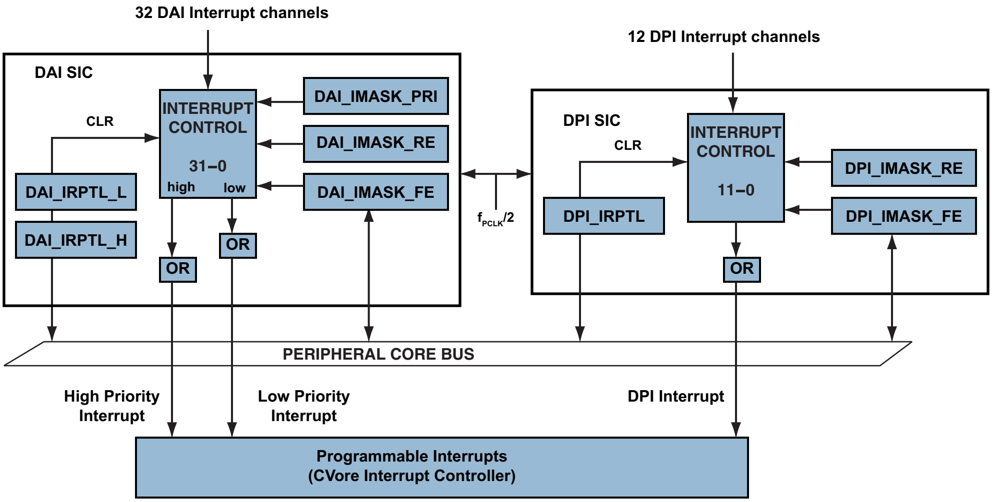

Figure 2-2. DAI/DPI System Interrupt Controllers (SIC)

The DAI/DPI contain their own system interrupt controllers that indicate to the core when DAI/DPI audio peripheral-related events have occurred. Since audio events generally occur infrequently relative to the SHARC core, the DAI/DPI interrupt controller reduces all of its interrupts onto three interrupt signals within the core's primary interrupt systems—one mapped with DAI low priority, one mapped with DAI high priority and the third mapped into the DPI interrupt. This allows programs to broadly indicate priority. In this way the DAI SIC provides 32 and the DPI SIC

12 independently configurable sources/channels. The output bus interrupt signals are logically ORed into one interrupt line and fed to the core's interrupt controller logic.

The DAI/DPI interrupt controllers have the same interrupt latency as the core interrupt controller, or 6 cycles of latency to respond to asynchronous interrupts.

Three registers are used to configure the DAI interrupt controller. Each of the 32 interrupt sources can be independently configured to trigger on an incoming signal's rising edge, falling edge, both edges, or neither edge.

Two registers are used to configure the DPI interrupt controller. Each of the 12 interrupt sources can be independently configured to trigger on an incoming signal's rising edge, falling edge, both edges, or neither edge. Note that all DAI/DPI interrupt control registers are memory mapped registers and are accessed via the peripheral bus while the core interrupt registers are system registers. For more information on core interrupts, see the processor programming reference manual.

DAI/DPI Interrupt Sources

The DAI's five peripheral sources are multiplexed into 32 interrupt sources and are labeled `DAI_INT31-0`. The DPI's three peripheral sources are multiplexed into 12 interrupt sources and are labeled `DPI_INT13-0` (Table 2-2).

There are two naming conventions. The DAI/DPI interrupt controller register bits are labeled `DAI_31-0_INT`/`DPI_13-0_INT` (def214xx.h file). Their corresponding SRU routing signals are labeled `DAI_INT_31-0_I`/`DPI_INT_13-0_I` (sru214xx.h file).

Table 2-2. Overview of DAI/DPI Interrupt Sources

Interrupt Source	Description	Signal Response
DAI_INT7–0	S/PDIF RX, 8 channels	Event
DAI_INT9–8	IDP Buffer, 2 channels	
DAI_INT17–10	IDP DMA, 8 channels	
DAI_INT7–0	S/PDIF RX, 8 channels	Waveform
DAI_INT21–18	ASRC, 4 channels	
DAI_INT31–22	Miscellaneous, S/PDIF TX, 10 channels	
DPI_INT2, 0	UARTRX/TX, 2 DMA channels	Event
DPI_INT4	TWI, 1 channel	
DPI_INT13–5	Miscellaneous, 9 channels	Waveform

DAI Interrupt Latch Priority Option

The DAI system interrupt controller register pair (DAI_IRPTL_H and DAI_IRPTL_L) replace functions normally performed by the core interrupt controller's IRPTL register. A single register (DAI_IRPTL_PRI) specifies to which latch these interrupts are mapped.

When a DAI interrupt is configured as low priority (DAI_IMASK_PRI bit cleared, default setting), it is latched in the DAI_IRPTL_L register. The low priority DAI interrupt, DAILI, is connected to the P12I core interrupt by default. The PICR register can alter this connection. Whenever a DAI low priority interrupt is set, the programmed DAILI bit in LIRPTL register sets, and the core services that low priority interrupt.

When a DAI interrupt is configured as high priority (DAI_IMASK_PRI bit set), it is latched in the DAI_IRPTL_H register. The high priority DAI interrupt, DAIHI, is connected to the P0I core interrupt by default. The PICR register can alter this connection. Whenever a DAI high priority interrupt is set, the programmed DAIHI bit in LIRPTL register sets, and the core services that interrupt with high priority.

The DAI triggers two interrupts in the IVT, one each for low or high priority. When any interrupt from the DAI needs to be serviced, one of the two core ISRs must interrogate the DAI's interrupt controller to determine the source(s).

DPI Interrupt Latch

The DPI SIC register (DPI_IRPTL) replaces functions normally performed by the core interrupt controller's IRPTL register.

When a DPI interrupt is configured, it is latched in the DPI_IRPTL register. The DPI interrupt is connected to the P14I core interrupt by default. The PICR register can alter this connection. Whenever a DPI interrupt is set, the programmed DPI bit in LIRPTL register sets and the core services that interrupt with the programmed priority.

DAI/DPI Interrupt Mask for Waveforms

Two registers (DAI_IMASK_RE and DAI_MASK_FE) replace the core interrupt controller's version of the IMASK register. As with the IMASK register, these DAI registers provide a way to specify which interrupts to acknowledge and handle, and which interrupts to ignore. These dual registers function as IMASK does, but with a higher degree of granularity.

Use of the DAI_IMASK_RE/DAI_IMASK_FE registers or the DPI_IMASK_RE/DPI_IMASK_FE registers allows programs to acknowledge and respond to rising edges, falling edges, both rising and falling edges, or neither rising nor falling edges so they can be masked separately.

Signals from the SRU can be used to generate interrupts. For example, when the DAI_30_INT bit of DAI_IMASK_FE register is set to one, any falling edge signals from the external channel generate an interrupt in the core and the interrupt latch is set. A read of the MASK register does not clear the IRPTL register.

DAI/DPI Interrupt Mask for Events

The system interrupt controller needs information about a peripheral's interrupt sources that correspond to event signals (refer to Table 2-2 on page 2-8). As a result, the rising edge is used as an interrupt source only. For DAI/DPI peripherals marked as events, programs may unmask an interrupt source on the rising edge only.

DAI/DPI Interrupt Service

The interrupt acknowledge operates differently when multiple channels are multiplexed into one interrupt output signal. When an interrupt from the DAI/DPI must be serviced, any of the three interrupt service routines (DAILI, DAIHI and DPII) must query the RIC to determine the source(s). Sources can be any one or more of the DAI channels (`DAI_INT31-0`) or DPI channels (`DPI_INT13-0`).

- When `DAI_IRPTL_H` is read, the high priority latched interrupts are cleared.
- When `DAI_IRPTL_L` is read, the low priority latched interrupts are cleared.
- When `DPI_IRPTL` is read, the latched interrupts are cleared.
- The IDP_FIFO_GTN_INT interrupt is not cleared when the `DAI_IRPTL_H`/L registers are read. This interrupt is cleared automatically when the situation that caused the interrupt goes away.
- A shadow register, `DPI_IRPTL_SH`, is provided for the primary register `DPI_IRPTL`. Reads of this register returns the data in the `DPI_IRPTL` register without clearing the contents of the register.

If an interrupt occurs in the same cycle as a latch register is cleared, the clear mechanism has lower priority and the new interrupt is registered.

The TWII and UARTRXI interrupts do not follow this rule. Acknowledge occurs in these peripheral's latch register.

Reading the interrupt latch registers (DAI_IRPTL_x/DPI_IRPTL) clears the interrupts (read-only-to-clear bit type). Therefore, the ISR must service all of the interrupt sources it discovers. That is, if multiple interrupts are latched in multiple mask registers, all of them must be serviced before executing an RTI instruction. Otherwise the condition is not cleared. For more information, see "Programmable Interrupt Priority Control Registers" on page A-14.

Interrupt Service

This section describes how the interrupt service routines operate to clear interrupt requests correctly.

Interrupt driven I/O is advantageous because the core does not need to poll input signal. (For more information, see the "Interrupts" section in each peripheral chapter.) When an interrupt is triggered, the sequencer typically finishes the current instruction and jumps to the IVT (interrupt vector table). From the IVT the address then typically vectors to the ISR routine. The sequencer jumps into this routine, performs program execution and then exits the routine by executing the RTI (return from interrupt) instruction. However this rule does not apply for all cases and is discussed below.

There are three interrupt acknowledge mechanisms used in an ISR routine and they depend on the peripheral:

- RTI instruction
- Read-only to-clear (ROC) status bit + RTI instruction
- Write-1-to clear (W1C) status bit + RTI instruction

The DAI/DPI interrupt controllers are designed such that in order to terminate correctly, the latch register must be read to identify the source. Note that this read automatically acknowledges the request before exiting an interrupt routine. For the W1C mechanism, programs must write into the specific bit of the latch register in order to terminate the interrupt properly.

If the acknowledge mechanism rules are not followed correctly, unwanted and sporadic interrupts will occur.

Core Buffer Service Request (I/O mode)

If the data stream peripherals access its data buffer of the respective DMA FIFO through the core, the buffer status plays a significant role in acknowledging the interrupt. If, for example, a receive buffer is full, an interrupt is generated and the buffer is read in the ISR, automatically clearing the request (ROC + RTI). Similarly, if a transmit buffer is empty, an interrupt is generated and the write clears the request (WOC + RTI).

DMA Access

If the peripherals access the buffer by DMA, the logic operates differently. In DMA, the buffer status has no effect on interrupts. Rather, the DMA count register generates an interrupt whenever it reaches zero. The acknowledge mechanisms may vary by the peripheral used.

Interrupt Latency

Good programming requires that an interrupt service acknowledge an interrupt request back to the peripheral as early as possible. This response allows the peripheral to sense additional events as quickly as possible.

The service routine must ensure that the requests are released before the RTI instruction executes. Otherwise, the service routine is invoked immediately after the execution of the RTI instruction. Some interrupt requests are cleared by write-one-to-clear (W1C) operations. This write command does not stall the core, rather it is automatically latched in a write buffer and synchronized with the peripheral clock domain (PCLK) before it is sent to the peripheral bus.

This process may require multiple CCLK cycles before the W1C operation arrives at the peripheral. If the W1C operation executes at the end of a service routine, a dummy read should be executed over the peripheral bus before the RTI instruction to ensure that the peripheral releases the request before the RTI executes. The following describe cases for interrupt latency.

- For peripherals with W1C acknowledge mechanisms a write into the peripheral's status register to clear the interrupt causes a certain amount of latency (because of register write effect latency).
- Interrupt-driven data transfers (core or DMA) from any peripheral that generates interrupts and which uses an ISR routine, a write into a peripheral data buffer (to clear the interrupt) or a control register causes a certain amount of latency (due to the existence of register write effect latency and buffer clock domains).

In both cases, if for example the program comes out of the interrupt service routine (RTI instruction) during that period of latency (maximum of 10 CCLK cycles), the interrupt is generated again. To avoid interrupt regeneration, use one of the following solutions.

The interrupt regeneration restriction does not apply to any SPORT in DMA operation mode.

1. Read an IOP register from the same peripheral block before the return from interrupt (RTI). The read forces the write to occur as shown in the example below.

```
ISR_SPI_Routine:
R0 = dm(i0,m0);
dm(TXSPI) = R0;     /* write to SPI data buffer */
R0 = dm(SPICTL);    /* this dummy read forces the previous write
                       to complete */
rti;

ISR_PWM_Routine:
r1=PWM_STAT3;
dm(PWMGSTAT)=r1;    /* W1C to PWM status reg */
r0=dm(PWMGSTAT);    /* this dummy read forces the previous write
                       to complete */
rti;
```

2. Add sufficient NOP instructions after a write. In the worst case, programs need to add ten NOP instructions after a write, as shown in the example code below.

```
ISR_Routine:
R0 = 0x0;
dm(SPICTL) = R0;          /* or disable SPI control */
nop; nop; nop; nop; nop;
nop; nop; nop; nop; nop;
rti;
```

DMA Completion Types

On SHARC processors, interrupts are generated after *internal transfer completion* (when the DMA count register has expired). However, in some cases the transfer may not have completed (due to different channel priorities) and valid data still resides in the peripheral's buffer, waiting to be transmitted. To overcome this problem, the interrupt *access completion* mode is introduced. In this mode the interrupt is generated when the last data has left the buffer. This option is available for the SPORT, SPI, link port and external port DMA. For details, refer to the specific peripheral's chapter.

Debug Features

This section describes the shadow registers used with the IDP, S/PDIF, ASRC, UART, TWI and DAI/DPI

Shadow Interrupt Register

The DAI/DPI interrupt controller has shadow registers to simplify debug activities since these registers do not manipulate status control. Any read of the DAI_IRPTL_x_SH or DPI_IRPTL_SH shadow registers provides the same data as a read of the DAI_IRPTL_x or DPI_IRPTL registers. However reads of the DAI/DPI shadow registers don't change the interrupt acknowledge status to the core interrupt controller.

3 I/O PROCESSOR

In applications that use extensive off-chip data I/O, programs may find it beneficial to use a processor resource other than the processor core to perform data transfers. The ADSP-214xx processors contain an I/O processor (IOP) that supports a variety of DMA (direct memory access) operations. Each DMA operation transfers an entire block of data. These operations include the transfer types shown in Table 3-1 and the list that follows.

Table 3-1. I/O Processor Specifications

Feature	Availability
Total DMA channels	See product-specific data sheet
Rotating DMA channel priority	Yes
Media Local Bus (MLB)	31
SPORT DMA channels	16
IDP DMA channels	8
UART DMA channels	2
FIR/FFT/IIR DMA channels	2
SPI DMA channels	2
MTM/DTCP DMA channels	2
External Port DMA channels	2
PDAP DMA channel	1
DMA channel interrupts	16
Clock Operation	f_{PCLK}

Features

The I/O processor features are briefly described in the following list.

- Two independent DMA buses (peripheral and external port DMA bus)
- Both buses have high priority over the core for internal memory access
- DMA transfer types for standard, chained and ping-pong (IDP)
- DMA channel interrupt priority programmable (PICR registers)
- Channel arbitration fixed or rotated
- SPORT DMA support chain insertion mode (Changing TCB list during runtime)
- External port DMA supports direction on the fly
- DMA transaction can be paused by clearing the DMA enable bit
- DMA can be halt during single step for debug

The I/O processor supports the following DMA transaction types.

- Internal memory ← IDP (DAI) unidirectional
- Internal memory ↔ SPORT (DAI)
- External memory ↔ SPORT (DAI)
- Internal memory ↔ SPI
- Internal memory ↔ Link port
- Internal memory ↔ MLB
- Internal memory ↔ UART

- Internal memory ↔ Accelerator
- Internal memory ↔ External memory (External port)
- Internal memory ↔ Internal memory (MTM, External port)

By managing DMA, the I/O processor frees the processor core, allowing it to perform other operations while off-chip data I/O occurs as a background task. The multi-bank architecture of the ADSP-214xx internal memory allows the core and IOP to simultaneously access the internal memory if the accesses are to different memory banks. This means that DMA transfers to internal memory do not impact core performance. The processor core continues to perform computations without penalty.

To further increase off-chip I/O, multiple DMAs can occur at the same time. The IOP accomplishes this by managing multiple DMAs of processor memory through the different peripherals. Each DMA is referred to as a *channel* and each channel is configured independently.

Register Overview

Two global IOP registers control the DMA arbitration over the I/O buses—the first for the peripheral bus and the second for the external port bus. This section provides brief descriptions of the major IOP registers. For complete information, see "Register Listing" on page B-1.

System Control Register (SYSCTL). Controls the peripheral DMA operation for fixed or rotating DMA channel arbitration.

External Port Control Register (EPCTL). Controls the external port bus arbitration between SPORT, EPDMA and core access.

DMA Channel Registers

The following sections provide information on the registers that control all DMA operations for each peripheral. Additional information on DMA operations can be found in specific peripheral chapters.

DMA Channel Allocation

Each channel has a set of parameter registers which are used to set up DMA transfers. Table 3-29 on page 3-39 shows the DMA channel allocation and parameter register assignments for the ADSP-214xx processors.

DMA channels vary by processor model. For a breakdown of DMA channels for a particular model, see the product-specific data sheet. Also note that each DMA channel has a specific peripheral assigned to it.

Standard DMA Parameter Registers

The parameter registers described below control the source and destination of the data, the size of the data buffer, and the step size used.

The length of DMA registers for the serial ports have changed from earlier SHARC processors in order to accommodate data transfers to/from external memory.

Index registers. These registers, shown in Table 3-2, provide an internal memory address that acts as a pointer to the next internal memory DMA read or write location. All internal index registers have 18-bit address width. However all index registers are based on an internal memory offset of 0x80000 (bit 19 set) so the total width results to 19 bits. This internal memory offset is not applicable for the index registers that correspond to SPORT DMAs as these registers are 28 bits.

Table 3-2. Index Registers

Register Name	Width (Bits)	Description
IISP0–7A	28	SPORTxA (supports external addresses)
IISP0–7B	28	SPORTxB (supports external addresses)
IISPI	19	SPI
IISPIB	19	SPIB
IDP_DMA_I0–7	19	IDPx
IDP_DMA_I0–7A	19	IDPx index A (ping-pong)
IDP_DMA_I0–7B	19	IDPx index B (ping-pong)
IIUART0RX	19	UART0 Receiver
IIUART0TX	19	UART0 Transmitter
IILB0–1	19	Link Port0–1
IIFIR	19	Accelerator FIR data input
CIFIR	19	Accelerator FIR coeff input
OIFIR	19	Accelerator FIR output
IIIIR	19	Accelerator IIR data input
CIIIR	19	Accelerator IIR coeff input
OIIIR	19	Accelerator IIR output
IIFFT	19	Accelerator FFT input
OIFFT	19	Accelerator FFT output
IIMTMW	19	MTM Write
IIMTMR	19	MTM Read
IIEP0–1	19	External Port
EIEP0–1	28	External Port (external)

Modify registers. These registers, shown in Table 3-3, provide the signed increment by which the DMA controller post-modifies the corresponding memory index register after the DMA read or write.

Table 3-3. Modify Registers

Register Name	Width (Bits)	Description
IMSP0–7A	16	SPORTA
IMSP0–7B	16	SPORTB
IMSPI	16	SPI
IMSPIB	16	SPIB
IDP_DMA_M0–7	6	IDP
IDP_DMA_M0–7A	6	IDP modify A (ping-pong)
IDP_DMA_M0–7B	6	IDP modify B (ping-pong)
IMLB0–1	16	Link Port
IMUART0RX	16	UART0 Receiver
IMUART0TX	16	UART0 Transmitter
IMFIR	16	Accelerator FIR data input
CMFIR	16	Accelerator FIR coeff input
OMFIR	16	Accelerator FIR output
IMIIR	16	Accelerator IIR data input
CMIIR	16	Accelerator IIR coeff input
OMIIR	16	Accelerator IIR output
IMFFT	16	Accelerator FFT input
OMFFT	16	Accelerator FFT output
IMMTMW	16	MTM Write
IMMTMR	16	MTM Read
IMEP0–1	16	External Port
EMEP0–1	27	External Port (external)

Count registers. These registers, shown in Table 3-4, indicate the number of words remaining to be transferred to or from memory on the corresponding DMA channel.

Table 3-4. Count Registers

Register Name	Width (Bits)	Description
CSP0–7A	16	SPORTA
CSP0–7B	16	SPORTB
ICSPI	16	SPI
ICSPIB	16	SPIB
IDP_DMA_C0–7	16	IDP
ICLB0–1	16	Link Port
CUART0RX	16	UART0 Receiver
CUART0TX	16	UART0 Transmitter
ICFIR	16	Accelerator FIR data input
CCFIR	16	Accelerator FIR coeff input
OCFIR	16	Accelerator FIR output
ICIIR	16	Accelerator IIR data input
CCIIR	16	Accelerator IIR coeff input
OCIIR	16	Accelerator IIR output
ICFFT	16	Accelerator FFT input
OCFFT	16	Accelerator FFT output
ICMTMW	16	MTM Write
ICMTMR	16	MTM Read
ICEP0–1	16	External Port
ECEP0–1	16	External Port (external)

Chain pointer registers. These registers, shown in Table 3-5, hold the starting address of the transfer control block (TCB parameter register values) for the next DMA operation on the corresponding channel. These registers also control whether the I/O processor generates an interrupt when the current DMA process ends.

For information on transfer control blocks (TCBs), see **"Chained DMA" on page 3-32**.

Table 3-5. Chain Pointer Registers

Register Name	Width (Bits)	Description
CPSP0–7A	28	SPORTA
CPSP0–7B	28	SPORTB
CPSPI	20	SPI
CPSPIB	20	SPIB
CPLB0–1	20	Link Port
CPUART0RX	20	UART0 Receiver
CPUART0TX	20	UART0 Transmitter
CPFIR	20	Accelerator FIR
CPIIR	20	Accelerator IIR
CPIFFT	21	Accelerator FFT input
CPOFFT	20	Accelerator FFT output
CPEP0–1	21	External Port

Extended DMA Parameter Registers

This section describes the enhanced parameter registers used for the accelerators and the external port.

Base registers. These registers, shown in Table 3-6, indicate the start address of the circular buffer to be transferred to/from memory on the corresponding DMA channel. All internal base registers have 18-bit address width. However all index registers are based on an internal memory offset of 0x80000 (bit 19 set) so the total width is 19 bits.

Table 3-6. Base Registers

Register Name	Width (Bits)	Description
IBFIR	19	Accelerator FIR input
OBFIR	19	Accelerator FIR output
IBIIR	19	Accelerator IIR input
OBIIR	19	Accelerator IIR output
IBFFT	19	Accelerator FFT input
OBFFT	19	Accelerator FFT output
EBEP0–1	28	External Port (external)

Length registers. These registers, shown in Table 3-7, define the length of the circular buffer to be transferred to/from memory on the corresponding DMA channel.

Table 3-7. Length Registers

Register Name	Width (Bits)	Description
ILFIR	19	Accelerator FIR input
OLFIR	19	Accelerator FIR output
ILIIR	19	Accelerator IIR input
OLIIR	19	Accelerator IIR output
ILFFT	19	Accelerator FFT input
OLFFT	19	Accelerator FFT output
ELEP0–1	26	External Port (external)

Miscellaneous External Port Parameter registers. These registers, shown in Table 3-8, are used for the delay line and scatter/gather DMA. They read from tap list buffers, store counters and index pointers.

Table 3-8. Miscellaneous External Port Parameter Registers

Register Name	Width (Bits)	Description
RCEP[1]	16	Delay line DMA read block size
RIEP[1]	19	Delay line DMA read internal index
RMEP[1]	27	Delay line DMA read external modifier
TCEP	16	Delay line/tap list DMA tap list count
TPEP	19	Delay line/tap list DMA tap list pointer

1 These registers are only accessible through the TCB loading.

MLB Parameter registers. The MLB interface does not have modify and count parameter registers like the other peripherals. Instead it has base and end address registers which implicitly define the DMA length. For more information, see Chapter 9, Media Local Bus.

Data Buffers

The data buffers or FIFOs (shown in Table 3-9) are used by each DMA channel to store data during the priority arbitration time period. The buffers (depending on the peripheral) are accessed by both DMA and the core.

Table 3-9. Data Buffers

Buffer Name	Total FIFO Depth	Description
TXSP0–7A	1 + 1	SPORTA Transmit (RW) + Shift Register
TXSP0–7B	1 + 1	SPORTB Transmit (RW) + Shift Register
RXSP0–7A	1 + 1	SPORTA Receive (RW) + Shift Register

Table 3-9. Data Buffers (Cont'd)

Buffer Name	Total FIFO Depth	Description
RXSP0–7B	1 + 1	SPORTB Receive (RW) + Shift Register
TXSPI	1 + 1	SPI Transmit (RW) + Shift Register
TXSPIB	1 + 1	SPIB Transmit (RW) + Shift Register
RXSPI	1 + 1	SPI Receive (RO) + Shift Register
RXSPIB	1 + 1	SPIB Receive (RO) + Shift Register
RXSPI_SHADOW	1	SPI Receive Shadow (RO)
RXSPIB_SHADOW	1	SPIB Receive Shadow (RO)
SPI DMA	4	DMA only
SPIB DMA	4	DMA only
IDP_FIFO	8	IDP FIFO Receive (RW)
TXLB0–1	2 + 1	Link Port Transmit Buffer (RW)
TXLB0–1_IN_SHADOW	1	Link Port Transmit Shadow Pack Register (RO)
TXLB0–1_OUT_SHADOW	1	Link Port Transmit Shadow Pack Register (RO)
RXLB0–1	2 + 1	Link Port Receive Buffer (RW)
RXLB0–1_IN_SHADOW	1	Link Port Receive Shadow Pack Register (RO)
RXLB0–1_OUT_SHADOW	1	Link Port Receive Shadow Pack Register (RO)
UARTRBR0	1 + 1	UART0 Receiver (RO) + Shift Register
UARTTHR0	1 + 1	UART0 Transmitter (WO) + Shift Register
Accelerator FFT input	8	Buffer for FFT only
Accelerator FFT output	8	Buffer for FFT only
MTM read/write	2	Internal DMA only
DFEP0–1	6	External Port DMA only
AMIRX	1	AMI Receive Packer
AMITX	1	AMI Transmit Packer
TXTWI8	1 + 1	TWI Transmit (WO) + Shift Register

Table 3-9. Data Buffers (Cont'd)

Buffer Name	Total FIFO Depth	Description
TXTWI16	2 + 1	TWI Transmit (WO) + Shift Register
RXTWI8	1 + 1	TWI Receive (RO) + Shift Register
RXTWI16	2 + 1	TWI Receiver (RO) + Shift Register
MLB_CCBCR	124 x 36	RX Buffer (RW) (I/O mode)
MLB_CNBCR	124 x 36	TX Buffer (RW) (I/O mode)

Some data buffers provide debug support to enable the buffer hang disable (BHD) bit. This feature can be enabled in the dedicated peripheral control register for the IDP, SPORT, link port, UART0 and the TWI.

Chain Pointer Registers

The chain pointer registers, described inTable 3-10, Table 3-11 (generic), Table 3-12 (SPORTs), Table 3-13 (external port) and Table 3-14 (FFT) are 20 bits wide. The lower 19 bits are the memory address field. Like other I/O processor address registers, the chain pointer register's value is offset to match the starting address of the processor's internal memory before it is used by the I/O processor. On the SHARC processor, this offset value is 0x80000.

Table 3-10. Chain Pointer Registers

Register	Width	Description
CPEP0–1	21	External Port
CPSP0–7A	29	SPORT A Channels
CPSP0–7B	29	SPORT B Channels
CPSPI	20	SPI
CPSPIB	20	SPIB
CPUART0RX	20	UART Receive

Table 3-10. Chain Pointer Registers (Cont'd)

Register	Width	Description
CPUART0TX	20	UART Transmit
CPLB0-1	20	Link Port
CPFIR	19	FIR
CPIIR	19	IIR
CPIFFT	21	Input FFT
CPOFFT	20	Output FFT

Table 3-11. Chain Pointer Register Bit Descriptions (CPx)

Bit	Name	Description
18–0	IIx address	Next chain pointer address
19	PCI	Program controlled interrupt 0 = no interrupt after current TCB 1 = interrupt after current TCB

For the new SPORT external memory functionality, when writing tests which involve the `PCI` bit, the external memory address should be split before writing to the chain pointer register.

Table 3-12. SPORT Chain Pointer Register Bit Descriptions (CPSPx)

Bit	Name	Description
18–0	IIx address	Next chain pointer address (bits 18–0 of the chain pointer)
19	PCI	Program controlled interrupt 0 = no interrupt after current TCB 1 = interrupt after current TCB
28–20	IIx address	Next chain pointer (external address, bits 27–19 of the chain pointer)

Note that the serial ports have the ability to fetch TCBs from external memory.

Table 3-13. External Port Chain Pointer Register Bit Descriptions (CPEPx)

Bit	Name	Description
18–0	IIx address	Next chain pointer address
19	PCI	Program controlled interrupt 0 = no interrupt after current TCB 1 = interrupt after current TCB
20	CPDR	DMA direction for next TCB 0 = write to internal memory 1 = read from internal memory

Table 3-14. FFT Input Chain Pointer Register Bit Descriptions (CPIFFT)

Bit	Name	Description
18–0	IIx address	Next chain pointer address
19	PCI	Program controlled interrupt 0 = no interrupt after current TCB 1 = interrupt after current TCB
20	COEFFSEL	Coefficient select for next TCB 0 = next TCB is data TCB 1 = next TCB is coeff TCB

Bit 19 of the chain pointer register is the program controlled interrupt (PCI) bit. This bit controls whether an interrupt is latched after every DMA in the chain (when set = 1), or whether the interrupt is latched after the entire DMA sequence completes (if cleared = 0). If a program contains a single chained DMA then the PCI interrupt is generated coincident with the start of next TCB loading.

However, if running multiple DMA channels this coincidence is no longer true since there are different DMA channel priorities versus interrupt priorities.

The PCI bit only effects DMA channels that have chaining enabled. Also, interrupt requests enabled by the PCI bit are maskable with the IMASK register.

TCB Storage

This section lists all the different TCB memory allocations used for DMA chaining on the peripherals. Note that all TCBs must be located in internal memory except SPORTs, where TCBs can exist in external memory.

Serial Port TCB

The serial ports support single and chained DMA. Table 3-15 shows the required TCBs for chained DMA

Table 3-15. SPORT TCBs

Address	Register
CP[27:0]	CPSPx Chain Pointer
CP[27:0] + 0x1	ICSPx Internal Count
CP[27:0] + 0x2	IMSPx Internal Modifier
CP[27:0] + 0x3	IISPx Internal/External Index

SPI TCB

The serial peripheral interfaces supports both single and chained DMA. However, unlike the serial ports, programs cannot insert a TCB in an active chain. Table 3-16 shows the required TCBs for chained DMA.

Table 3-16. SPI/SPIB TCBs

Address	Register
CP[18:0]	CPSPI/B Chain Pointer
CP[18:0] + 0x1	ICSPI/B Internal Count
CP[18:0] + 0x2	IMSPI/B Internal Modifier
CP[18:0] + 0x3	IISPI/B Internal Index

UART TCB

The UART interface supports both single and chained DMA. However, unlike the serial ports, programs cannot insert a TCB in an active chain. Table 3-17 shows the required TCBs for chained DMA.

Table 3-17. UART0 TCBs

Address	Register
CP[18:0]	RXCP_UAC0/TXCP_UAC0 Chain Pointer
CP[18:0] + 0x1	RXC_UAC0/TXC_UAC0 Internal Count
CP[18:0] + 0x2	RXM_UAC0/TXM_UAC0 Internal Modifier
CP[18:0] + 0x3	RXI_UAC0/TXI_UAC0 Internal Index

Link Port TCB

The link port interface supports both single and chained DMA. Table 3-18 shows the required TCBs for chained DMA.

Table 3-18. Link Port TCBs

Address	Register
CP[18:0]	CPLPx Chain Pointer
CP[18:0] + 0x1	CLBx Internal Count

Table 3-18. Link Port TCBs (Cont'd)

Address	Register
CP[18:0] + 0x2	IMLBx Internal Modifier
CP[18:0] + 0x3	IILBx Internal Index

FIR Accelerator TCB

The FIR accelerator DMA supports circular buffer chained DMA. Table 3-19 shows the required TCBs for chained DMA. The FIR accelerator does not support circular buffering for the coefficient buffer.

Table 3-19. FIR TCBs

Address	Register
CP[18:0]	CPFIR
CP[18:0] + 0x1	CCFIR
CP[18:0] + 0x2	CMFIR
CP[18:0] + 0x3	CIFIR
CP[18:0] + 0x4	OBFIR
CP[18:0] + 0x5	OCFIR
CP[18:0] + 0x6	OMFIR
CP[18:0] + 0x7	OIFIR
CP[18:0] + 0x8	IBFIR
CP[18:0] + 0x9	ICFIR
CP[18:0] + 0xA	IMFIR
CP[18:0] + 0xB	IIFIR
CP[18:0] + 0xC	FIRCTL2

The `CCFIR` register is loaded with the values in the `CCFIR` TCB field and is decremented from that value onwards. However, coefficient loading continues until the number of coefficients, equal to the tap length, are read. This is true even if the CCFIR register reaches zero as in the case of a tap length = 10, and the CCFIR field in the TCB is initialized to 0. The value in the CCFIR register is –10 after all coefficients are loaded.

IIR Accelerator TCB

The IIR accelerator supports circular buffer chained DMA. Table 3-20 shows the required TCBs for chained DMA.

In the IIR accelerator DMA, two different TCB loading sequences are available: one TCB loads five parameters for the coefficients (IIRCTL2, CIIIR, CMIIR, CCIIR and CPIIR). The second loads 10 parameters for the data (IIRCTL2, IIIR, IMIIR, ICIIR, IBIIR, OIIIR, OMIIR, OCIIR, OBIIR and CPIIR).

Table 3-20. IIR TCBs

Address	Register
CP[18:0]	CPIIR
CP[18:0] + 0x1	CCIIR
CP[18:0] + 0x2	CMIIR
CP[18:0] + 0x3	CIIIR
CP[18:0] + 0x4	OBIIR
CP[18:0] + 0x5	OCIIR
CP[18:0] + 0x6	OMIIR
CP[18:0] + 0x7	OIIIR
CP[18:0] + 0x8	IBIIR
CP[18:0] + 0x9	ICIIR
CP[18:0] + 0xA	IMIIR
CP[18:0] + 0xB	IIIIR
CP[18:0] + 0xC	IIRCTL2

FFT Accelerator TCB

The FFT accelerator supports circular buffer chained DMA. Table 3-21 and Table 3-22 shows the required TCBs for chained DMA.

Table 3-21. FFT Input TCBs

Address	Register
CP[18:0]	CPIFFT
CP[18:0] + 0x1	IBFFT
CP[18:0] + 0x2	ILFFT
CP[18:0] + 0x3	ICFFT
CP[18:0] + 0x4	IMFFT
CP[18:0] + 0x5	IIFFT

The input TCB controls both data and coefficients. Bit 20 (COEFFSEL) of the input chain pointer register (CPIFFT), indicates whether the TCB is for loading data or coefficients. For coefficient TCBs (COEFFSEL=1), circular buffering and the input length (ILFFT) and base length (IBFFT) TCB fields are ignored.

Table 3-22. FFT Output TCBs

Address	Register
CP[18:0]	CPOFFT
CP[18:0] + 0x1	OBFFT
CP[18:0] + 0x2	OLFFT
CP[18:0] + 0x3	OCFFT
CP[18:0] + 0x4	OMFFT
CP[18:0] + 0x5	OIFFT

External Port TCB

The external port interface supports many different types of DMA, resulting in different lengths of TCBs. The TCB size varies from six locations (chained DMA) to 13 locations (delay line DMA). Table 3-23 shows the required TCBs for chained DMA.

Table 3-23. External Port TCBs

Address	Register
CP[18:0]	CPEP
CP[18:0] + 0x1	EMEP
CP[18:0] + 0x2	EIEP
CP[18:0] + 0x3	ICEP
CP[18:0] + 0x4	IMEP
CP[18:0] + 0x5	IIEP

The order the descriptors are fetched with circular buffering enabled is shown in Table 3-24.

Table 3-24. External Port TCBs for Circular DMA

Address	Register
CP[18:0]	CPEP
CP[18:0] + 0x1	ELEP
CP[18:0] + 0x2	EBEP
CP[18:0] + 0x3	EMEP
CP[18:0] + 0x4	EIEP
CP[18:0] + 0x5	ICEP
CP[18:0] + 0x6	IMEP
CP[18:0] + 0x7	IIEP

For delay line DMA, TCB loading is split into two sequences to improve overall priority. The first TCB loads the write parameters (IIEP–ELEP) and the second loads the read parameters (RIEP–CPEP). This two stage loading is transparent to the application. The order the descriptors are fetched with circular buffering enabled is shown in Table 3-25.

Table 3-25. External Port TCBs for Delay Line DMA

Address	Register
Delay Line Read	
CP[18:0]	CPEP
CP[18:0] + 0x1	TPEP
CP[18:0] + 0x2	TCEP
CP[18:0] + 0x3	RMEP
CP[18:0] + 0x4	RCEP
CP[18:0] + 0x5	RIEP
Delay Line Write	
CP[18:0] + 0x6	ELEP
CP[18:0] + 0x7	EBEP
CP[18:0] + 0x8	EMEP
CP[18:0] + 0x9	EIEP
CP[18:0] + 0xA	ICEP
CP[18:0] + 0xB	IMEP
CP[18:0] + 0xC	IIEP

The order the descriptors are fetched for scatter/gather DMA with circular buffering enabled is shown in Table 3-26 and Table 3-27.

Table 3-26. External Port TCBs for Scatter/Gather DMA

Address	Register
CP[18:0]	CPEP
CP[18:0] + 0x1	TPEP
CP[18:0] + 0x2	TCEP
CP[18:0] + 0x3	EMEP
CP[18:0] + 0x4	EIEP
CP[18:0] + 0x5	ICEP
CP[18:0] + 0x6	IMEP
CP[18:0] + 0x7	IIEP

Table 3-27. External Port TCBs for Circular Scatter/Gather DMA

Address	Register
CP[18:0]	CPEP
CP[18:0] + 0x1	ELEP
CP[18:0] + 0x2	EBEP
CP[18:0] + 0x3	TPEP
CP[18:0] + 0x4	TCEP
CP[18:0] + 0x5	EMEP
CP[18:0] + 0x6	EIEP
CP[18:0] + 0x7	ICEP
CP[18:0] + 0x8	IMEP
CP[18:0] + 0x9	IIEP

Clocking

The fundamental timing clock of the IOP is peripheral clock (PCLK). All DMA data transfers over the IO0 or IO1 buses are clocked at PCLK speed.

Functional Description

The following several sections provide detail on the function of the I/O processor.

Automated Data Transfer

Because the IOP registers are memory-mapped, the processors have access to program DMA operations. A program sets up a DMA channel by writing the transfer's parameters to the DMA parameter registers. After the index, modify, and count registers (among others) are loaded with a starting source or destination address, an address modifier, and a word count, the processor is ready to start the DMA.

The peripherals each have a DMA enable bit in their channel control registers. Setting this bit for a DMA channel with configured DMA parameters starts the DMA on that channel. If the parameters configure the channel to receive, the I/O processor transfers data words received at the buffer to the destination in internal memory. If the parameters configure the channel to transmit, the I/O processor transfers a word automatically from the source memory to the channel's buffer register. These transfers continue until the I/O processor transfers the selected number of words as determined by the count parameter. DMA through the IDP ports occurs in receive mode (into internal memory) only.

DMA Transfer Types

Standard DMA. A standard DMA (once it is configured) transfers data from location A to location B. An interrupt can be used to indicate the end of the transfer. To start a new DMA sequence after the current one is finished, a program must first clear the DMA enable bit (control register), write new parameters to the index, modify, and count registers (parameter registers), then set the DMA enable bit to re-enable DMA (control register).

An instance where standard DMA can be used is to copy data from a peripheral to internal memory for processor booting. With the help of the loader tool, the tag (header information) of the boot stream is decoded to get the storage information which includes the index, modify, and count of a specific array to start another standard DMA.

Chained DMA. Chained DMA sequences are a set of multiple DMA operations, each autoinitializing the next in line. To start a new DMA sequence after the current one is finished, the IOP automatically loads new index, modify, and count values from an internal memory location (or external memory location for DMA to external ports) pointed to by that channel's chain pointer register. Using chaining, programs can set up consecutive DMA operations and each operation can have different attributes.

Chained DMA with direction on the fly (External Port). The external port DMA controller supports chained DMA sequences with an additional feature that allows the port to change the data direction for each individual TCB. An additional bit in the TCB differentiates between a read or write operation.

The IDP port does not support DMA chaining.

Ping-pong DMA (IDP). In ping-pong DMA, the parameters have two memory index values (index A and index B), one count value and one modifier value. The DMA starts the transfer with the memory indexed by A. When the transfer is completed as per the value in the count register, the DMA restarts with the memory location indexed by B. The DMA restarts with index A after the transfer to memory with index B is completed as per the count value. This repeats until the DMA is stopped by resetting the DMA enable bit.

Circular Buffering DMA (FFT, FIR, IIR, External Port). This mode resembles the chained DMA mode, however two additional registers (base and length) are used. This mode performs DMA within the circular buffer, which is useful for filter implementation since core interaction is limited, conserving bandwidth.

DMA Direction

The IOP supports DMA in four directions. These are described in the following sections.

Internal to External Memory

DMA transfers between internal memory and external memory devices use the processor's external port. For these types of transfers, the application code provides the DMA controller with the internal memory buffer size, address, and address modifier, as well as the external memory buffer size, address, address modifier, and the direction of transfer. After setup, the DMA transfers begin when the program enables the channel and continues until the I/O processor transfers the entire buffer to processor memory. Table 3-29 on page 3-39 shows the parameter registers for each DMA channel.

Peripheral to Internal Memory

Similarly, DMA transfers between internal memory and serial, IDP, or SPI ports have DMA parameters. When the I/O processor performs DMA between internal memory and one of these ports, the program sets up the parameters, and the I/O uses the port instead of the external bus.

The direction (receive or transmit) of the peripheral determines the direction of data transfer. When the port receives data, the I/O processor automatically transfers the data to internal memory. When the port needs to transmit a word, the I/O processor automatically fetches the data from internal memory. Figure 3-1 on page 3-27 shows more detail on DMA channel data paths.

Peripheral to External Memory (SPORTs)

The SPORTs allow direct DMA transfers between the SPORT and external memory space. Programs do not need to first copy data into internal memory and then run an external port DMA to external memory space.

Internal Memory to Internal Memory

The SHARC processors can use memory-to-memory DMA to transfer 64-bit blocks of data between internal memory locations.

DMA Controller Addressing

Figure 3-1 shows a block diagram of the I/O processor's address generator (DMA controller). "Standard DMA Parameter Registers" on page 3-4 lists the parameter registers for each DMA channel. The parameter registers are uninitialized following a processor reset.

The I/O processor generates addresses for DMA channels much the same way that the Data Address Generators (DAGs) generate addresses for data memory accesses. Each channel has a set of parameter registers, including an index register and modify register that the I/O processor uses to address a data buffer in internal memory. The index register must be initialized with a starting address for the data buffer. As part of the DMA operation, the I/O processor outputs the address in the index register onto the processor's I/O address bus and applies the address to internal memory during each DMA cycle—a clock cycle in which a DMA transfer is taking place.

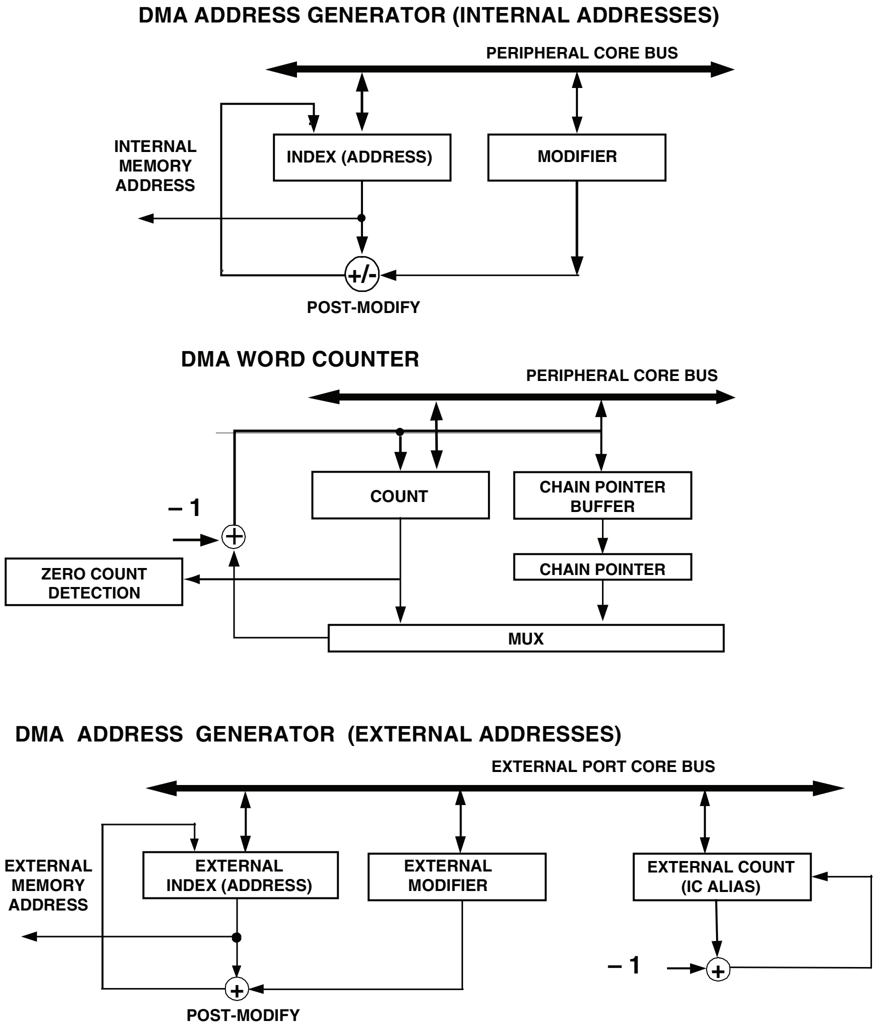

Figure 3-1. DMA Address Generator

Internal Index Register Addressing

All addresses in the index registers are offset by a value matching the processor's first internal normal word addressed RAM location, before the I/O processor uses the addresses. For the ADSP-214xx processors, this offset value is 0x0008 0000. This internal memory offset is not applicable for the index registers that correspond to SPORT DMAs as these registers are 28 bits.

The following rules for data transfers must be followed.

- The DMA controller requires data transfers with an I/O of 32 bits. Therefore index addresses must always be normal word space.
- If the peripheral receives smaller I/O sizes, the peripheral packs data into a 32-bit data format (the peripherals include the SPORT, SPI, UART, AMI, and link port) with the help of shift registers.

After transferring each data word to or from internal memory, the I/O processor adds the modify value to the index register to generate the address for the next DMA transfer and writes the modified index value to the index register. The modify value in the modify register is a signed integer, which allows both increment and decrement modifies. The modify value can have any positive or negative integer value. Note that:

- If the I/O processor modifies the internal index register past the maximum 19-bit value to indicate an address out of internal memory, the index wraps around to zero. With the offset for the SHARC processor, the wraparound address is 0x80000.
- If a DMA channel is disabled, the I/O processor does not service requests for that channel, whether or not the channel has data to transfer.

If a program loads the count register with zero, the I/O processor does not disable DMA transfers on that channel. The I/O processor interprets the zero as a request for 2^{16} transfers. This count occurs because the I/O processor starts the first transfer before testing the count value. To quickly disable a DMA channel, clear its channel DMA enable bit or write a 1 to the DMA word count register.

External Index Register Addressing

The external port DMA channels each contain additional parameter registers: the external index registers (EIEPx), external modify registers (EMEPx), and external count registers (ECEPx). The DMA controller generates 28-bit external memory addresses over the IOD1 bus using the EIEPx register during DMA transfers between internal memory and external memory.

Unlike previous SHARCs, all SPORT DMA channels can transfer data from the SPORTs to the external memory space. This transfer uses the 28-bit IIxSPx register.

DMA Channel Status

There are two methods the processor uses to monitor the progress of DMA operations; interrupts, which are the primary method, and status polling. The same program can use either method for each DMA channel. The following sections describe both methods in detail.

Programs can check the appropriate DMA status bits (for example the status bits in the SPMCTL register for the serial ports) to determine which channels are performing a DMA or chained DMA. All DMA channels can be active or inactive. If a channel is active, a DMA is in progress on that channel. The I/O processor indicates the active status by setting the channel's bit in the status register.

Note that there is 1 PCLK cycle latency between a change in DMA channel status and the status update in the corresponding register.

The peripheral's DMA controller tracks status information of the channels in each of the peripheral registers (for example SPMCTLx, SPIDMACx, DAI_STAT, DMACx, and MTMCTL).

- DMA channel status (status bit is set until the DMA terminates)
- TCB chain loading status (status bit is set until TCB loading completes)

If polling the status of a chained DMA, the DMA status bit is first set when the TCB has terminated, then it is cleared. The TCB status loading bit is set until the load is finished and cleared on load completion. This procedure is repeated for all subsequent DMA blocks.

Note that polling the DMA status registers (especially chained DMA) reduces I/O bandwidth.

DMA Bus Architecture

This section provides information on IOP bus architecture.

The SHARC processor contains two independent 32-bit DMA buses (Figure 3-3 on page 3-38). The IOD0 bus is used for the peripherals to the internal memory and the IOD1 bus is used for external-to-internal memory transfers.

The IOD0 bus is the path that the IOP uses to transfer data between internal memory and the peripherals. When there are two or more peripherals with active DMAs in progress, they may all require data to be moved to or from memory in the same cycle. For example, the SPI port may fill its buffer just as a SPORT shifts a word into its buffer. To determine which word is transferred first, the DMA channels for each of the processor's I/O ports negotiate channel priority with the I/O processor using an internal DMA request/grant handshake.

The IOD0 and IOD1 buses operate independently. However, in some cases there may be address conflicts if both buses access the same internal memory block. In this case, the IOD0 bus has first priority.

Each I/O port has one or more DMA channels, and each channel has a single request and a single grant. When a particular channel needs to read or write data to internal memory, the channel asserts an internal DMA request. The I/O processor prioritizes the request with all other valid DMA requests. When a channel becomes the highest priority requester, the I/O processor asserts the channel's internal DMA grant. In the next clock cycle, the DMA transfer starts.

Standard DMA Start and Stop Conditions

A standard DMA sequence starts when chaining is disabled, and the DMA enable bit transitions from low to high. Once a program starts a DMA process, the process is influenced by DMA channel priority.

A DMA sequence ends when one of the following occurs.

- The count register decrements to zero.
- Chaining is disabled and the channel's DMA enable bit transitions from high to low.

To abort a standard DMA, write a 1 directly to the DMA count register.

Operating Modes

The following sections provide information on the different operating modes supported through DMA.

Chained DMA

DMA data transfers can be set up as continuous or periodic. Furthermore, these DMA transfers can be configured to run automatically using chained DMA. With chained DMA, the attributes of a specific DMA are stored in internal memory and are referred to as a *Transfer Control Block* or TCB. The DMA controller loads these attributes in chains for execution. This allows for multiple chains that are an finite or infinite.

When chaining is enabled on a DMA channel, polling should not be used to determine channel status only because the DMA appears inactive if it is sampled while the next TCB is loading. In such cases where chaining is enabled, along with the polling of DMA status bit, polling of chaining status bit should also occur to so that the correct status of the DMA is known. For example, with an external port DMA with chaining enabled, the `CHS` bit should be polled as well as the `DMAS` and `EXTS` bits.

TCB Memory Storage

The location of the DMA parameters for the next sequence comes from the chain pointer register that points to the next set of DMA parameters stored in the processor's internal memory. In chained DMA operations, the processor automatically initializes and then begins another DMA transfer when the current DMA transfer is complete. Each new set of parameters is stored in a user-initialized memory buffer or TCB for a chosen peripheral. Table 3-28 provides a brief description of the TCBs.

The size of a TCB varies and is based on the peripheral to be used: the SPORTs, link ports and SPI require four locations, the external port requires six to 13 locations, the accelerator five to 13 locations. Allowing different TCB sizes reduces the memory load since only the required TCBs are allocated in internal memory.

Table 3-28. Principal TCB Allocation for a Serial Peripheral

Address	Register	Description
CPx	Chain pointer register	Chain pointer for DMA chaining
CPx + 0x1 (ICx)	Internal count register	Length of internal buffer
CPx + 0x2 (IMx)	Internal modify register	Stride for internal buffer
CPx + 0x3 (IIx)	Internal index register	Internal memory buffer

Chain Assignment

The structure of a TCB is conceptually the same as that of a traditional linked-list. Each TCB has several data values and a pointer to the next TCB. Further, the chain pointer of a TCB may point to itself to continuously re run the same DMA. The I/O processor reads each word of the TCB and loads it into the corresponding register.

Programs must assign the TCB in memory in the order shown in Figure 3-2 and Listing 3-1, placing the index parameter at the address pointed to by the chain pointer register of the previous DMA operation of the chain. The end of the chain (no further TCBs are loaded) is indicated by a TCB with a chain pointer register value of zero.

The address field of the chain pointer registers is only 19 bits wide. If a program writes a symbolic address to bit 19 of the chain pointer there may be a conflict with the `PCI` bit. Programs should clear the upper bits of the address then AND the `PCI` bit separately, if needed, as shown below.

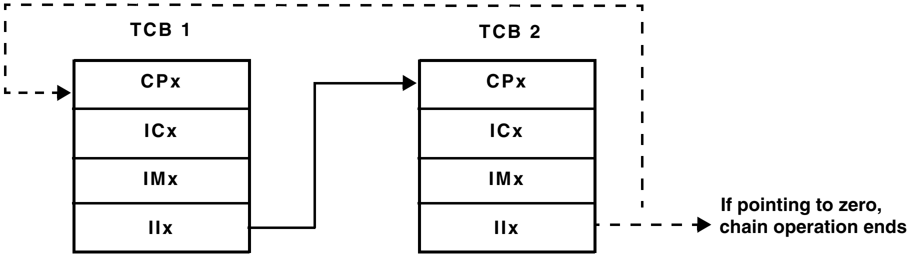

Figure 3-2. Chaining in the SPI and Serial Ports

ⓘ Clear the chain pointer register before chaining is enabled.

Listing 3-1. Chain Assignment

```
R0=0;
dm(CPx)=R0;            /* clear CPx register */

/* init DMA control registers */

R2=(TCB1+3) & 0x7FFFF;    /* load IIx address of next TCB
                             and mask address */
R2=bset R2 by 19;        /* set PCI bit */
dm(TCB2)=R2;             /* write address to CPx location of
                            current TCB */
R2=(TCB2+3) & 0x7FFFF;   /* load IIx address of next TCB and
                            mask address*/
R2=bclr R2 by 19;         /* clear PCI bit */
dm(TCB1)=R2;            /* write address to CPx location of
                           current TCB */
dm(CPx)=R2;             /* write IIx address of TCB1 to CPx
                           register to start chaining*/
```

ⓘ Chained DMA operations may only occur within the same channel. The processor does not support cross-channel chaining.

Starting Chain Loading

A DMA sequence is defined as the sum of the DMA transfers for a single channel, from when the parameter registers initialize to when the count register decrements to zero. Each DMA channel has a chaining enable bit (CHEN) in the corresponding control register.

To start the chain, write the internal index address of the first TCB to the chain pointer register. When chaining is enabled, DMA transfers are initiated by writing a memory address to the chain pointer register. This is also an easy way to start a single DMA sequence, with no subsequent chained DMAs.

During TCB chain loading, the I/O processor loads the DMA channel parameter registers with values retrieved from internal memory.

When starting chain loading, note that the SPI port is an exception to the above. To execute the first DMA in a chain for this peripheral, the DMA parameter registers also need to be explicitly programmed. For more information, see "DMA Transfers" on page 16-26.

The address in the chain pointer register points to the highest address of the TCB (containing the index parameter). This means that if a program declares an array to hold the TCB, the chain pointer register should point to the last location of the array and not to the first TCB location.

Buffered Chain Loading Register

The chain pointer register is buffered (see Figure 3-1 on page 3-27). Before the chain loading starts the buffer is copied into the chain pointer register and is decremented after each register is loaded.

The chain pointer register can be loaded with a new address at any time during the DMA sequence (CHEN bit =1). This allows a DMA channel to have chaining status deactivated (chain pointer register = 0x0) until some event occurs that loads the chain pointer register with a non zero value. Writing all zeros to the address field of the chain pointer register also deactivates chaining for the next TCB.

TCB Chain Loading Priority

A TCB chain load request is prioritized like all DMA channels. Therefore, the TCB chain loading request has the same priority level as the DMA channel itself. The I/O processor latches a TCB loading request and holds it until the load request has the highest priority. If multiple chaining requests are present, the I/O processor services the TCB block for the highest priority DMA channel first.

A channel that is in the process of chain loading cannot be interrupted by any other request (TCB, DMA channel). The chain loading sequence is atomic and the I/O bus is locked until all the DMA parameter registers are loaded. For a list of DMA channels in priority order, see Table 3-29.

Chain Insert Mode (SPORTs Only)

It is possible to insert a single SPORT DMA operation or another DMA chain within an active SPORT DMA chain. Programs may need to perform insertion when a high priority DMA requires service and cannot wait for the current chain to finish. This is supported only for SPORT DMA channels. For more information, see Chapter 11, Serial Ports (SPORTs).

Peripheral DMA Arbitration

DMA bus arbitration is required to delegate a peripheral's data stream over a common bus. DMA bus arbitration is a multistage process; peripheral needs to win multiple stages to finally get ownership of the DMA bus.

Peripheral Group Stage 1 Arbitration

The first arbitration stage (peripheral group arbitration) arbitrates between the peripherals themselves (SPORTs, IDP, MLB, UART, link port and external port DMA). The winning channel moves into the DMA bus arbitration (stage 2, Table 3-29 on page 3-39).

The peripheral group arbitration is fixed or rotating (not optional) depending on the peripheral (with one exception external port DMA channels)

Figure 3-3 shows the arbitration schemes for the peripheral and external port DMA bus.

Peripheral DMA Bus Stage 2 Arbitration

When more than one of these peripheral groups requests access to the IOD0 bus in a clock cycle, the bus arbiter, which is attached to the IOD0 bus, determines which master should have access to the bus and grants the bus to that master. Peripheral DMA bus arbitration can be set to use either a fixed or rotating algorithm by setting or clearing `DCPR` bit in the `SYSCTL` register as follows.

- fixed arbitration (default)
- rotating arbitration

In the fixed priority scheme (`DCPR` = 0), the lower indexed peripheral group has the highest priority as shown in Table 3-29.

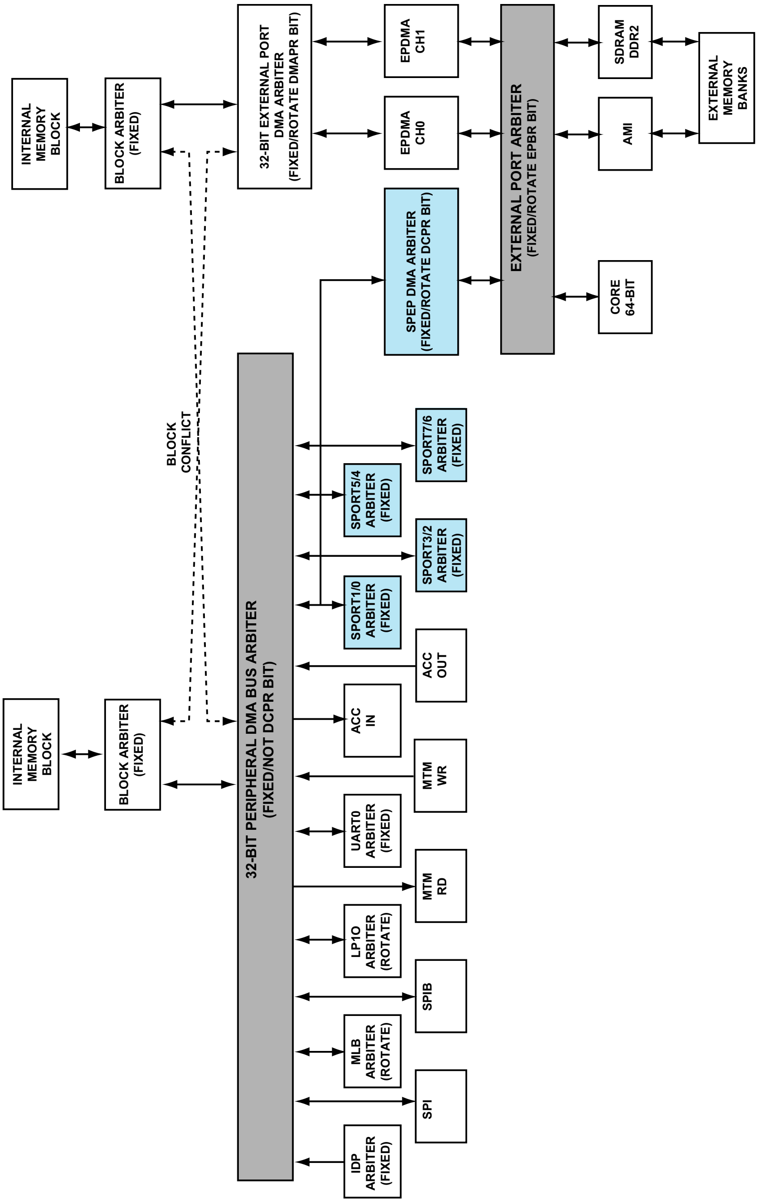

Figure 3-3. Peripheral Arbitration Schemes

External Port DMA Arbitration

Two peripheral groups and the core arbitrate for the external port bus:

- SPEP (SPORT/external port) DMA bus 32-bit
- External port DMA bus 32-bit
- Core bus 64-bit

Table 3-29. Peripheral DMA Channel Priorities for SHARC Processors

Peripheral DMA Bus Arbiter Optional (Stage 2)	Peripheral Group Arbiter (Stage 1)	Peripheral	Control Registers	Parameter Registers
A (highest)	0 (highest)	SPORT1A	SPCTL5-0, SPMCTL5-0	IISP5-0A, IMSP5-0A, CSP5-0A, CPSP5-0A, IISP5-0B, IMSP5-0B, CSP5-0B, CPSP5-0B
	1	SPORT1B		
	2	SPORT0A		
	3 (lowest)	SPORT0B		
B	0 (highest)	SPORT3A		
	1	SPORT3B		
	2	SPORT2A		
	3 (lowest)	SPORT2B		
C	0 (highest)	SPORT5A		
	1	SPORT5B		
	2	SPORT4A		
	3 (lowest)	SPORT4B		

Table 3-29. Peripheral DMA Channel Priorities for SHARC Processors (Cont'd)

Peripheral DMA Bus Arbiter Optional (Stage 2)	Peripheral Group Arbiter (Stage 1)	Peripheral	Control Registers	Parameter Registers
D	0 (highest)	IDP0	IDP_CTL2-0, IDP_PP_CTL	IDP_DMA_I7-0, IDP_DMA_M7-0, IDP_DMA_C7-0, IDP_DMA_I7-0A, IDP_DMA_I7-0B, IDP_DMA_PC7-0
	1	IDP1		
	2	IDP2		
	3	IDP3		
	4	IDP4		
	5	IDP5		
	6	IDP6		
	7 (lowest)	IDP7		
E		SPI	SPICTL, SPIDMAC,	IISPI, IMSPI, CSPI, CPSPI
F	0-30 channels (rotating)	Media LB	MLB_CECR30–0	MLB_SBCR, MLB_ABCR, MLB_CBCR, MLB_CCBCR30-0, MLB_CNBCR30-0
G		SPIB	SPICTLB, SPIDMACB	IISPIB, IMSPIB, CSPIB, CPSPIB
H		MTMWR	MTMCTL (or DTCP)	IIMTMW, IMMTMW, CMTMW
I		MTMRD	MTMCTL (or DTCP)	IIMTMR, IMMTMR, CMTMR
J	0 (highest)	UART0RX	UART0RXCTL	IIUART0RX, IMUART0RX, CUART0RX, CPUART0RX,
	1 (lowest)	UART0TX	UART0TXCTL	IIUART0TX, IMUART0TX, CUART0TX, CPUART0TX,
K	0 –1 channel (rotating)	LP0	LCTL1–0	IILB1-0, IMLB1-0, ICLB1-0, CPLB1-0
		LP1		

Table 3-29. Peripheral DMA Channel Priorities for SHARC Processors (Cont'd)

Peripheral DMA Bus Arbiter Optional (Stage 2)	Peripheral Group Arbiter (Stage 1)	Peripheral	Control Registers	Parameter Registers
L	0 (highest)	SPORT7A	SPCTL7-6, SPMCTL7-6	IISP7-6A, IMSP7-6A, CSP7-6A, CPSP7-6A, IISP7-6B, IMSP7-6B, CSP7-6B, CPSP7-6B
	1	SPORT7B		
	2	SPORT6A		
	3 (lowest)	SPORT6B		
M		ACC IN	PMCTL1, FIRCTL1, FIRCTL2, IIRCTL1, IIRCTL2, FFTCTL1, FFTCTL2	IIFIR, IMFIR, ICFIR, IBFIR,CIFIR, CMFIR, CLFIR, CPFIR
				IIIIR, IMIIR, ICIIR, IBIIR,CIIIR, CMIIR, CLIIR, CPIIR
				IIFFT, IMFFT, ICFFT, IBFFT, CIFFT, CMFFT, CLFFT, CPIFFT
N		ACC OUT		OIFIR, OMFIR,OCFIR, OBFIR, COFIR, CMFIR,CLFIR, CPFIR
				OIIIR, OMIIR, OCIIR, OBIIR, COIIR, CMIIR, CLIIR, CPIIR
				OIFFT, OMFFT, OCFFT, OBFFT, COFFT, CMFFT, CLFFT, CPOFFT

External Port Group Stage 1 Arbitration

External port DMA channels transfer data between internal memories or between internal and external memory over the IOD1 bus. When both external port channels request access to the IOD1 bus in a clock cycle, the external port bus arbiter, which is attached to the IOD1 bus, determines which master should have access to the bus and grants the bus to that master.

The external port channel arbitration can be set to use either a fixed or rotating algorithm by setting or clearing the `DMAPR` bits in the `EPCTL` register as follows.

- fixed arbitration channel 0 high
- rotating arbitration (default)

SPORT/External Port Group Stage 2 Arbitration

The data connection between the SPORT and the external port is performed over the SPEP (SPORT/external port) DMA bus. After the first arbitration stage in the SPORT group, the winning channels arbitrate for the external port bus. By default the SPORT0/1 group has highest priority and SPORT7/6 lowest priority. The arbitration of the SPEP bus can be set to use either a fixed or rotating mode by setting the `DCPR` bit in the `SYSCTL` register as follows.

- fixed arbitration (default)
- rotating arbitration

The `DCPR` bit controls arbitration on the peripheral and SPEP bus.

External Port DMA Bus Stage 3 Arbitration

In the last stage the SPORT group, DMA group and the core arbitrate for the DMA bus. The arbitration can be set to use either a fixed or rotating algorithm by setting the EPBR bits in the EPCTL register as follows.

- Priority order from highest to lowest is SPORT, external port DMA, core.
- Priority order from highest to lowest is external port DMA, SPORT, core.
- Highest priority is core. SPORT and external port DMA are in rotating priority.
- Rotating priority (default).

Table 3-30 shows the priority from highest to lowest SPORT, external port DMA and core (EPBR = 00).

Fixed Versus Rotating Priority

Programs can change DMA arbitration modes between fixed and rotate on the fly which incurs an effect latency of 2 PCLK cycles.

Peripheral and External Port DMA Block Conflicts

Note that if both DMA buses arbitrate for the same internal memory block (Figure 3-3 on page 3-38) the peripheral DMA always has a higher priority over the external port DMA bus. For more information refer to "Memory" chapter of *ADSP-2136x Programming Reference Manual*.

Interrupts

This section provides information on using interrupts. This information includes interrupt sources, masking and servicing.

Table 3-30. External Port Bus Priorities for SHARC Processors (EPBR = 00)

External Port Bus Arbiter optional (Stage 3)	SPEP Group Arbiter optional (Stage 2)	Peripheral Group Arbiter (Stage 1)	Peripheral	Control Registers	Parameter Registers
A (highest)	0 (highest/ rotating)	0 (highest)	SPORT1A	SPCTL7-0, SPMCTL7-0	IISP7-0A, IMSP7-0A, CSP7-0A, CPSP7-0A, IISP7-0B, IMSP7-0B, CSP7-0B, CPSP7-0B
		1	SPORT1B		
		2	SPORT0A		
		3 (lowest	SPORT0B		
	1	0 (highest)	SPORT3A		
		1	SPORT3B		
		2	SPORT2A		
		3 (lowest	SPORT2B		
	2	0 (highest)	SPORT5A		
		1	SPORT5B		
		2	SPORT4A		
		3 (lowest	SPORT4B		
D	3	0 (highest)	SPORT7A		
		1	SPORT7B		
		2	SPORT6A		
		3 (lowest)	SPORT6B		
E		0 (highest/ rotating)	EPDMA0	DMAC1–0	IIEP1-0, IMEP0, ICEP0, EIEP0, EMEP0, ELEP0, EBEP0, RIEP0, RCEP0, RMEP0, TCEP0, TPEP0, CPEP0
		1 (lowest)	EPDMA1		
F			64-Bit Core		

Sources

The information in this section is generic and provides a basic understanding of interrupt sources. For more information, see the "Interrupts" section of the specific peripheral.

DMA Complete

When a standard (single block) DMA process reaches completion (the DMA count decrements to zero) on any DMA channel, the interrupt controller latches that DMA channel's interrupt.

The next two sections describe the two types of interrupts that are used to signal interrupt completion. These are based on the type of peripheral used.

Internal Transfer Completion

This mode of interrupt generation resembles the traditional SHARC DMA interrupt generation. The interrupt is generated once the DMA internal transfers are complete, independent of whether the DMA is a transmit or receive. Therefore, when the completion interrupt is generated for external transmit DMAs, there may still be an external access pending at the external DMA interface.

The I/O processor only generates a DMA complete interrupt when the channel's count register decrements to zero as a result of actual DMA transfers. Writing zero to a count register does not generate the interrupt.

To stop a DMA preemptively, write a one to the count register. This causes one additional word to be transferred or received, and an interrupt is then generated.

Access Completion

A DMA complete interrupt is generated when accesses are finished. For an external write DMA, the DMA complete interrupt is generated only after the external writes on the DMA external interface are complete. For an external read DMA, the interrupt is generated when the internal DMA writes are complete. In this mode the DMA interface can be disabled as soon as the interrupt is received.

The access completion option is supported by the SPORTs, SPI, link ports and external port.

Chained DMA Interrupts

For chained DMA, the channel generates interrupts in one of two ways:

1. If `PCI` = 1, (bit 19 of the chain pointer register is the program controlled interrupts, or `PCI` bit) an interrupt occurs for each DMA in the chain.
2. If `PCI` = 0, an interrupt occurs at the end of a completed chain. For more information on DMA chaining, see “Functional Description” on page 3-23.

Figure 3-4 shows the PCI timing during TCB loading. After the DMA count for the last word of frame N becomes zero, the PCI interrupt is latched. At the same time the DMA reloads the TCB for that specific channel (assuming no higher priority DMA requests). Finally the DMA channel resumes operation for frame N–1.

By clearing a channel’s `PCI` bit during chained DMA, programs mask the DMA complete interrupt for a DMA process within a chained DMA sequence.

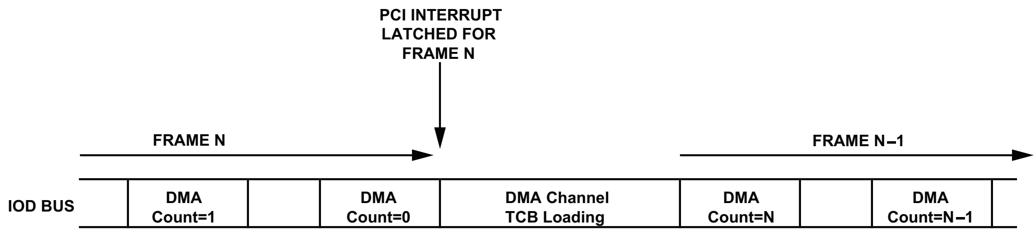

Figure 3-4. DMA Chaining

Masking

For information on interrupt masking, see the "Masking" section of the specific peripheral.

Service

For information on interrupt masking, see the "Service" section of the specific peripheral.

Interrupt Versus Channel Priorities

At their default setting shown in Table 3-31, the DMA interrupt priorities do not match the DMA channel priorities. However, if both priorities schemes should match, the DMA interrupt priorities can be re-assigned by dedicated settings of the `PICRx` registers.

ⓘ At their default setting for the external port there are two programmable interrupts (P9I and P13I) which arbitrate against two external port DMA channels.

Table 3-31. Default Channel vs. Interrupt Priorities (Peripheral DMA Bus)

Programmable Interrupt	Default Interrupt Priority	Priorities	Default DMA Channel Priority
P0I	DAIHI	Highest	SPORT5–0, 12 channels
P1I	SPII		
P3I	SP1I		IDP7–0, 8 channels
P4I	SP3I		
P5I	SP5I		SPI – 1 channel
P6I	SP0I		
P7I	SP2I		MLB – 31 channels
P8I	SP4I		
P11I	SP7I		SPI B – 1 channel
P12I	DAILI		
P14I	DPII		MTM – 2 channels
P15I	MTMI		
P16I	SP6I		UART0 – 2 channels
P17I	No default		
P18I	SPIBI	Lowest	SPORT7–6, 4 channels Accelerator (I/O), 2 channels

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

IOP Effect Latency

Table 3-32 lists the time required to load a specific TCB from the internal memory into the DMA controller. During this time, both buses (for a peripheral DMA, the IOD0 bus and for external port DMA the IOD1 bus) are locked and cannot be interrupted.

IOP Throughput

Since the I/O processor controls two I/O buses (peripheral and external port) the maximum bandwidth per IOD bus is gained for:

- Internal memory writes with $f_{PCLK} \times$ 32-bit
- Internal memory isolated reads with $f_{PCLK}/3 \times$ 32-bit
- Internal memory back to back reads with $f_{PCLK}/2 \times$ 32-bit

The maximum bandwidth per IOD1 bus (external port) is gained for:

- Internal memory writes with $f_{PCLK} \times$ 32-bit
- Internal memory isolated reads with $f_{PCLK} \times$ 32-bit
- Internal memory back to back reads with $f_{PCLK} \times$ 32-bit

Table 3-32. I/O Processor TCB Chain Loading Access

Chained TCB Type	TCB Size	Number of Core Cycles
SPI DMA, SPORT DMA[1], Link port DMA	4	26
IIR Accelerator DMA coefficient[2]	5	22
IIR Accelerator DMA data[2]	10	40
External Port standard DMA, FFT Accelerator DMA, Delay Line DMA read	6	34
External Port Circular Buffer DMA, Delay Line DMA write	7	40
External Port Scatter/Gather DMA	8	42
External Port Circular Buffer Scatter/Gather DMA	10	50
FIR Accelerator DMA[2]	13	94

1 If the TCB for a SPORT is located in external memory, additional access cycles are required for External Port arbitration and AMI or DDR2 cycles.
2 For throughput performance add 6 core cycles.

Programming Model

This section provides a general procedure for configuring DMAs. There is more specific information on DMA in each peripheral chapter.

General Procedure for Configuring DMA

To configure the processors to use DMA, use the following general procedure. Note this is a generic model. For specific information refer to the individual programming model section in the peripheral specific chapter.

1. Clear all relevant registers (DMA/peripheral control, chain pointer).
2. Determine interaction method (enable `IRQEN`/`IMASK` setting or status polling).

3. Define the DMA channels interrupt priority (`PICR` registers).
4. Determine the DMA channel priority (fixed or rotating).
5. Determine the DMA address region for source and destination (index, modifier, count).
6. Determine the DMA transfer type (standard, chained, circular).
7. Set the DMA enabled bit/write index address of first TCB to the chain pointer register.

Debug Features

The JTAG interface provides some user debug features for DMA in that it allows programs to place breakpoints on the IOD buses. Programmers can then insert DMA related breakpoints. For more information, see the CrossCore or VisualDSP++ tools documentation and *SHARC Processor Programming Reference*.

Emulation Considerations

An emulation halt will optionally stop the DMA engine. The JTAG interface provides some user debug features for DMA. Placing breakpoints on the IOD address buses allows DMA related breakpoints. For more information, see the CrossCore or VisualDSP++ tools documentation and *SHARC Processor Programming Reference*.

4 EXTERNAL PORT

The external memory interface provides a glueless interface to external memories. The asynchronous memory interface and the SDRAM/DDR2 memory that interfaces to the external port is clocked by the SDRAM or DDR2 clock. The interface specifications are shown in Table 4-1.

Table 4-1. External Port Specifications

Feature	AMI	SDRAM Interface (ADSP-2147x and ADSP-2148x Processors)	DDR2 Interface (ADSP-2146x Processor)
Connectivity			
Multiplexed Pinout	Yes (External Port)	Yes (External Port)	No
SRU DAI Required	No	No	No
SRU DAI Default Routing	N/A	N/A	N/A
SRU2 DPI Required	No	No	No
SRU2 DPI Default Routing	N/A	N/A	N/A
Interrupt Control	Yes	Yes	Yes
Protocol			
Master Capable	Yes	Yes	Yes
Slave Capable	No	No	No
Transmission Simplex	Yes	Yes	Yes
Transmission Half-Duplex	Yes	Yes	Yes
Transmission Full-Duplex	No	No	No

Table 4-1. External Port Specifications (Cont'd)

Feature	AMI	SDRAM Interface (ADSP-2147x and ADSP-2148x Processors)	DDR2 Interface (ADSP-2146x Processor)
Access Type			
Data Buffer	Yes	Yes	Yes
Core Data Access	Yes	Yes	Yes
DMA Data Access	Yes	Yes	Yes
DMA Channels	2	2	2
DMA Chaining	Yes	Yes	Yes
Miscellaneous			
Clock Power Management	Yes	Yes	Yes
Boot Capable	Yes	No	No
Local Memory	No	No	No
Max Clock Operation	SDCLK or DDR2-CLK	SDCLK	DDR2CLK

Features

The external port has the following features.

- Supports access to the external memory by core and DMA accesses. The external memory address space is divided into four banks. Any bank can be programmed as either asynchronous or synchronous memory.
- An asynchronous memory interface which communicates with SRAM, FLASH, and other devices that meet the standard asynchronous SRAM access protocol.

- A SDRAM controller (ADSP-2147x and ADSP-2148x processors) that supports a glue-less interface with any of the standard SDRAMs.
- A DDR2 controller (ADSP-21469 processor) that supports a glue-less interface with any of the standard DDR2.
- A 2-channel external port DMA which supports standard, circular, chained, scatter/gather and delay line operating modes for internal to external or internal to internal transfers and optional direction change on the fly.
- Arbitration logic to coordinate between SPORT, AMI, SDRAM/DDR2 transfers (core versus DMA) between internal and external memory over the external port.
- External port supports various ratios of core to external port clock determined by programming bits in the power management control registers (PMCTL). For more information, see "Power Management Registers (PMCTL, PMCTL1)" on page A-7.

Pin Descriptions

For the external port pin descriptions of the AMI and SDRAM/DDR2 interfaces, see the appropriate processor-specific data sheet.

Pin Multiplexing

The address data and memory select pins are multiplexed for the AMI and SDRAM controller of ADSP-2147x and ADSP-2148x processors. The ADSP-2146x processors have dedicated pins for the AMI and the DDR2 controller and therefor are not multiplexed.For more information on multiplexing schemes refer to "Pin Multiplexing" on page 24-28.

Register Overview

This section provides brief descriptions of the major registers. For complete register information, see Appendix A, Register Reference.

External Port

External Port Control (EPCTL). This register enables the external banks for the SDRAM or the AMI. Moreover controls accesses between the processor core and DMA, and between different DMA channels.

External Port DMA (DMAC1–0). DMA transfers between the internal and external memory space are controlled with these registers. For the corresponding DMA modes/parameter register information, see "External Port DMA" on page 4-125.

Power Management Control (PMCTL). Controls the SDCLK to core clock ratio or DDR2CLK to core clock ratio related to the external port timing.

Asynchronous Memory Interface

AMI Control (AMICTLx). These registers control the mode of operations for the four banks of external memory. Note for all AMI timing bit settings, all defined cycles are derived from the SDRAM/DDR2 clock.

AMI Status (AMISTAT). This 32-bit register provides status information for the AMI interface and can be read at any time.

SDRAM Controller

SDRAM Control (SDCTL). Configures various aspects of SDRAM operation. These are control clock operation, bank configuration, and SDRAM commands. Programmable parameters associated with the SDRAM access timing.

SDRAM Control Status (SDSTATx). Provides information on the state of the controller. This information can be used to determine when it is safe to alter SDRAM control parameters or as a debug aid.

SDRAM Refresh Rate Control (SDRRC). Provides a flexible mechanism for specifying auto-refresh timing.

DDR2 Controller

DDR2 Control 0 (DDR2CTL0). Contains the bits that control the DDR2 size, enables mode register and allows forcing of specific DDR2 commands.

DDR2 Control 1 (DDR2CTL1). Includes the timing programmable parameters associated with the DDR2 access timing. All the values for this register are defined in terms of number of clock cycles from the DDR2 data sheet.

DDR2 Control 2 (DDR2CTL2). Includes the programmable parameters associated to the burst type, burst length and CAS latency.

DDR2 Control 3–5 (DDR2CTLx). Include the programmable parameters associated with the DDR2 extended mode registers 1 through 3.

DDR2 Status (DDR2STAT1–0). Provide information on the state of the DDR controller. This information can be used to determine when it is safe to alter DDR control parameters or as a debug aid.

DDR2 Refresh Control (DDR2RRC). Provides a programmable refresh counter which has a period based value which coordinates the supplied clock rate with the DDR2 device's required refresh rate.

DDR2 DLL Control (DLL1–0CTL1). A built-in DLL in the DDR2 controller provides a 90° phase shifted clock to manage the data (DDR2_DATA) to data strobe (DDR2_DQS) timing relationships. For each data byte a control register is responsible. The bits are used to reset the DLL logic and to start a new DLL initialization.

DDR2 DLL Status (DLL1–0STAT0). After the built-in DLL has started the bits return the status if the DLL has locked. A control register is responsible for each data byte.

DDR2 Pad Control (DDR2PADCTL1–0). If the DDR2 interface is not used, these registers should be used to power-down the receiver pads for further power savings.

Shared DDR2 Memory

System Control (SYSCTL). Contains control bits for shared memory as Force synchronization and enable bus lock.

System Status (SYSTAT). Returns the status of synchronized system and the current bus master.

Bus Time-out Maximum, Bus Count (BMAX, BCOUNT). Contains counters which forces the current bus master to relinquish the bus if expired to guarantee a fair bus sharing.

Clocking AMI/SDRAM (ADSP-2147x/ ADSP-2148x Models)

The fundamental timing clock of the external port is SDRAM clock (SDCLK).

The AMI/SDRAM controller is capable of running at up to 166 MHz for ADSP-2148x processors and 133 MHz for ADSP-2147x processors. The various possible AMI/SDRAM clock to core clock frequency ratios are shown in Table 4-2.

The SDRAM clock ratio settings are independent from the peripheral clock (PCLK).

Table 4-2. External Port Clock Frequencies

CCLK:SDCLK Clock Ratio	CCLK = 400 MHz	CCLK = 333 MHz	CCLK = 266 MHz	CCLK = 200 MHz
1:2.0	N/A	166	133	100
1:2.5	160	133	106	80
1:3.0	133	111	88	67
1:3.5	114	95	76	57
1:4.0	100	83	66	50

For information on processor instruction rates, see the appropriate processor data sheets.

To obtain certain higher SDRAM frequencies, the core frequency may need to be reduced.

The external port and SDRAM clocks may be shut off for power savings. For more information, see "Disabling Peripheral Clocks" on page 23-11.

Clocking AMI/DDR2 (ADSP-2146x Models)

The fundamental timing clock of the external port is DDR2 clock (DDR2_-CLK). The AMI/DDR2 controller is capable of running up to core clock/2 speed (CCLK/2) and can run at various frequencies, depending on the programmed DDR2 clock (DDR2_CLK) to core clock (CCLK) ratios. For information processor instruction rates, see the appropriate processor data sheet.

> The DDR2 clock ratio settings are independent from the peripheral clock (PCLK).

The external port and DDR2 clocks may be shut off for power savings. For more information, see "Disabling Peripheral Clocks" on page 23-11.

External Port Arbiter

The external port arbiter is a key component of the module. The arbiter performs the following functions.

- Controls the speed for the AMI SDRAM/DDR2
- Controls the external banks individually
- Performs access arbitration for the processor core, AMI, SDRAM/DDR2 and SPORT access
- Allows channel freezing between core and DMA access to improve fairness of bus ownership

Functional Description

The external port has four ports for communication:

- Peripheral core bus for control of external port IOP registers
- External port core 64-bit bus for core access to external memory banks.
- External port DMA bus for transfers between the external port and internal memory.
- SPORT EP DMA bus for transfers between the external port and the SPORTs.

Figure 4-1 shows a diagram of the external port for the ADSP-2147x and ADSP-2148x processors (containing a SDRAM interface).

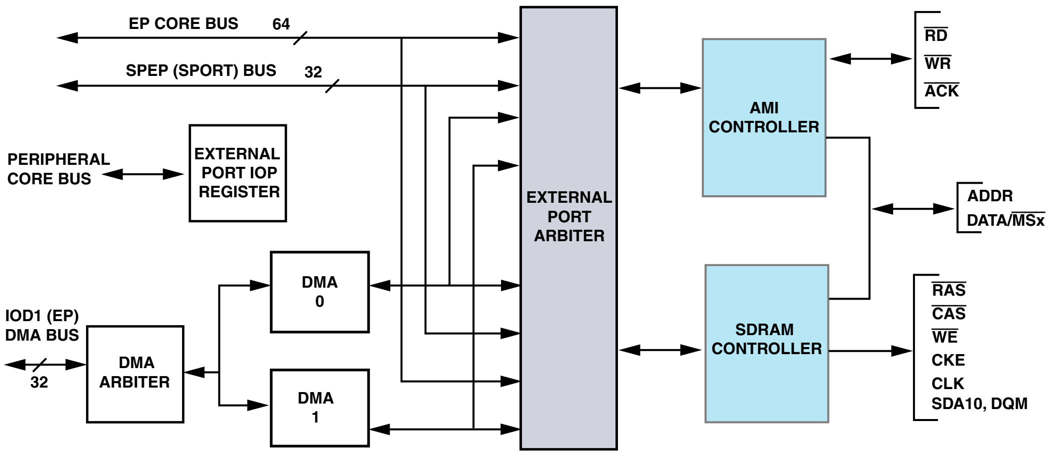

Figure 4-1. Functional Block Diagram (ADSP-2147x and ADSP-2148x Processors)

Figure 4-2 shows a diagram of the external port for the ADSP-2146x processor (containing a DDR2 interface).

As shown in the figures, the external port is a fundamental block since every access in the external memory space is handled by this port. The AMI or the SDRAM/DDR2 controller modules act as peripherals to the external world and as such they are responsible for filling the buffers with data based on the protocol used. The external port also keeps track of the two DMA channels which can serve as data streams via the external and internal memory.

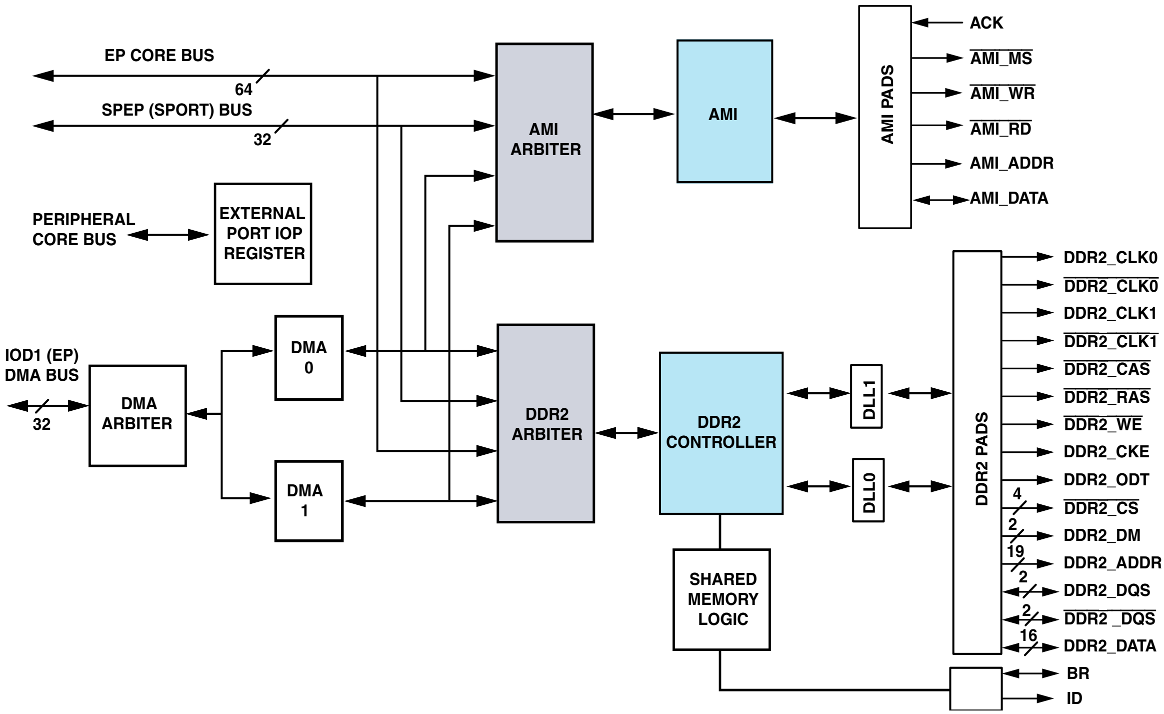

Figure 4-2. Functional Block Diagram (ADSP-2146x Processor)

Operating Mode

The following operation mode applies to the external port arbiter.

Arbitration Modes

Arbitration can be changed to be fixed or rotating (default). The SPORT/external port 32-bit DMA bus (SPEP) arbiter and the external port arbiter allow priority rule changes. For more information, see "SPORT/External Port Group Stage 2 Arbitration" on page 3-42.

The external port uses a three stage arbitration process whereby all DMA requests need to pass through the first stage until one request wins. When this occurs, the winning DMA channel needs to arbitrate with a SPORT DMA group (for example group A has four DMA channels SP1A/B, SP0A/B). The winning DMA channel then has a last arbitration process with the core where the following occurs.

1. External port DMA channels 0/1 rotating priority or high/low priorities.
2. Winning DMA channel arbitrating with SPORT DMA groups.
3. Winning DMA channel arbitrating with core access.

In the EPCTL register, the EBPR and DMAPR bits define the priorities. All the bits of EPCTL register can be changed only when the external port is idle (when all DMA engines are idle and no core or SPORT access to external memory are pending).

Arbitration Freezing

Arbitration length freezing can be used to improve the throughput of read accesses by programming the various freeze bits of the EPCTL register.

When multiple DMA channels are reading data from SDRAM/DDR2 memory, channel freezing can improve the data throughput. By setting the freeze bits (FRZDMA, FRZCR, and FRZSP), each channel request is frozen for programmed accesses. For example, if the processor core is frozen for 32 accesses, and if the core requests 32 accesses to SDRAM/DDR2 sequentially, data throughput improves.

Freezing is based on the fact that sequential accesses to the SDRAM/DDR2 provide better throughput than non-sequential accesses. The arbiter also allows core or DMA access freeze which helps to balance out system performance.

Channel freezing has no effect on write accesses.

Asynchronous Memory Interface

The asynchronous memory interface (AMI) is described in the following sections.

Features

The AMI has the following features and capabilities.

- User defined combinations of programmable wait states.
- External hardware acknowledge signals.
- Data packing support for 8 and 16 bits (ADSP-2147x and ADSP-2148x).
- External instruction fetch from 8 and 16 bits (16 bits for ADSP-2147x and ADSP-2148x).
- Both the processor core and the I/O processor have access to external memory using the AMI.
- Support a glueless interface with any of the standard SRAMs.
- Bank 0 can accommodate up to 6M words, and banks 1, 2, and 3 can accommodate up to 8M words each (ADSP-2147x and ADSP-2148x).
- Bank 0 can accommodate up to 2M words, and banks 1, 2, and 3 can accommodate up to 4M words each (ADSP-2146x).

Functional Description

The following sections provide a functional overview of the asynchronous memory interface.

The AMI communicates with SRAM, FLASH and any other memory device that conforms to its protocol. It provides a DMA interface between internal memory and external memory, performs instruction (48-bit) fetch from external memory, and directs core access to external memory locations. The AMI on the ADSP-2147x and ADSP-2148x supports 8- and 16-bit data access to external memory.

The external interface follows standard asynchronous SRAM access protocol. The programmable wait states, hold cycle and idle cycles are provided to interface memories of different access times. To extend access the ACK signal can be pulled low by the external device as an alternative to using wait states.

For ADSP-2146x products, writing to AMI memory space with the AMIEN bit in the AMICTLx register = 0, the write is postponed until the AMI controller is enabled (AMIEN bit = 1). However, once this occurs, the AMI address pins and the $\overline{\text{WR}}$ strobe start to toggle uncontrollably, causing the AMI to fail. Therefore, programs must enable the AMI (AMIEN bit = 1) prior to allowing any writes.

Parameter Timing

This section describes the programmable timing parameters for the AMI which include wait states for idle or hold cycles. Programmable timing allows the interface to be flexible and efficient regardless of whether the data transfers are being run from the core or from DMA or regardless of the sequence of transactions (read followed by read, read followed by write, and so on).

The bits used to set programmable timing for the AMI are located in "AMI Control Registers (AMICTLx)" on page A-27.

Asynchronous Reads

Figure 4-3 shows an asynchronous read bus cycle. Asynchronous read bus cycles proceed as follows.

1. At the start of the setup period, $\overline{\texttt{MSx}}$ and $\overline{\texttt{AMI\_RD}}$ assert. The address bus becomes valid.
2. At the beginning of the read access period and after the 3rd cycles, $\overline{\texttt{AMI\_RD}}$ deasserts.
3. At the beginning of the hold period, read data is sampled on the rising edge of the `SDCLK` clock.
4. At the end of the hold period, some $\overline{\texttt{IDLE}}$ cycles happened in the case the read is followed by a write. Also, $\overline{\texttt{MSx}}$ deasserts unless the next cycle is to the same memory bank.

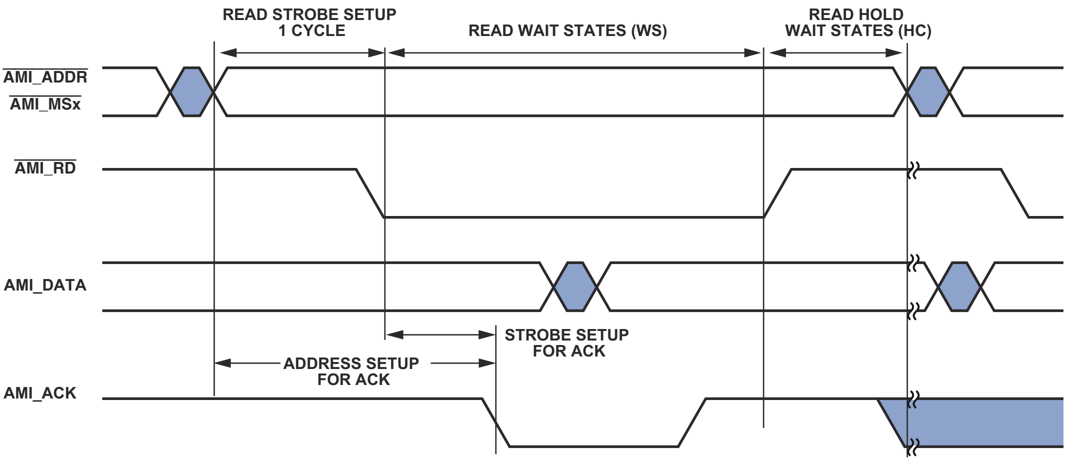

Figure 4-3. AMI Asynchronous Reads

Asynchronous Writes

Figure 4-4 shows an asynchronous write bus cycle. Asynchronous write bus cycles proceed as follows.

1. At the start of the setup period, $\overline{\texttt{MSx}}$, the address bus, data buses, become valid.
2. At the beginning of the write access period, $\overline{\texttt{WR}}$ asserts.
3. At the beginning of the hold period, $\overline{\texttt{WR}}$ deasserts.
4. One hold cycle is introduced before next access can happen. Also, $\overline{\texttt{MSx}}$ deasserts unless the next cycle is to the same memory bank.

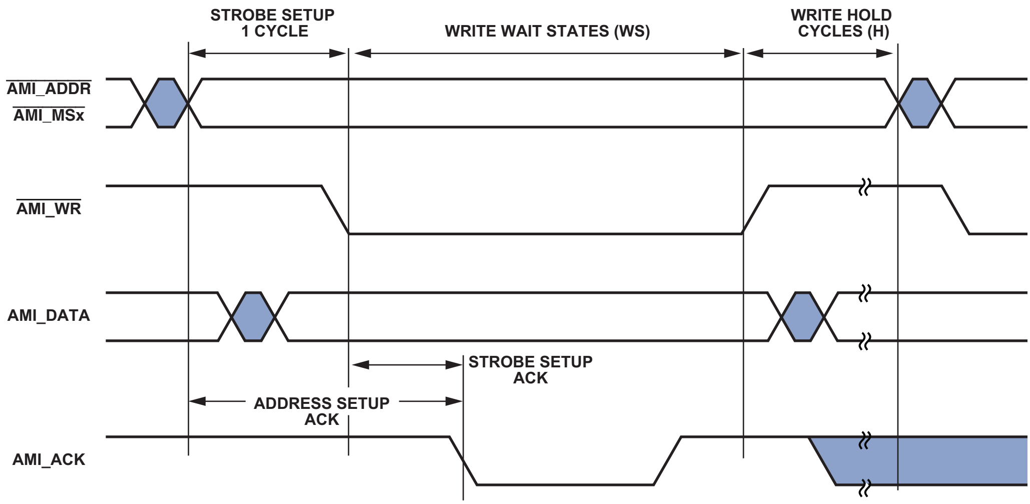

Figure 4-4. AMI Asynchronous Writes

Idle Cycles

An idle cycle is inserted by default for an AMI read followed by write or a read followed by a read from a different bank or a read followed by an external access by another device in order to avoid data bus driver conflicts.

If an idle cycle is programmed for a particular bank, then a minimum of 1 idle cycle is inserted for reads even if they are from the same bank. In order to achieve better read throughput, an idle cycle should be set to 0. For more information refer to the product-specific data sheet.

Wait States

Wait states and acknowledge signals are used to allow the processors to connect to memory-mapped peripherals and slower memories. Wait states are programmable from 1 to 31.

Hold Cycles

A bus hold cycle is an inactive bus cycle that the processor automatically generates at the end of a write to allow a longer hold time for address and data. Programs may disable holds, or hold off processing for one or more external port processor cycles. Note the address, data (if a write), and bank select (if in banked external memory) remain unchanged and are driven for one or more cycles after the read or write strobes are deasserted.

Data Storage and Packing

The processors have the ability to use logical addressing when an external memory smaller than 32 bits is used. When logical addresses are used, multiple external addresses seen by the memory correspond to a single internal address, depending on the width of the memory being accessed, and the packing mode setting of the AMI controller.

The external physical address map is shown in Table 4-3.

For an external bus width of 8 bits with packing enabled (`PKDIS` = 0), the external physical address `ADDR23-0` generation is `ADDR23-2` = bits 21–0 in the address being supplied to the external port by the core or DMA controller. Here, `ADDR1-0` corresponds to the 1st/2nd/3rd/4th 8-bit word.

Table 4-3. AMI Address Memory Map

Bus Width	External Memory Bank	Internal Logical Address (supported memory map)	External Physical Address (on ADDR23–0)
ADSP-2146x/ADSP-2147x/ADSP-2148x			
8-bit (and PKDIS = 0)	0	0x0020_0000 – 0x003F_FFFF	0x80_0000 – 0xFF_FFFF
8-bit (and PKDIS = 0)	1, 2, 3	0x0400_0000 – 0x043F_FFFF 0x0800_0000 – 0x083F_FFFF 0x0C00_0000 – 0x0C3F_FFFF	0x00_0000 – 0xFF_FFFF
ADSP-2147x/ADSP-2148x			
16-bit (and PKDIS = 0)	0	0x0020_0000 – 0x007F_FFFF	0x40_0000 – 0xFF_FFFF
16-bit (and PKDIS = 0)	1, 2, 3	0x0400_0000 – 0x047F_FFFF 0x0800_0000 – 0x087F_FFFF 0x0C00_0000 – 0x0C7F_FFFF	0x00_0000 – 0xFF_FFFF

External Instruction Fetch

The processors support direct fetch of ISA instructions from external memory, using the 8/16-bit external port. Fetching is supported from external memory space bank 0 which is selected by $\overline{\text{MS0}}$. This external memory can either be asynchronous memory, such as SRAM or flash.

Interrupt Vector Table (IVT)

The interrupt vector table can be located in the internal ROM (0x80 000, IIVT bit = 0) or internal RAM (0x8C 000, IIVT bit = 1) based on the selected boot mode. However for all boot modes except the reserved boot mode, the default IIVT bit setting is 1 (SYSCTL).

Therefore, if instruction fetch from external memory is desired at reset, the program needs to set up the appropriate interrupt vector tables in internal memory as part of the boot-up code before beginning to fetch these instructions.

When an unmasked interrupt occurs and is serviced, program execution automatically jumps to the location of the corresponding interrupt vector table in internal memory. Upon returning from the interrupt, the sequencer resumes fetching instructions from external memory because locating the IVT in external memory is not supported.

Instruction Packing

Any address produced by the sequencer which falls in external memory is first translated into the physical address in external memory based on the actual data bus width of external memory as shown in Figure 4-5.

The controller completes the required number of accesses from consecutive locations for returning a 48-bit word instructions.

Only bank0 can be populated for external instruction fetch.

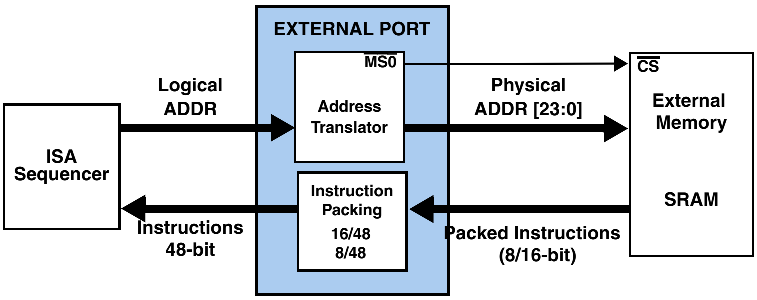

Figure 4-5. Logical Versus Physical Addresses

External Instruction Fetch from AMI Boot Space

External instruction fetch (ISA) from boot prom is useful if functions are only executed once (runtime environment, routing for example) and performance is not a primary concern. This type of instruction fetch helps to reduce the internal memory load.

For systems that boot and fetch instructions from a boot PROM, additional external logic is required. For AMI boot the processor asserts the $\overline{\text{MS1}}$ memory chip select only, and for external instruction fetch it asserts the $\overline{\text{MS0}}$ memory chip select only. Therefore both memory selects need to be combined to assert the memory select for both cases.

8-Bit Instruction Storage and Packing

For packed 8-bit instructions the controller performs six required accesses from consecutive locations for returning a 48-bit word instruction.

In Table 4-4, the logical to physical translation is a multiplication by a factor of 6 and N = 0xAAAA9. Therefore, the 8-bit wide AMI supports 0.7 million instructions.

Table 4-4. Logical Versus Physical Address Mapping, 8-Bit AMI

Logical ISA Normal Word Address, Program Sequencer	Physical Address, External Bus	Data 7–0
0x20 0000	0xC0 0000	Instr0[7:0]
	0xC0 0001	Instr0[15:8]
	0xC0 0002	Instr0[23:16]
	0xC0 0003	Instr0[31:24]
	0xC0 0004	Instr0[39:32]
	0xC0 0005	Instr0[47:40]
0x20 0001	0xC0 0006	Instr1[7:0]
	0xC0 0007	Instr1[15:8]
	0xC0 0008	Instr1[23:16]
	0xC0 0009	Instr1[31:24]
	0xC0 000A	Instr1[39:32]
	0xC0 000B	Instr1[47:40]
...	...	

Table 4-4. Logical Versus Physical Address Mapping, 8-Bit AMI (Cont'd)

Logical ISA Normal Word Address, Program Sequencer	Physical Address, External Bus	Data 7–0
0x2A AAA9	0xFF FFFA	InstrN[7:0]
	0xFF FFFB	InstrN[15:8]
	0xFF FFFC	InstrN[23:16]
	0xFF FFFD	InstrN[31:24]
	0xFF FFFE	InstrN[39:32]
	0xFF FFFF	InstrN[47:40]

16-Bit Instruction Storage and Packing

For packed 16-bit instructions the controller performs three required accesses from consecutive locations for returning a 48-bit word instruction.

In Table 4-5 the logical to physical translation is a multiplication by a factor of 3 and N = 0x355554. Therefore, the 16-bit wide AMI memory supports 3.3 million instructions.

Table 4-5. Logical Versus Physical Address Mapping, 16-Bit AMI

Logical ISA Normal Word Address, Program Sequencer	Physical Address, External Bus	Data 15–0
0x20 0000	0x60 0000	Instr0[15:0]
	0x60 0001	Instr0[31:16]
	0x60 0002	Instr0[47:32]
0x20 0001	0x60 0003	Instr1[15:0]
	0x60 0004	Instr1[31:16]
	0x60 0005	Instr1[47:32]

Table 4-5. Logical Versus Physical Address Mapping, 16-Bit AMI (Cont'd)

Logical ISA Normal Word Address, Program Sequencer	Physical Address, External Bus	Data 15–0
0x20 0002	0x60 0006	Instr2[15:0]
	0x60 0007	Instr2[31:16]
	0x60 0008	Instr2[47:32]
...	...	
0x55 5554	0xFF FFFD	InstrN[15:0]
	0xFF FFFE	InstrN[31:16]
	0xFF FFFF	InstrN[47:32]

Mixing Instructions and Data in External Bank 0

It is possible to store both 48-bit instructions as well as 16-bit data in external memory bank 0. However, care must be taken while specifying the proper starting addresses if 48-bit instructions are stored or interleaved with 16-bit data in the same memory bank.

In 16-bit wide external SRAM memory, one instruction is packed into three 16-bit memory locations, while 32-bit data occupies two memory locations.

For example, if 2k instructions are placed in 16-bit wide SRAM memory starting at the bank 0 (logical address 0x0020 0000 corresponding to physical address 0x0060 0000) and ending at logical address 0x002007FF (corresponding to physical address 0x0060 17FF), then data buffers can be placed starting at an address that is offset by 3k 16-bit words (for example, starting at 0x0060 1800).

Cache for External Instruction Fetch

To circumvent the relative difference in clock domains between the core and external memory interface (1:2 in the best case) and enable faster execution throughput, the functionality of the traditional "conflict" cache on the SHARC has been enhanced to serve as an instruction cache in external instruction fetch operations.

In previous generations of SHARC processors, the function of the conflict cache had been to cache only those instructions whose fetching conflicted with access of a data operand from memory over the PM bus. The enhancements to the cache architecture mean that the functionality of the cache remains intact for execution from internal memory whereas it behaves as instruction cache for external memory execution.

Every instruction that is fetched from external memory into the program sequencer is also simultaneously loaded into the cache.

The next time that this instruction needs to be fetched from external memory, it is first searched for in the cache. The instruction is stored using the entire 24-bit address. Figure 4-6 shows the format for storing an instruction.

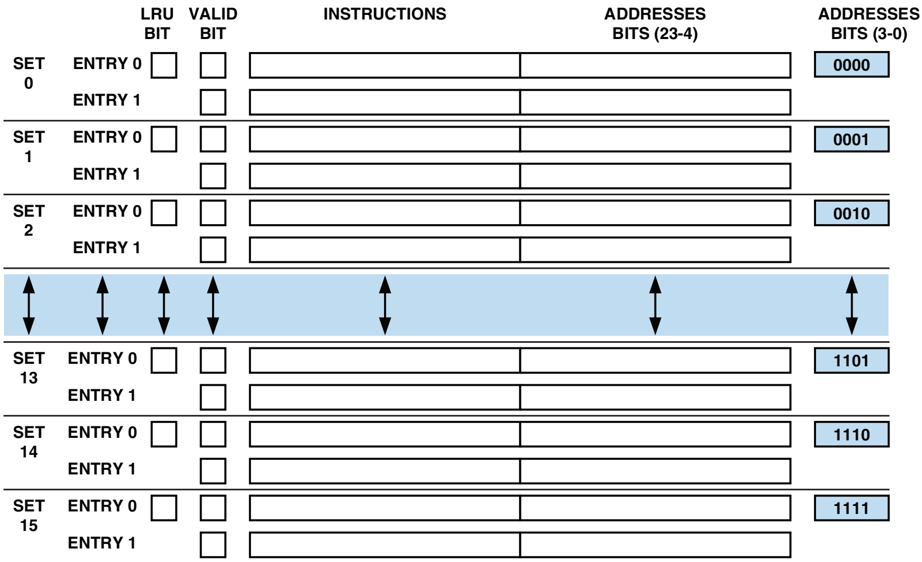

Figure 4-6. Instruction Cache Architecture

In other words, the 32-entry 2-way set-associative cache in the SHARC has been modified to act as an instruction cache when the program sequencer executes instructions from external memory, while continuing to work as the traditional conflict cache when the sequencer executes instructions located in internal memory. This context switching from conflict cache to instruction cache and vice-versa happens automatically without the need for any user intervention.

The first time that an instruction from a particular address is fetched from external memory, there is a cache miss when the sequencer looks for this instruction within the cache. Consequently, the instruction has to be fetched from external memory and a copy of instruction is stored in cache. Upon subsequent executions of this instruction, the sequencer search results in a cache hit, resulting in the instruction being fetched from cache instead of external memory. This allows for an instruction throughput that is equivalent to internal memory execution.

This context-dependent caching preserves the cache performance of the traditional SHARC conflict cache as well as significantly improving program instruction throughput for repetitive instructions such as those inside loops when executing from external memory. Analyses of typical application code examples have shown that this 32-entry instruction cache improves execution throughput by 50-80% over not having this cache.

In general, cache hits occur for all instructions which are fetched and executed multiple times (for example loops, subroutine calls, negative branches, and so on). Typical applications, such as signal processing algorithms, are ideal candidates for significant performance improvements as a result of the cache.

An important and significant result of the instruction being fetched from the cache is that it frees up the external port as well as the internal PM and DM buses for other operations such as data transfers, operand fetches, or DMA transfers.

The following example shows the innermost loop of a FIR filter.

```
lcntr=FILTER_TAPS-1, do macloop until lce;
    macloop: f12=f0*f4, f8=f8+f12, f0=dm(i0,m1), f4=pm(i9,m9);
```

In this example, if the code is stored and executed from external memory, the first time through this loop the program sequencer places the appropriate 24-bit address on the external address bus, and fetches the instruction in line 2 from external memory. While this instruction is being fetched and processed by the sequencer, it is also simultaneously stored in the internal instruction cache.

For every subsequent iteration of this loop, the instruction is fetched from the internal cache, thereby occurring in a single cycle, while freeing up the internal memory buses to fetch the data operands required for the instruction.

Previously, in the absence of the internal instruction cache, the number of cycles taken by the loop for a case of FILTER_TAPS = 16 would have been a minimum of 96 cycles over an 8-bit wide external bus (excluding any conflicts for data operand fetches). However, with the presence of the instruction cache, and assuming that the execution is from external AMI, the number of cycles is reduced to 17 core clock cycles over an 8-bit wide external bus.

As might be expected, it is important to remember that the instruction cache does not play a significant role in improving the efficiency of strictly linearly executed code from external memory.

Operating Modes

The AMI operating modes are described in the following sections.

Data Packing

The combination of the (PKDIS) and (MSWF) bits allow combinations of packing for 8/16 to 32 bits. These modes are summarized in Table 4-6.

Table 4-6. Data Packing Bit Settings (PKDIS)

Packing Mode	PKDIS Bit Setting	MSWF Bit Setting	Description
Enabled	0	0	8- or 16-bit received data is packed to 32-bit data and transmitted 32-bit data is unpacked to 2 16-bit data or 4 8-bit data. First 8- or 16-bit word read/written occupies the least significant position in the 32-bit packed word.
Enabled	0	1	8- or 16-bit received data is packed to 32-bit data and 32-bit data to be transmitted is unpacked to 2 16-bit data or 4 8-bit data. First 8- or 16-bit word read/written occupies the most significant position in the 32-bit packed word.
Disabled	1	N/A	8- or 16-bit data received is zero filled. For transmitted data only 16-bit or the 8-bit LSB part of the 32-bit data word is written to external memory.

External Access Extension

The AMI controller has an ACK Pin which can be used for external access extension. When ACK is enabled, the wait state value should be set to indicate when the processor can sample ACK after the $\overline{\text{AMI\_RD}}$/$\overline{\text{AMI\_WR}}$ edge goes low (refer to Figure 4-3 and Figure 4-4). If ACK is not enabled, the minimum value for WS is 2 (a wait state value of 0 corresponds to 32 wait cycles). If ACK is enabled, the minimum allowed value for WS is 1.

When ACK is enabled (ACKEN = 1), the processor samples the ACK signal after two wait states plus the expiration of the wait state count programmed in the AMICTLx register. It is imperative that the WS value is initialized when the acknowledge enable bit (ACKEN) is set.

Predictive Reads

The AMI controller allows two types of read access:

- predictive reads (default)
- non predictive reads

Predictive read (PREDIS bit = 0) reduces the time delay between two reads. The predictive address is generated and compared with the actual address. If they do not match, then that read data is ignored. Every last read access is therefore a duplication of the 2nd to last read with the same address. Note that this redundant read does not update the memory location.

In contrast, when no predictive read (PREDIS bit = 1) is used, the delay between two reads increases. Note that both DMA and the processor core have predictive read capability. Further note that the PREDIS bit should not be changed when the AMI is performing an access. Disabling predictive reads reduces peripheral performance.

If an access to an external FIFO is required at maximum speed, programs can also clear PREDIS (=0). The last access before a non AMI access should be a dummy AMI write access. This ensures that the last predictive read is omitted.

The PREDIS bit (bit 21) is a global bit that when set in any of the AMICTLx registers provides access to all memory banks.

SDRAM Controller (ADSP-2147x/ADSP-2148x)

The SHARC processors support a glueless interface with any of the standard SDRAMs.

Features

The SDRAM controller can support up to 254M words of SDRAM in four banks. Bank 0 can accommodate up to 62M words, and banks 1, 2, and 3 can accommodate up to 64M words each. The interface has the following additional features.

- I/O width 16-bits
- Types of 32, 64, 128, 256, and 512M bit with I/O of x4, x8, and x16
- Page sizes of 128, 256, 512, 1k, 2k words
- Variable memory address map (bank or page interleaving)
- Supports up to 254M words of SDRAM memory
- No-burst mode (BL = 1) with sequential burst type
- Open page policy—any open page is closed only if a new access in another page of the same bank occurs
- Supports multibank operation within the SDRAM
- Uses a programmable refresh counter to coordinate between varying clock frequencies and the SDRAM's required refresh rate
- Provides multiple timing options to support additional buffers between the processor and SDRAM
- Allows independent auto-refresh while the asynchronous memory interface (AMI) has control of the external port
- Supports self-refresh mode for power savings
- Predictive data accesses for higher read data throughput (read optimization)

- Supports external instruction fetch in bank 0 for ISA and VISA operation
- Supports 64-bit SIMD mode by the core
- Supports dual data instruction type 1

Pin Descriptions

The pins used by the external memory interface are described in the *ADSP-2147x SHARC Processor Data Sheet* and the *ADSP-2148x SHARC Processor Data Sheet*. Additional information on pin multiplexing can be found in "Pin Descriptions" on page 24-2.

Functional Description

The SDRAM control signals ($\overline{\text{MSx}}$, SDCKE, $\overline{\text{SDRAS}}$, $\overline{\text{SDCAS}}$, $\overline{\text{SDWE}}$, SDA10) define various operation modes to the SDRAM. Table 4-7 provides a reference to these commands and the pin state for each one.

The configuration is programmed in the SDCTL register. The SDRAM controller can hold off the processor core or DMA controller with an internally connected acknowledge signal, as controlled by refresh, or page miss latency overhead.

A programmable refresh counter is provided which generates background auto-refresh cycles at the required refresh rate based on the clock frequency used. The refresh counter period is specified with the RDIV field in the SDRAM refresh rate control register ("Refresh Rate Control Register (SDRRC)" on page A-35).

The internal 32-bit non-multiplexed address is multiplexed into:

- SDRAM column address

- SDRAM row address
- Internal SDRAM bank address

Based on the addressing mapping bit (ADDRMODE = 0) the lowest bits are mapped into the column address, next bits are mapped into the row address, and the final two bits are mapped into the internal bank address.

If ADDRMODE = 1 the lowest bits are mapped into the column address, next bits are mapped into the internal bank address and the final bits are mapped into the row address. This mapping is based on the SDCAW and SDRAW values programmed into the SDRAM control register.

The controller uses no burst mode (BL = 1) for read and write operations. This requires the controller to post every read or write address on the bus as for non-sequential reads or writes, but does not cause any performance degradation.

For read commands, there is a latency from the start of the read command to the availability of data from the SDRAM, equal to the CAS latency. This latency is always present for any single read transfer. Subsequent reads with optimization enabled do not have any latency.

SDRAM Commands

This section provides a description of each of the commands that the controller uses to manage the SDRAM interface. These commands are handled automatically by the controller. A summary of the various commands used by the on-chip controller for the SDRAM interface follows and is shown in Table 4-7 on page 4-36.

- Load mode register—initializes the SDRAM operation parameters during the power-up sequence.
- Single precharge—closes a specific internal bank depending on user code.

- Precharge all—closes all internal banks, preceding any auto-refresh command.
- Activate—activates a page in the required internal SDRAM bank
- Read/write
- Auto-refresh—causes the SDRAM to execute an internal CAS before RAS refresh.
- Self-refresh entry—places the SDRAM in self-refresh mode, in which the SDRAM powers down and controls its refresh operations internally.
- Self-refresh exit—exits from self-refresh mode by expecting auto-refresh commands from controller.
- NOP/command inhibit—no operation used to insert wait states for activate and precharge cycles

Load Mode Register

This command initializes SDRAM operation parameters. It is a part of the SDRAM power-up sequence. Load mode register uses the address bus of the SDRAM as data input. The power-up sequence is enabled by writing 1 to the SDPSS bit in the SDCTL register, subsequent SDRAM accesses initiate the power-up sequence. The exact order of the power-up sequence is determined by the SDPM bit of the SDCTL register.

The load mode register command initializes the following parameters.

- Burst length = 1, bits 2–0, always zero
- Wrap type = sequential, bit 3, always zero
- Ltmode = latency mode (CAS latency), bits 6–4, programmable in the SDCTL register
- Bits 14–7, always zero

While executing the load mode register command, the unused address pins are set to zero. During the first SDCLK cycle following load mode register, the controller issues only NOP commands to satisfy the t_{MRD} specification.

Bank Activation

The bank activation command is required for first access to any internal bank in SDRAM. This command open a row in the particular bank for the subsequent access. The value on the ADDR18-17 pins selects the bank. And the address provided on the ADDR15-0 pins selects the row. This row remains open for access until a single precharge command is issued to that bank. The single precharge command must be issued before opening a different row in the same bank.

Single Precharge

For a page miss during reads or writes in any specific internal SDRAM bank, the controller uses the single precharge command to close that bank. All other internal banks are untouched.

Precharge All

The precharge all command is given to precharge all internal banks at the same time before executing an auto-refresh. All open banks are automatically closed. This is possible since the controller uses a separate SDA10 pin which is asserted high during this command. This command precedes the auto-refresh command.

Read/Write

This command is executed if the next read/write access is in the present active page. During the read command, the SDRAM latches the column address. The delay between activate and read commands is determined by the t_{RCD} parameter. Data is available from the SDRAM after the CAS latency has been met.

In the write command, the SDRAM latches the column address. The write data is also valid in the same cycle. The delay between activate and write commands is determined by the t_{RCD} parameter.

The controller does not use the auto-precharge function of SDRAMs, which is enabled by asserting SDA10 high during a read or write command.

Figure 4-7 and Figure 4-8 show the SDRAM write and read timing of the processors.

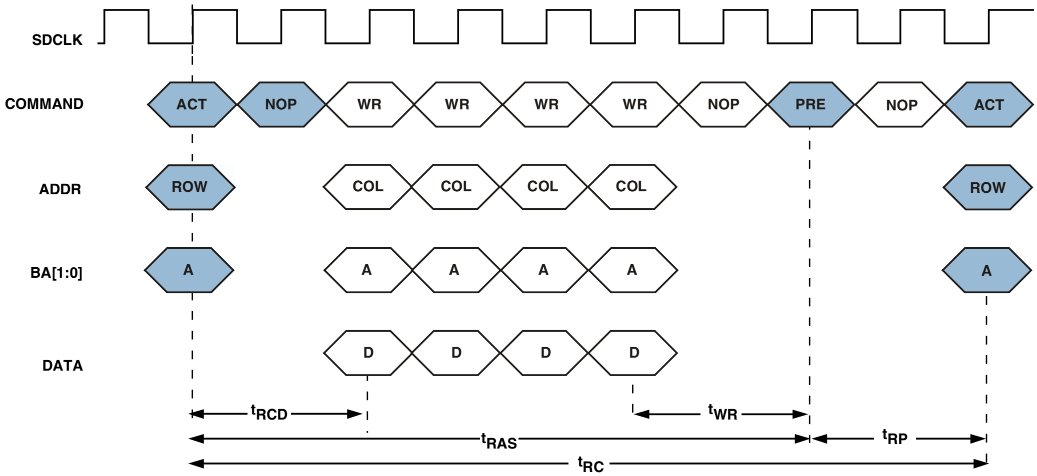

Figure 4-7. Write Timing Diagram

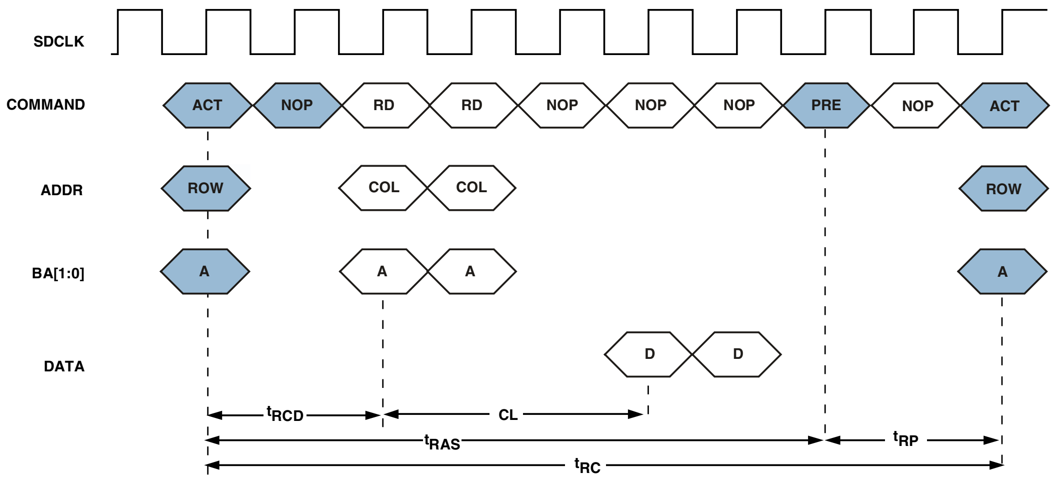

Figure 4-8. Read Timing Diagram

Auto-Refresh

The SDRAM internally increments the refresh address counter and causes a CAS before RAS (CBR) refresh to occur internally for that address when the auto-refresh command is given. The controller generates an auto-refresh command after the controller refresh counter times out. The RDIV value in the SDRAM refresh rate control register (SDRRC) must be set so that all addresses are refreshed within the t_{REF} period specified in the SDRAM timing specifications.

Before executing the auto-refresh command, the controller executes a pre-charge all command to all external banks. The next activate command is not given until the t_{RFC} specification ($t_{RFC} = t_{RAS} + t_{RP}$) is met. Auto-refresh commands are also issued by the controller as part of the power-up sequence and after exiting self-refresh mode.

No Operation/Command Inhibit

The no operation (NOP) command to the SDRAM has no effect on operations currently in progress. The command inhibit command is the same as a NOP command; however, the SDRAM is not chip-selected. When the controller is actively accessing the SDRAM, but needs to insert additional commands with no effect, the NOP command is given. When the controller is not accessing any SDRAM external banks, the command inhibit command is given.

Command Truth Table

Table 4-7 provides the bit states of the SDRAM for specific SDRAM commands. Note that an X means do not care.

Table 4-7. SDRAM Pin States During SDRAM Commands

Command	SDCKE (n–1)	SDCKE (n)	MS3–0	SDRAS	SDCAS	SDWE	SDA10	Addresses
Mode register set	1	1	0	0	0	0	Opcode	Opcode
Activate	1	1	0	0	1	1	Valid	Valid
Read	1	1	0	1	0	1	0	Valid
Single Precharge	1	1	0	0	1	0	0	Valid
Precharge all	1	1	0	0	1	0	1	X
Write	1	1	0	1	0	0	0	Valid
Auto-refresh	1	1	0	0	0	1	X	X
Self-refresh entry	1	0	0	0	0	1	X	X
Self-refresh	0	0	X	X	X	X	X	X
Self-refresh exit	0	1	1	X	X	X	X	X
Nop	1	1	0	1	1	1	X	X
Inhibit	1	1	1	X	X	X	X	X

Refresh Rate Control

The SDRAM refresh rate control register provides a flexible mechanism for specifying auto-refresh timing. The controller provides a programmable refresh counter which has a period based on the value programmed into the lower 12 bits of this register. This coordinates the supplied clock rate with the SDRAM device's required refresh rate.

The delay (in number of `SDCLK` cycles) between consecutive refresh counter time-outs must be written to the `RDIV` field. A refresh counter time-out triggers an auto-refresh command to the external SDRAM bank.

Programs should write the RDIV value to the SDRRC register before the SDRAM power-up sequence is triggered. Change this value only when the controller is idle as indicated in the SDSTAT register.

To calculate the value to write to the SDRRC register, use the following equation.

$RDIV = (SDCLK \times t_{REFI}) - (t_{RAS} + t_{RP})$ where:

Where:

- SDCLK = SDRAM system clock frequency
- t_{REFI} = SDRAM maximum average auto refresh period (in μs). (Note $t_{REFI} = t_{REF}$/Number of row addresses)
- t_{RAS} = Active to precharge time (SDRAS bit in the SDCTL register) in number of clock cycles
- t_{RP} = RAS to precharge time (SDRP bit in the SDCTL register) in number of clock cycles

This equation calculates the number of clock cycles between required refreshes and subtracts the required delay between bank activate commands to the same bank ($t_{RC} = t_{RAS} + t_{RP}$). The t_{RC} value is subtracted, so that in the case where a refresh time-out occurs while an SDRAM cycle is active, the SDRAM refresh rate specification is guaranteed to be met. The result from the equation is always rounded down to an integer. Below is an example of the calculation of RDIV for a typical SDRAM in a system with a 133 MHz SDRAM clock.

$$RDIV = \left(\frac{133 \times (10^{6}) \times 64 \times (10^{-3})}{8192}\right) - (6 + 3) = 1030$$

- f_{SDCLK} = 133 MHz
- t_{REF} = 64 ms

- NRA = 8192 row addresses
- t_{RAS} = 6
- t_{RP} = 3

This means RDIV is 0x406 (hex) and the SDRAM refresh rate control register is written with 0x406.

The RDIV value must be programmed to a nonzero value if the SDRAM controller is enabled. When RDIV = 0, operation of the SDRAM controller is not supported and can produce undesirable behavior.

Some SDRAM vendors use separate timing specifications for the row active time (t_{RC}) and row refresh time (t_{RFC}). The controller ignores the t_{RFC} spec. For auto-refresh, it uses the equation $t_{RC} = t_{RAS} + t_{RP}$. However since both timing specifications must meet (especially for extended temperature range) the modification of the t_{RAS} specification resolves the timing equation without performance degradation ($t_{RFC} = t_{RAS} + t_{RP}$).

Internal SDRAM Bank Access

The following sections describe the different scenarios for SDRAM bank access.

Single Bank Access

The controller keeps only one page open at a time if all subsequent accesses are to the same row or another row in the same bank.

Multi-Bank Access

The processors are capable of supporting multi-bank operation, thus taking advantage of the SDRAM architecture.

Operation using single versus multi-bank accesses depends only on the address to be posted to the device, it is NOT an operation mode.

Any first access to SDRAM bank (A) forces an activate command before a read or write command. However, if any new access falls into the address space of the other banks (B, C, or D) the controller leaves bank (A) open and activates any of the other banks (B, C, or D). Bank (A) to bank (B) active time is controlled by $t_{RRD} = t_{RCD} + 1$. This scenario is repeated until all four banks (A–D) are opened and results in an effective page size of up to four pages. This is because the absence of latency allows switching between these open pages (as compared to one page in only one bank at a time). Any access to any closed page in any opened bank (A–D) forces a precharge command only to that bank. If, for example, two external port DMA channels are pointing to the same internal SDRAM bank, this always forces precharge and activation cycles to switch between the different pages. However, if the two external port DMA channels are pointing to different internal SDRAM banks, there is no additional overhead. See Figure 4-9.

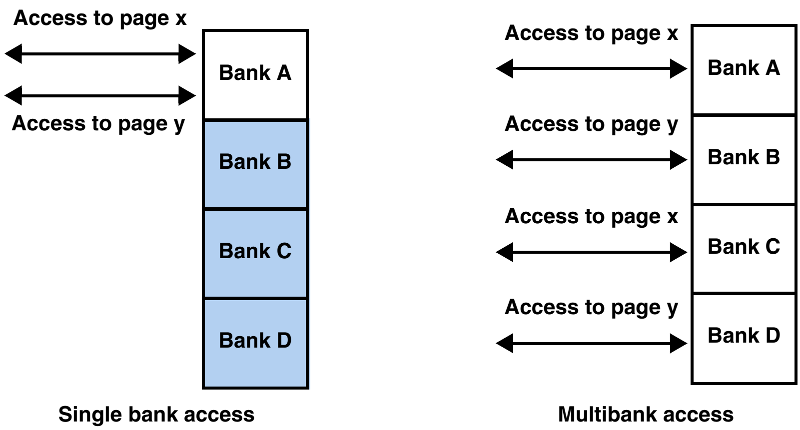

Figure 4-9. Single Versus Multibank Access

Furthermore the controller supports four external memory selects containing each SDRAM. All external banks ($\overline{\text{MS3-0}}$) provide multi-bank support, so the maximum number of open pages is 4 × 4 = 16 pages.

Multi-bank access reduces precharge and activation cycles by mapping opcode/data among different internal SDRAM banks driven by the A18–17 pins and external memory selects ($\overline{\text{MSx}}$).

Multi-Bank Operation with Data Packing

A logical address corresponds to 2 physical addresses when X16DE = 1. Consequently a physical address (for example of 512 x 16 page size) translates into a logical address of 256 × 16 words to satisfy the packing. Accordingly, all row addresses are shifted by 2.

A populated SDRAM of 2M × 16 × 4 with a 512 word page size, connected to external bank 0 and using bank interleaving (SDADDRMODE bit = 0) has the following logical map.

```
0x200000 logical start address int bankA
0x2000FF logical end address int bankA

0x300000 logical start address int bankB
0x3000FF logical end address int bankB

0x400000 logical start address int bankC
0x4000FF logical end address int bankC

0x500000 logical start address int bankD
0x5000FF logical end address int bankD
```

The same SDRAM with page interleaving (SDADDRMODE bit = 1) has the following address map:

```
0x200000 logical start address int bankA
0x2000FF logical end address int bankA

0x200100 logical start address int bankB
0x2001FF logical end address int bankB

0x200200 logical start address int bankC
0x2002FF logical end address int bankC

0x200300 logical start address int bankD
0x2003FF logical end address int bankD
```

Timing Parameters

The controller requires many timing settings in order to correctly access the SDRAM devices. Those that are user configurable can be found in "SDRAM Registers" on page A-31.

Fixed Timing Parameters

The timing specifications below are fixed by the controller.

- t_{MRD} (mode register delay). Required delay time to complete the mode register write. This parameter is fixed to 2 cycles.
- t_{RRD} (row active A to row active B delay). Required delay between two different SDRAM banks. This parameter is fixed to t_{RCD} +1 cycle.
- t_{RC} (row access cycle). Required delay time to open and close a single row. This parameter is fixed to $t_{RC} = t_{RAS} + t_{RP}$ cycles.
- t_{RFC} (row refresh cycle). Required delay time to refresh a single row. This parameter is fixed to $t_{RFC} = t_{RC}$ cycles.
- t_{XSR} (exit self-refresh mode). Required delay to exit the self-refresh mode. This parameter is fixed to $t_{XSR} = t_{RC}$ cycles.

Data Mask

The SDRAM controller provides one `DQM` pin (`SDDQM`), all SDRAM `DQM` pins could be connected to `SDDQM` pin. The `SDDQM` pin is driven high from reset deassertion until SDRAM initialization completes, after that it's driven low irrespective of whether any accesses occur.

Note that some manufacturer's require keeping `DQM` high during the power-up initialization sequence.

Resetting the Controller

Like any other peripheral, the SDRAM controller can be reset by a hard or a soft reset. A hard reset puts the PLL in bypass mode where the SDRAM clock runs at a lower frequency.

A hard or soft reset also causes data loss, and programs need to re-initialize SDRAM before it can be used again.

Running reset (RESETOUT pin as an input) does not reset the SDRAM controller.

16-Bit Data Storage and Packing

The processors use logical addressing when an external memory smaller than 32 bits is used. Logical addresses require multiple external addresses seen by the memory correspond to a single internal address, depending on the width of the memory being accessed.

The external physical address map is shown in Table 4-3.

Table 4-8. SDRAM Address Memory Map

Bus Width	External Memory Bank	Internal Logical Address (supported memory map)	External Physical Address (on ADDR23–0)
16-bit	0	0x0020_0000 – 0x007F_FFFF	0x40_0000 – 0xFF_FFFF
16-bit	1, 2, 3	0x0400_0000 – 0x047F_FFFF 0x0800_0000 – 0x087F_FFFF 0x0C00_0000 – 0x0C7F_FFFF	0x00_0000 – 0xFF_FFFF

External Instruction Fetch

The processors support direct fetch of ISA instructions from external memory, using the 8/16-bit external port. Fetching is supported from external memory space bank 0 which is selected by $\overline{\text{MS0}}$.

Interrupt Vector Table (IVT)

The interrupt vector table can be located in the internal ROM (0x80 000, IIVT bit = 0) or internal RAM (0x8C 000, IIVT bit = 1) based on the selected boot mode. However for all boot modes except the reserved boot mode, the default IIVT bit setting is 1 (SYSCTL).

Therefore, if instruction fetch from external memory is desired at reset, the program needs to set up the appropriate interrupt vector tables in internal memory as part of the boot-up code before beginning to fetch these instructions.

When an unmasked interrupt occurs and is serviced, program execution automatically jumps to the location of the corresponding interrupt vector table in internal memory. Upon returning from the interrupt, the sequencer resumes fetching instructions from external memory because locating the IVT in external memory is not supported.

Fetching ISA Instructions From External Memory

The SDRAM controller along with the processor core incorporates appropriate enhancements so that instruction code can be fetched from the SDRAM at the maximum possible throughput. Throughput is limited only by the SDRAM when the code is non sequential.

The address map for code is same as for data. Each address refers to a 32-bit word. Any address produced by the sequencer is checked to deter-mine if it falls in the external memory and if so, the SDRAM controllers initiate access to the SDRAM. Because the sequencer address bus is limited to 24 bits, only part of the external memory address area can be used to store code. As explained in the following section, the address generated by the sequencer undergoes translation to produce a physical address, since the SDRAM data bus width is less than 48 bits.

Whether fetching ISA or VISA instructions, the IVT needs to be placed in the ISA normal word space (NW).

Instruction Packing

Any address produced by the sequencer which falls in external memory is first translated into the physical address in external memory based on the actual data bus width of external memory as shown in Figure 4-8.

The controller completes the required number of accesses from consecutive locations for returning a 48-bit word instructions. For a 16-bit SDRAM bus, it performs three accesses.

Only bank0 can be populated for external instruction fetch.

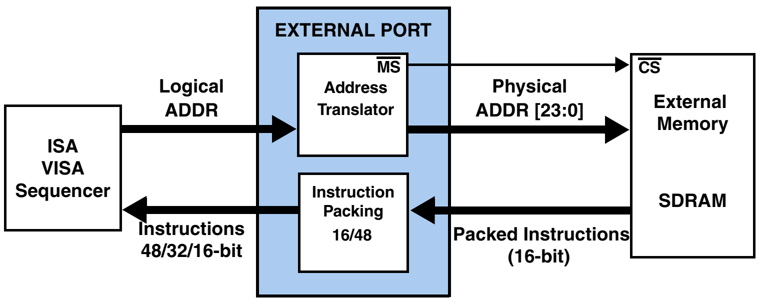

Figure 4-10. Logical Versus Physical Addresses

16-Bit Instruction Storage and Packing

In Table 4-9 P = 0xE00000. Therefore, the total number of external memory instructions for a 16-bit wide SDRAM memory is 14 million.

Table 4-9. Logical Versus Physical Address Mapping, 16-Bit SDRAM

Logical ISA Normal Word Address, Program Sequencer	Physical Address, External Bus	Data 15–0
0x20 0000	0x60 0000	Instr0[15:0]
	0x60 0001	Instr0[31:16]
	0x60 0002	Instr0[47:32]
0x20 0001	0x60 0003	Instr1[15:0]
	0x60 0004	Instr1[31:16]
	0x60 0005	Instr1[47:32]
0x20 0002	0x60 0006	Instr2[15:0]
	0x60 0007	Instr2[31:16]
	0x60 0008	Instr2[47:32]
...	...	
0x5F FFFF	0x11F FFFD	InstrN[15:0]
	0x11F FFFE	InstrN[31:16]
	0x11F FFFF	InstrN[47:32]

Fetching VISA Instructions From External Memory

The SHARC processors support fetching instructions from external SDRAM or DDR2 memory. These instructions may be stored either as traditional 48-bit SHARC ISA instructions, or as VISA instructions. There is an overhead incurred when fetching data in general directly from external memory owing to inherent latencies and overheads associated with accessing SDRAM/DDR2 memory. Additionally, there are latencies involved with accessing non-sequential VISA instructions from external memory because of the width of the external SDRAM/DDR2 data bus (instructions have to be fetched as 16-bit units).

VISA mode execution is not supported through the asynchronous memory interface (AMI).

In VISA operation, the sequencer fetches 3 x 16-bit of data which decodes in one, two or three instructions. For more information on VISA operation refer to *SHARC Processor Programming Reference*.

Just as the same physical internal memory on the processors can be accessed and addressed in many different ways, the external memory space can also be viewed either as logical or physical addresses. To support VISA in external memory, the external memory address range has been divided into two ranges:

- Normal word – 0x20 0000 to 0x5F FFFF (ISA)
- Short word – 0x60 0000 to 0xFF FFFF (VISA)

When the processor accesses any instruction from the external normal word space, the instruction is deemed to have the traditional SHARC instruction encoding. When the processor accesses any instruction from external short word space, the instruction is deemed to have the new VISA instruction encoding.

For a x16 memory, the external port interface effectively translates the addresses in range 0x20 0000 – 0x5F FFFF to 0x60 0000 – 0x11F FFFF, when accessing 48-bit instructions in legacy (ISA) encoding from external memory. The external port performs three accesses to form one 48-bit word before forwarding it to the IAB.

Note that the external port interface passes the addresses in the range 0x60 0000 – 0xFF FFFF as is to external memory. As in the previous case, the external port accesses three short words to return a 48-bit word to the IAB for each access requested by the sequencer. The short words for a VISA section of code are packed in such a way that lowest of the addresses pertaining to a given instruction has the most significant short word of that instruction and the highest address has the least significant short word (see Figure 4-11).

This packed instruction, when fetched in VISA space, is internally rotated before it reaches the instruction alignment buffer and cache. However, if this instruction is fetched in ISA space, this rotation does not occur and the instruction is cached without rotation. Eventually, if the cache gives out the instruction to the processor, it is a corrupted instruction.

To avoid this, code should not be placed in either of the following ranges: 0x5F FFFD – 0x5F FFFF and 0x60 0000 – 0x60 0008.

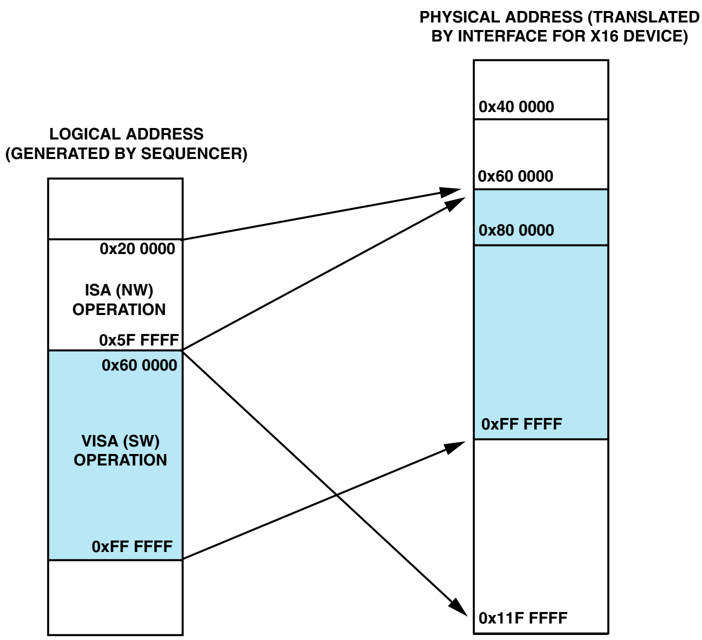

Figure 4-11. Translation of Logical to Physical External Memory Addressing

Mixing Instructions and Data in External Bank 0

It is possible to store both 48-bit instructions as well as 16-bit data in external memory bank 0. However, care must be taken while specifying the proper starting addresses if 48-bit instructions are stored or interleaved with 16-bit data in the same memory bank.

In 16-bit wide external SDRAM/DDR2 memory, one instruction is packed into three 16-bit memory locations, while 32-bit data occupies two memory locations.

For example, if 2k instructions are placed in 16-bit wide SDRAM/DDR2 memory starting at the bank 0 (logical address 0x0020 0000 corresponding to physical address 0x0060 0000) and ending at logical address 0x002007FF (corresponding to physical address 0x0060 17FF), then data buffers can be placed starting at an address that is offset by 3k 16-bit words (for example, starting at 0x0060 1800).

Cache for External Instruction Fetch

To circumvent the relative difference in clock domains between the core and external memory interface (1:2 in the best case) and enable faster execution throughput, the functionality of the traditional "conflict" cache on the SHARC has been enhanced to serve as an instruction cache in external instruction fetch operations.

In previous generations of SHARC processors, the function of the conflict cache had been to cache only those instructions whose fetching conflicted with access of a data operand from memory over the PM bus. The enhancements to the cache architecture mean that the functionality of the cache remains intact for execution from internal memory whereas it behaves as instruction cache for external memory execution.

Every instruction that is fetched from external memory into the program sequencer is also simultaneously loaded into the cache.

The next time that this instruction needs to be fetched from external memory, it is first searched for in the cache. The instruction is stored using the entire 24-bit address. Figure 4-6 shows the format for storing an instruction.

In other words, the 32-entry 2-way set-associative cache in the SHARC has been modified to act as an instruction cache when the program sequencer executes instructions from external memory, while continuing to work as the traditional conflict cache when the sequencer executes instructions located in internal memory. This context switching from conflict cache to instruction cache and vice-versa happens automatically without the need for any user intervention.

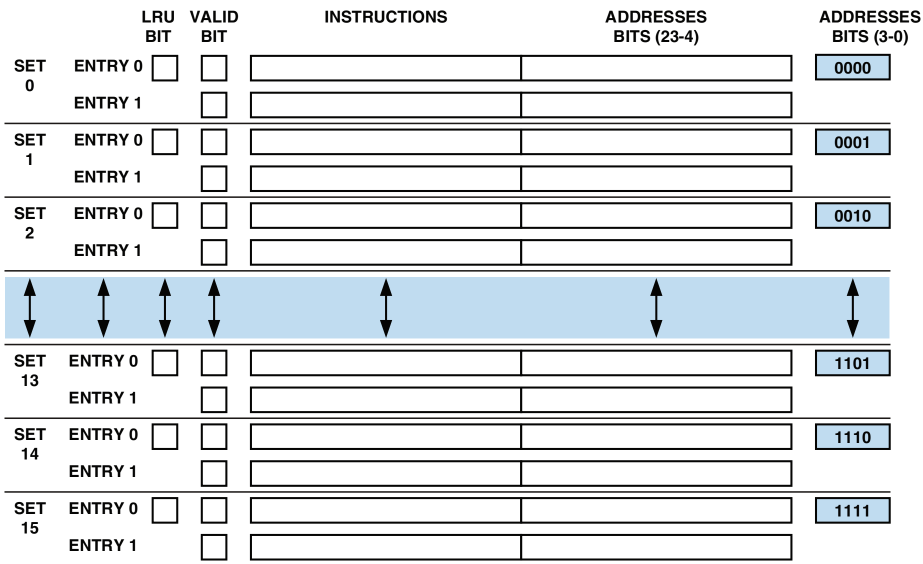

Figure 4-12. Instruction Cache Architecture

The first time that an instruction from a particular address is fetched from external memory, there is a cache miss when the sequencer looks for this instruction within the cache. Consequently, the instruction has to be fetched from external memory and a copy of instruction is stored in cache.

Upon subsequent executions of this instruction, the sequencer search results in a cache hit, resulting in the instruction being fetched from cache instead of external memory. This allows for an instruction throughput that is equivalent to internal memory execution.

This context-dependent caching preserves the cache performance of the traditional SHARC conflict cache as well as significantly improving program instruction throughput for repetitive instructions such as those inside loops when executing from external memory. Analyses of typical application code examples have shown that this 32-entry instruction cache improves execution throughput by 50-80% over not having this cache.

In general, cache hits occur for all instructions which are fetched and executed multiple times (for example loops, subroutine calls, negative branches, and so on). Typical applications, such as signal processing algorithms, are ideal candidates for significant performance improvements as a result of the cache.

An important and significant result of the instruction being fetched from the cache is that it frees up the external port as well as the internal PM and DM buses for other operations such as data transfers, operand fetches, or DMA transfers.

The following example shows the innermost loop of a FIR filter.

```
lcntr=FILTER_TAPS-1, do macloop until lce;
    macloop: f12=f0*f4, f8=f8+f12, f0=dm(i0,m1), f4=pm(i9,m9);
```

In this example, if the code is stored and executed from external memory, the first time through this loop the program sequencer places the appropriate 24-bit address on the external address bus, and fetches the instruction in line 2 from external memory. While this instruction is being fetched and processed by the sequencer, it is also simultaneously stored in the internal instruction cache.

For every subsequent iteration of this loop, the instruction is fetched from the internal cache, thereby occurring in a single cycle, while freeing up the internal memory buses to fetch the data operands required for the instruction.

Previously, in the absence of the internal instruction cache, the number of cycles taken by the loop for a case of `FILTER_TAPS` = 16 would have been a minimum of 96 SDRAM cycles over a 8-bit wide external bus (excluding any conflicts for data operand fetches). However, with the presence of the instruction cache, and assuming that the execution is from external AMI, the number of cycles is reduced to 17 core clock cycles over a 8-bit wide external bus.

As might be expected, it is important to remember that the instruction cache does not play a significant role in improving the efficiency of strictly linearly executed code from external memory.

Address Versus SDRAM Types

Table 4-10 provides addressing for various sizes of SDRAM memory.

Table 4-10. Translation of Logical to Physical Addressing for SDRAM

DDR2 Device	Physical Address Range Mapped to Memory Device	Mapping Between External Port Address Range and Memory Device
32 Mb (x16)	0x60 0000 – 0x7F FFFF	0x00 0000 – 0x1F FFFF
64 Mb (x16)	0x60 0000 – 0x9F FFFF	0x020 0000 – 0x3F FFFF 0x000 0000 – 0x1F FFFF
128 Mb (x16)	0x60 0000 – 0xDF FFFF	0x020 0000 – 0x7F FFFF 0x000 0000 – 0x1F FFFF
256 Mb (x16)	0x60 0000 – 0x15F FFFF	0x020 0000 – 0xFF FFFF 0x000 0000 – 0x1F FFFF

Operating Modes

The following sections provide on the operating modes of the SDRAM interface.

Address Mapping

To access SDRAM, the controller multiplexes the internal 32-bit non-multiplexed address into three portions:

- Row address bits
- Column address bits
- Bank address bits

The non multiplexed address that is seen from the core/DMA is referred to as IA31–0 in the following sections.

Address Translation Options

To provide flexible addressing, the `SDADDRMODE` bit (bit 31) in the `SDCTL0` register is used to select the address mapping scheme—page interleaving or bank interleaving (default).

Page Interleaving Map

Programming the `SDADDRMODE` bit to 1 selects the page interleaving scheme. In this scheme consecutive pages fall in consecutive banks. The bank address bits follow the most significant column address bits. This is shown in Figure 4-13.

One advantage of the page interleaving is that the effective page size is up to four pages (assuming four banks activated) and all the addresses are sequential. If using delay line DMA mode, the addresses for a long delay line are all sequential, simplifying the addressing. Moreover, SDRAM sequential addressing provides maximum performance.

Page interleaving is not supported with 2 bank devices.

Bank Interleaving Map

Programming the `SDADDRMODE` bit to 0 selects the bank interleaving scheme. In this scheme consecutive pages sit in the same bank. The bank address bits follow the most significant row address bits. This is shown in Figure 4-13.

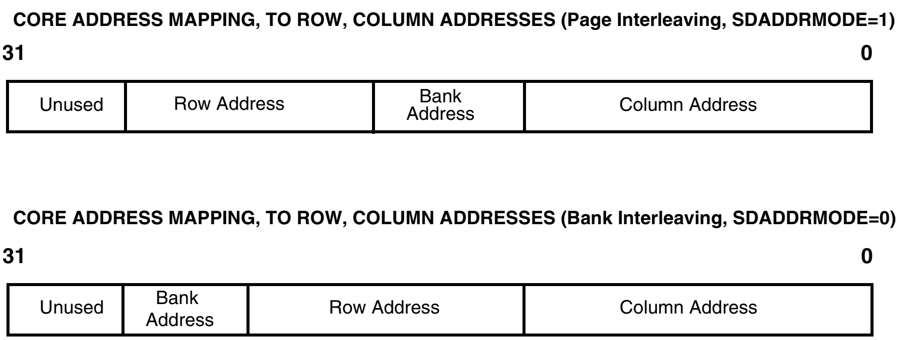

Figure 4-13. Core Address Mapping—Page and Bank Interleaving

One advantage of bank interleaving is that the effective page size is also up to four pages (assuming that four banks are activated) but the addresses of the four pages are not sequential. If using two external port DMAs pointing to the SDRAM space, this scheme has the advantage where every bank uses single DMA buffer addressing.

For two-banked SDRAMs, connect BA with A17. Note that page interleaving is not supported with 2 bank devices.

The mapping of the addresses depends on the row address width (SDRAW), column address width (SDCAW), and the address mode bit (SDADDRMODE) setting.

Address Width Settings

Address width settings can be configured as shown in Table 4-17.

Table 4-11. External Memory Address Bank Decoding

IA[27]	IA[26]	External Bank
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3

Number of Internal Banks. The controller assumes the SDRAM is comprised of four bank devices. However, SDRAM can use two bank devices by not connecting the ADDR18 pin.

Row Address Width (SDRAW). These bits in the SDCTL register determine the row width of the SDRAM. The SDRAW bits can be programmed for row widths of 8 to 15.

Column Address Width (SDCAW). The SDRAM memory control register also includes external bank specific programmable parameters. The external bank can be configured for a different SDRAM size. The SDRAM controller determines the internal SDRAM page size from the PGSZ128 and SDCAW parameters. Page sizes of 128, 256, 512, 1K, 2K words are supported.

16-Bit Address Mapping

Even if the external data width is 16 bits, the processor supports only 32-bit data accesses. If X16DE is enabled (=1) the controller performs two 16-bit accesses to get and place 32-bit data. The controller takes the IA address and appends one extra bit to the LSB to generate the address externally.

In the following sections and in Table 4-12 and Table 4-13, the mapping of internal addresses to the external addresses is discussed. The mapping of the addresses depends on the address mode (SDADDRMODE bit) on row address width (SDRAW), and on column address width (SDCAW).

The X16DE bit must always be set.

For example, if the processor core requests address 0x200–0000 for a 32-bit access, the controller performs two 16-bit accesses at 0x400–0000 and 0x400–0001, using $\overline{\text{MS0}}$ to get one 32-bit data word.

The column and row addresses seen by 16-bit SDRAMs is shown in Table 4-12 where SDADDRMODE = 1, X16DE = 1, SDRAW2-0 = 101 (13 bits), and SDCAW1-0 = 10 (10 bits).

Table 4-12. Page Interleaving Map (1K Page Size)

Pin	Column Address	Row Address	Bank Address	Pins of SDRAM
A[18]			IA[10]	BA[1]
A[17]			IA[9]	BA[0]
A[13]				
A[12]		IA[23]		A[12]
A[11]		IA[22]		A[11]
SDA10	1'b0	IA[21]		A[10]
A[9]	IA[8]	IA[20]		A[9]
A[8]	IA[7]	IA[19]		A[8]

Table 4-12. Page Interleaving Map (1K Page Size) (Cont'd)

Pin	Column Address	Row Address	Bank Address	Pins of SDRAM
A[7]	IA[6]	IA[18]		A[7]
A[6]	IA[5]	IA[17]		A[6]
A[5]	IA[4]	IA[16]		A[5]
A[4]	IA[3]	IA[15]		A[4]
A[3]	IA[2]	IA[14]		A[3]
A[2]	IA[1]	IA[13]		A[2]
A[1]	IA[0]	IA[12]		A[1]
A[0]	1/0	IA[11]		A[0]

Table 4-13 where `SDADDRMODE` = 0, `X16DE` = 1, `SDRAW2-0` = 100 (12 bits), and `SDCAW1-0` = 11 (11 bits).

Table 4-13. Bank Interleaving Map (2K Page Size)

Pin	Column Address	Row Address	Bank Address	Pins of SDRAM
A[18]			IA[23]	BA[1]
A[17]			IA[22]	BA[0]
A[13]				
A[12]				
A[11]	IA[9]	IA[21]		A[11]
SDA10	1'b0	IA[20]		A[10]
A[9]	IA[8]	IA[19]		A[9]
A[8]	IA[7]	IA[18]		A[8]
A[7]	IA[6]	IA[17]		A[7]
A[6]	IA[5]	IA[16]		A[6]
A[5]	IA[4]	IA[15]		A[5]

Table 4-13. Bank Interleaving Map (2K Page Size) (Cont'd)

Pin	Column Address	Row Address	Bank Address	Pins of SDRAM
A[4]	IA[3]	IA[14]		A[4]
A[3]	IA[2]	IA[13]		A[3]
A[2]	IA[1]	IA[12]		A[2]
A[1]	IA[0]	IA[11]		A[1]
A[0]	1/0	IA[10]		A[0]

Parallel Connection of SDRAMs

To specify a SDRAM system, multiple possibilities are given based on the different memory sizes. For a 16-bit I/O capability, the following can configured.

- 1 x 16-bit/page 512 words
- 2 x 8-bit/page 1k words
- 4 x 4-bit/page 2k words

The SDRAM's page size is used to determine the system you select. All three systems have the same external bank size, but different page sizes. Note that larger page sizes, allow higher performance but larger page sizes require more complex hardware layouts.

Even if connecting SDRAMs in parallel, the controller always considers the cluster as one external SDRAM bank because all address and control lines feed the parallel parts as shown in Figure 4-15.

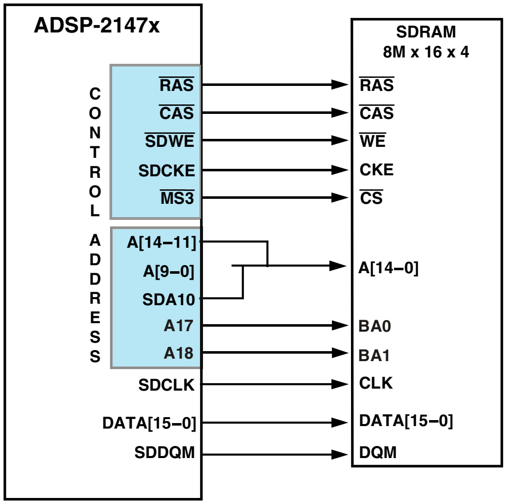

Figure 4-14. Single Processor System With SDRAM

Buffering Controller for Multiple SDRAMs

If using multiples SDRAMs or modules, the capacitive load will exceed the controller's output drive strength. In order to bypass this problem an external latch can be used for decoupling by setting the SDBUF (bit 23). This adds a cycle of data buffering to read and write accesses. An example single processor system is shown in Figure 4-15.

SDRAM Read Optimization

To achieve better performance, read addresses can be provided in a predictive manner to the SDRAM memory. This is done by setting (=1) SDROPT (bit 16) and correctly configuring the SDMODIFY bits (20–17) in the SDRRC register according to the core's DAG modifier or the DMA's modify parameter register.

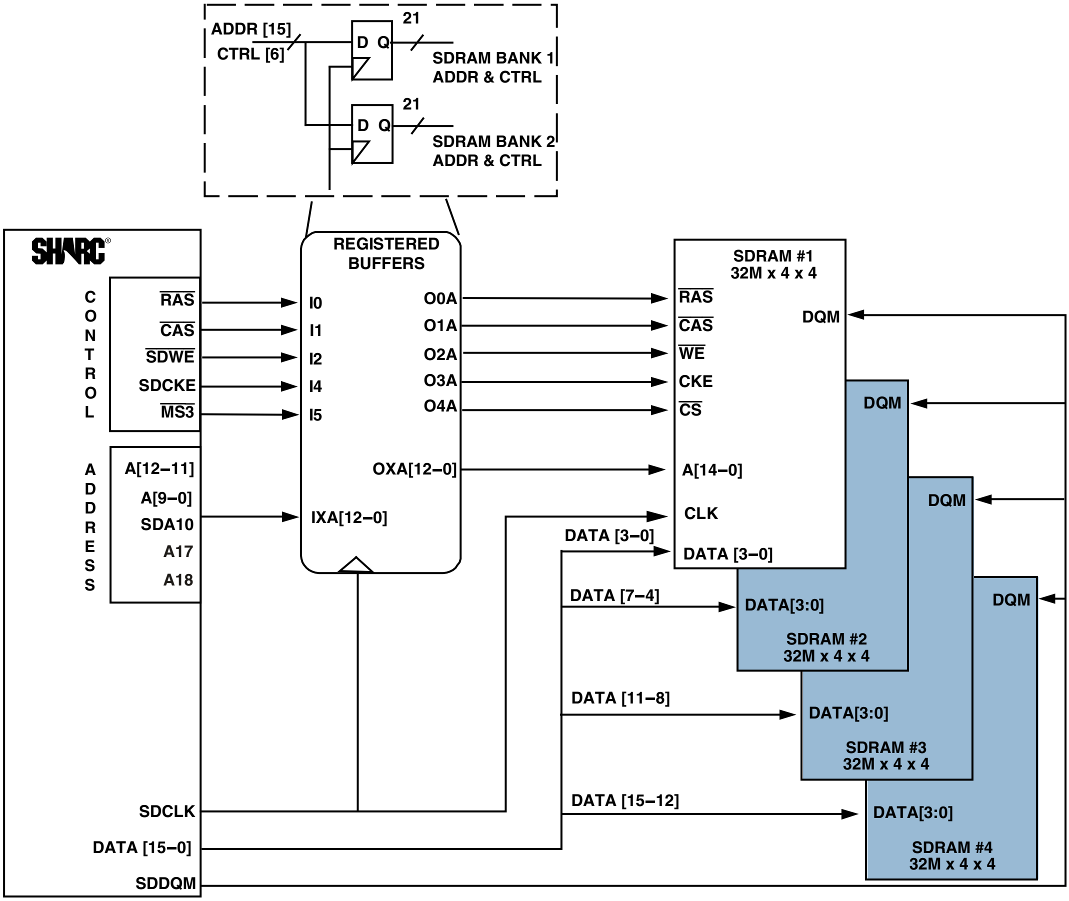

Figure 4-15. Uniprocessor System With Multiple Buffered SDRAM Devices

The predictive address given to the memory depends on the SDMODIFY bit values. For example, if the DAG modifier = 2, the SDMODIFY value should also be 2, in which case the address + 2 is the predictive value provided to the SDRAM address pins. Programs may choose to determine whether read optimization is used or not. If read optimization is disabled, then each read takes 7 cycles for a CAS latency of 3, even for sequential reads.

With read optimization enabled, 32 sequential reads, with offsets ranging from 0 to 15, take only 37 SDCLK cycles. Read optimization should not be enabled while reading at the external bank boundaries. For example, if SDMODIFY = 1, then 32 locations in the boundary of the external banks should not be used. These locations can be used without optimization enabled. If SDMODIFY = 2, then 64 locations cannot be used at the boundaries of the external bank (if it is fully populated).

Use read optimization for core and DMA, with a constant modifier to achieve better performance. With multiple channels running with ping-pong accesses, use arbitration freezing to get better throughput.

By default, the read optimization is enabled (SDROPT = 1) with a modifier of 1 (SDMODIFY = 1). Read optimization assumes that the SDRAM pointer has a constant modifier. For non-sequential accesses, turning off optimization provides better results.

Core Accesses

Any break of sequential reads of full page accesses can cause a throughput loss due to a maximum of four extra reads (eight 16-bit reads). Listing 4-1 shows how to achieve maximum core access throughput. Any cycle between consecutive reads to an SDRAM address results in non-sequential reads.

Listing 4-1. Maximum Throughput Using Sequential Reads

```
ustat1=dm(SDCTL);
bit set ustat1 SDROPT|SDMODIFY1;
dm(SDCTL)=ustat1;
nop;
I0 = sdram_addr;
M0 = 1;
Lcntr = 512, do(PC,1) until lce;
R0 = R0 + R1, R0 = dm (I0, M0);
```

The example shows read optimization can be used efficiently using core accesses. All reads are on the same page and it takes 1184 cycles to perform 512 reads.

Without read optimization, 512 reads use 6144 processor cycles if all of the reads are on the same page. With read optimization (Listing 4-2), 512 reads take 7168 cycles, due to the breaking of sequential reads.

Listing 4-2. Interrupted Reads With Read Optimization

```
ustat1=dm(SDCTL);
bit set ustat1 SDROPT|SDMODIFY2;
dm(SDCTL)=ustat1;
nop;
I0 = sdram_addr;
M0 = 2;

Lcntr = 512, do(PC,2) until lce;
R0 = R0 + R1, R0 = dm (I0, M0);
NOP;
```

DMA Access

Listing 4-3 shows an example of external port DMA using read optimization.

Listing 4-3. EPDMA With Read Optimization

```
ustat1=dm(SDCTL);
bit set ustat1 SDROPT|SDMODIFY2;
dm(SDCTL)=ustat1;
nop;

r0=DFLSH;
dm(DMAC1)=r0;
r0=intmem;      dm(IIEP1)=r0;
r0=2;           dm(IMEP1)=r0;
r0=N;           dm(ICEP1)=r0;
r0=2;           dm(EMEP1)=r0;
r0=extmem;      dm(EIEP1)=r0;
r0=DEN;
dm(DMAC1)=r0;
```

Notes on Read Optimization

The core and the DMA engine take advantage of the major improvements during reads using read optimization. However, in situations where both the core and DMA need to read from different internal memory banks with different modifiers at the same time, programs need to choose whether or not to use optimization. Note that from a throughput prospective, external port arbitration also is a factor. A good rule is that the requester with the higher priority should have the same modifier as `SDMODIFY`. In other words, if DMA has a higher priority over the core, then the DMA modifier should match the `SDMODIFY` setting.

Self-Refresh Mode

This mode causes refresh operations to be performed internally by the SDRAM, without any external control. This means that the controller does not generate any auto-refresh cycles while the SDRAM is in self-refresh mode.

Self-refresh entry—Self-refresh mode is enabled by writing a 1 to the SDSRF bit of the SDRAM memory control register (SDCTL). This deasserts the SDCKE pin and puts the SDRAM in self-refresh mode if no access is currently underway. The SDRAM remains in self-refresh mode for at least t_{RAS} and until an internal access (read/write) to SDRAM space occurs.

Self-refresh exit—When any SDRAM access occurs, the controller asserts SDCKE high which causes the SDRAM to exit from self-refresh mode. The controller waits to meet the t_{XSR} specification ($t_{XSR} = t_{RAS} + t_{RP}$) and then issues an auto-refresh command. After the auto-refresh command, the controller waits for the t_{RFC} specification ($t_{RFC} = t_{RAS} + t_{RP}$) to be met before executing the activate command for the transfer that caused the SDRAM to exit self-refresh mode. Therefore, the latency from when a transfer is received by the controller while in self-refresh mode, until the activate command occurs for that transfer, is $2 \times (t_{RC} + t_{RP})$ cycles.

System clock during self-refresh mode. Note that the SDCLK is not disabled by the controller during self-refresh mode. However, software may disable the clocks by clearing the DSDCTL bit in the SDCTL register. Programs should ensure that all applicable clock timing specifications are met before the transfer to SDRAM address space (which causes the controller to exit the self-refresh mode). If a transfer occurs to SDRAM address space when the DSDCTL bit is cleared, an internal bus error is generated, and the access does not occur externally, leaving the SDRAM in self-refresh mode.

The following steps are required when using self-refresh mode.

1. Set the SDSRF bit to enter self-refresh mode
2. Poll the SDSRA bit in the SDRAM status register (SDSTAT) to determine if the SDRAM has already entered self-refresh mode.
3. Optionally: set the DSDCTL bit to freeze SDCLK
4. Optionally: clear the DSDCTL bit to re-enable SDCLK
5. SDRAM controller executes a self refresh exit sequence on receiving a SDRAM access request.

The minimum time between a subsequent self-refresh entry and exit command is the t_{RAS} cycle. If a self-refresh request is issued during any external port DMA, the controller grants the request with the t_{RAS} cycle and continues DMA operation afterwards.

Forcing SDRAM Commands

The controller has bits which can be specifically used to aid in debug and in specific system solutions.

By setting the SDPSS bit after reset, all mode registers are automatically updated. The SDPSS bit should be cleared when using forced commands.

Force Precharge All

Whenever an auto-refresh or a mode register set command is issued, the internal banks are required to be in idle state. Setting bit 21 (=1) forces a precharge all command to accomplish this. If the precharge all command is not issued, the auto-refresh and mode register set commands can be illegal depending on the current state.

Note that it is a good practice always to perform a force precharge all command before a forced refresh/mode register command.

Force Load Mode Register

Programs can use the Force LMR command by setting bit 22 (=1) in the SDCTL register. This command is preceded by a precharge all (if banks not idle) followed by a mode register write.

The Force LMR bit allows changes to the MODE register based settings during runtime. These settings include the CL (CAS latency) timing specification which needs to be changed to adapt to a new frequency operation.

Force Auto-Refresh

Bit 20 (=1) forces the auto refresh to be immediately executed (not waiting until the refresh counter has expired). This is useful for test purposes but also to synchronize the refresh time base with a system relevant time base.

DDR2 DRAM Controller (ADSP-2146x)

The DDR2 DRAM controller (the controller) on ADSP-2146x processors enable a transfer of data to and from synchronous DDR2 DRAM. It supports a glueless interface with four external banks, controlled by the memory chip select pins ($\overline{\text{DDR2\_CS3-0}}$), of standard DDR2 DRAMs of 256 Mbit to 2 Gbit with configurations x8 and x16.

Features

The features of the DDR2 DRAM controller are listed below.

- Supports up to 8G bit (254M x 32-bit) of DDR2 memory
- Supports DDR2-400 of 256M bit, 512M bit, 1G bit and 2G bit with x8 and x16

- Supports 4 and 8 bank DDR2 devices with page sizes of 512, 1K, 2K, and 4K words
- Variable memory address map (bank or page interleaving)
- Burst mode of 4 (BL = 4) with sequential burst type
- Supports multibank operation with open page policy
- Supports self-refresh mode and precharge power-down to reduce power consumption
- Supports read optimization (predictive solution)
- DDR2 PHY does provide the physical interface to JEDEC standard DDR2 Memories
- Supports programmable ODT (on-die termination)
- Supports 64-bit data SIMD and external instruction fetch in bank 0 for ISA/VISA instructions
- Independent transfers between DDR2 and AMI modules

Pin Descriptions

The pins used by the external memory interface are described in the *ADSP-2146x SHARC Processor Data Sheet*. Additional information on pin multiplexing can be found in "Pin Descriptions" on page 24-2.

Functional Description

On SDRAM systems all timing is referenced to the rising edge of the clock as per the JEDEC specification. However, since the clock speed has increased, this approach becomes limited based on setup and hold times. DDR2 is no longer system synchronous (as SDRAM), it is source synchronous which means the data source provides a reference signal (called the data strobe signal or DQS) which is sampled by the receiver and used to latch the data accordingly.

Two main modules shown in Figure 4-16 control the high speed throughputs/constraints. One block is the DDR2 controller which also interfaces to the arbiter (core vs. DMA) containing the state machine to generate the supported commands to the DDR2 memory. The other main module is the DDR2 PHY which owns DLL circuits and I/O logic (Data and Data strobes).

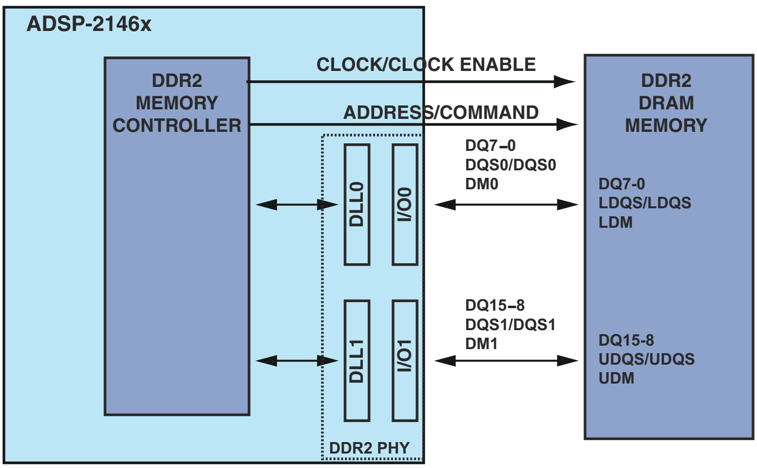

Figure 4-16. DDR2 Controller

DDR2 Controller

The controller uses burst length 4 (BL = 4) for read and write operations. This requires the controller to post only the first read or write address on the bus, all subsequent sequential address are posted by the DDR2 internal burst counter.

For read commands, there is a latency from the start of the read command to the availability of data from the DDR2, equal to the CAS latency. This latency is always present for any single read transfer. Subsequent reads do not have latency. Note that writes also have latency which = read latency – 1. For more information on commands used by the DDR2 controller, see "SDRAM Commands" below.

The configuration is programmed in the `DDR2CTL5-0` registers. The DDR2 controller can hold off the processor core or DMA controller with an internally connected acknowledge signal, as controlled by refresh, or page miss latency overhead. A programmable refresh counter is provided which generates background auto-refresh cycles at the required refresh rate based on the clock frequency used. The refresh counter period is specified using the `RDIV` field in the DDR2 refresh rate control register ("Refresh Rate Control Register (DDR2RRC)" on page A-52).

DDR2 memory accesses are burst oriented per the JEDEC specification. The burst accesses are NOT divisible and therefore every DDR2 access needs to satisfy the burst length of 4 words (4x16) even if not required for an application. This makes single read/write accesses inefficient and the controller needs to ignore (read) or mask (write) unwanted data.

DDR2 Arbiter

For read accesses, the DDR2 memory drives 16 bit data at both the edges of DDR2 clock which is sampled by the DDR2 controller data path, synchronized with internal clock and transferred to DDR2 arbiter as a single 64 bit data. The DDR2 arbiter in turns transfers the 64-bit data to corresponding queue for which the read request command was accepted. The queue in turn transfers the same on to DMA or core bus and unpacks the 64 bit data word in to 2 single words (32-bit) before transferring them on to external port DMA bus.

DDR2 PHY

The DDR2 PHY supplies the complete physical interface to JEDEC standard DDR2 SDRAM Memories. Figure 4-16 shows a representation of part a system of the DDR2 PHY with the controller and the external memory. There are two DLL circuits (DLL1–0) for each data byte lane (upper and lower byte)

- DLL0 controls DDR2_DATA7-0, DDR2_DQS0 and DDR2_DM0 pins.
- DLL1 controls DDR2_DATA15-8, DDR2_DQS1 and DDR2_DM1 pins.

As per JEDEC specifications, the data (DQ) are center-aligned with the DQS signal during a write to the memory and edge-aligned with the DQS signal during a read from the memory.

The DDR2 controller's command enables either the write or read path in the memory I/O.

During a DRAM write, the DDR2 controller performs the multiplexing of positive and negative edge data. This in turn is driven onto DQ as write data when the write path in the memory I/O buffers is activated. The corresponding write DDR2_DQS is also driven through the memory I/O, but after a phase shift of 90 degrees (controlled by the controller DLL).

During a memory read, the data from the DDR2 memory (SSTL-18 level) is converted to the core voltage logic level inside the DDR2 memory I/O pads. This is captured by the DDR2 DLL using precise delays on the data strobe (DDR2_DQS) line provided to the controller.

Read data is sent by the DDR2 DRAM on both the rising and falling edges of the DDR2_DQS signal. The read data is captured by the on-chip DLL using a delayed DQS that is phase shifted by approximately 90 degrees for the positive edge data and by approximately 90 degrees for the negative edge data. These delays are precisely generated by the internal on-chip DLL circuit. The captured data is sent out, corresponding to the data launched by the DRAM with the positive edge and negative edge of DDR2_DQS respectively.

For correct DDR2 operation it is important to connect control/data signals (DQx, DMx and DQSx) to their byte lane group only (high or low byte DLL). Only data signals (DQx) within a byte lane are allowed to be mixed, for example DQ7-0 can be connected to DQ0-7 of DDR2 device to match trace lengths in layout design.

It should be noted that the DDR2 PHY does not directly control the address and command lines. (DDR2_ADDR, DDR2_RAS, DDR2_CAS, DDR2_WE).

DDR2 Memory DLL

Although read data is captured at the controller using data strobes (DQS), this represents only a portion of the read timing (data capture timing). The complete DDR2 controller timing including the controller generating a read command, capturing the data, and transferring the data to its internal data path begins and ends in the controller clock domain. For this reason, it is necessary to specify a relationship between the DDR2 memory output data, data strobes and the input clock. Uncompensated timing variations (process, voltage and temperature) that occur in SDRAMs are not acceptable at the targeted clock frequencies. For this reason, a delay locked loop (DLL) is included in DDR2 memories to compensate for process, temperature and voltage variations. Note the variation updates occur during the auto-refresh command.

DDR2 memory systems require that the controller/memory DLLs are enabled (if the DLL is disabled the operation mode not JEDEC compliant). It is achieved by clearing the SH_DLL_DIS bit (controller, DDR2CTL0) and the DDR2DLLDIS bit (memory, DDR2CTL3).

Self Calibration Logic

Both data byte lanes are internally re timed such that they can be captured directly by the controller on the positive edge of DDR2_CLK, irrespective of the arbitrary phase relation that may exist between DDR2_CLK and the DDR2_DQS. During initial operation (external bank calibration), the on-chip DLL determines the phase difference between the DDR2_CLK and DDR2_DQS and re times the data captured accordingly.

In the last stage of the DDR2 init sequence, the controller starts an automated external memory bank calibration by sending dummy read commands which drive the memory's DQS strobes (via the memory DLL) back to the controller's DQS pins. The delay (phase relation between the internal DDR2 clocks and the DQS signals) is sensed and stored in a DLL register. This coarse delay represents PCB flight delay. The initial goal is to shift the DQS strobes into the center of what becomes the Read Data capture window.

To compensate for process, voltage and temperature related shifting of the DQS strobes, a continuous calibration runs during normal operation. The calibration logic monitors the delay taps of the DQS input paths. If a shift is detected, the delay count on the DQS strobe input paths can be fine adjusted to keep them centered in the Read Data capture window. The update is done during DDR2 memory auto-refresh command since the data path is idle avoid impacting normal data operations and controller efficiency.

Mode Registers

DDR2 functionality is programmed through the (extended) Mode registers (per JEDEC definition). These registers need to be programmed prior to using the interface. During the mode register command the address bus (DDR2_ADDR15-0) is used to program the various options while the DDR2 bank select pins (DDR2_BA1-0) are used to select one of the 4 mode registers.

All bit settings in the mode registers must be programmed. This can be done using the DDR2PSS bit in the DDR2 memory control register (DDR2CTL0) which automatically programs all mode registers bits appropriately. Note that options that are not supported are programmed to zero.

Note that programs can change the optional bits for forced mode registers (DDR2CTL0 register) individually afterwards. For more information refer to automated initialization sequence.

Load Mode Register

The MR command initializes DDR2 operation parameters. The controller supports CAS latency. For more information, see “DDR2 Control Register 2 (DDR2CTL2)” on page A-46.

Load Extended Mode Register

The EMR command initializes enhanced DDR2 operation parameters. The following options are supported.

- DDR2 DLL disable
- Output drive strength reduced
- Additive latency
- On die termination

- Differential DQS signal disable
- Output buffer disable

For more information, see "DDR2 Control Register 3 (DDR2CTL3)" on page A-48.

Load Extended Mode Register 2

The EMR2 command initializes enhanced 2 DDR2 operation parameters. The controller does not support any of these options. For more information, see "DDR2 Control Register 4 (DDR2CTL4)" on page A-50.

Load Extended Mode Register 3

The EMR3 command initializes enhanced 3 DDR2 operation parameters. The controller does not support any of these options. For more information, see "DDR2 Control Register 5 (DDR2CTL5)" on page A-51.

DDR2 Commands

This section provides a description of each of the commands that the DDR2 controller uses to manage the DDR2 interface. These commands are handled automatically by the DDR2 controller. A summary of the various commands, including the truth tables used by the on-chip controller for the DDR2 interface can be found in the JEDEC specification (JESD79–2x).

Bank Activation

This command is required if the next data access is on a different page in the same internal bank or in a different internal bank that is in an idle state. The controller executes the pre-charge command, followed by a bank activate command, to activate the page in the desired DDR2 internal bank. The controller is able to open up to eight pages at the same time in different internal banks. For 8 banked devices, the controller follows the t_{FAW} specification.

Precharge

This command is executed by the controller if the address to be accessed falls in a different page in the same external bank and the same internal bank. A precharge is not done if the address to be accessed falls in an open page in another internal or external bank.

For page miss reads or writes, only the external and internal banks to be accessed by the read or write is pre-charged. For auto-refresh and self-refresh, all external DDR2 banks are pre-charged at one time.

Precharge All

This command is given to precharge all internal banks. Just before an auto refresh or self refresh, or during the power up sequence, the controller always issues the precharge command to all internal DDR2 banks. For eight bank devices, the t_{FAW} period must be satisfied while performing the precharge all command.

Burst Read

The burst read command is initiated by having $\overline{\text{DDR2\_CS}}$ and $\overline{\text{DDR2\_CAS}}$ low while holding DDR2_RAS and DDR2_WE high at the rising edge of the clock. The address inputs determine the starting column address for the burst. The delay from the start of the command to when the data from the first cell appears on the outputs is equal to the value of the read latency (RL).

The data strobe output (DDR2_DQS) is driven low one clock cycle before valid data (DDR2_DATA) is driven onto the data bus (Figure 4-17). The first bit of the burst is synchronized with the rising edge of the data strobe (DDR2_DQS).

Each subsequent data-out appears on the DDR2_DATA The first bit of the burst is synchronized with the rising edge of the data strobe (DDR2_DQS). Each subsequent data-out appears on the DDR2_DATA pin in phase with the DDR2_DQS signal in a source synchronous manner.

The RL is equal to an additive latency (AL) plus CAS latency (CL). The CL is defined by the mode register (MR), similar to the existing SDRAM. The AL is defined by the EMR1 register.

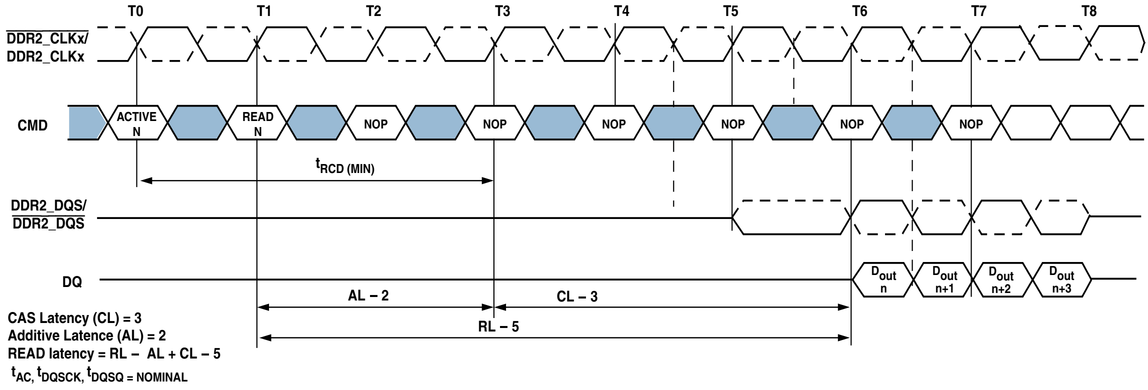

Figure 4-17. Burst Read

Burst Write

The burst write command, shown in Figure 4-18, is initiated by having DDR2_CS, DDR2_CAS and DDR2_WE pins low while holding DDR2_RAS high at the rising edge of the clock. The address inputs determine the starting column address. Write latency (WL) is defined by a read latency (RL) minus one and is equal to (AL + CL – 1) and is the number of clocks of delay that are required from the time the write command is registered to the clock edge associated to the first DDR2_DQS strobe.

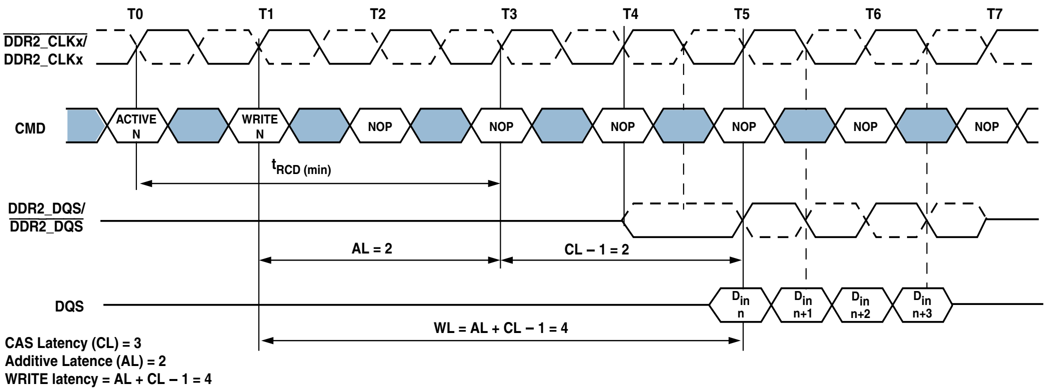

Figure 4-18. Burst Write

A data strobe signal (DDR2_DQS) should be driven low (preamble) nominally a 1/2 clock prior to the WL. The first data bit of the burst cycle must be applied to the DDR2_DATA pins at the first rising edge of DDR2_DQS following the preamble.

The subsequent burst bit data are issued on successive edges of DDR2_DQS until the burst length is completed. When the burst has finished, any additional data supplied to the DDR2_DATA pins is ignored. The DDR2_DATA signal is ignored after the burst write operation is complete. The time from the completion of the burst write to bank precharge is the write recovery time (WR).

Auto-Refresh

The DDR2 internally increments the refresh address counter and causes a CAS before RAS (CBR) refresh to occur internally for that address when the auto-refresh command is given. The controller generates an auto-refresh command after the refresh counter times out. The RDIV value in the DDR2RRC register must be set so that all addresses are refreshed within the t_{REF} period specified in the DDR2 timing specifications.

Before executing the auto-refresh command, the DDR controller executes a pre-charge all command to all external banks. The next activate command is not given until the t_{RFC} specification is met. Auto-refresh commands are also issued by the controller as part of the power-up sequence and after exiting self-refresh mode.

Self-Refresh Entry

Self-refresh mode causes refresh operations to be performed internally by the DDR2 controller, without any external control. This means that the controller does not generate any auto refresh cycles while it is in self-refresh mode. The self-refresh entry command is performed by writing a 1 to the DDR2SRF bit of the memory control register (DDR2CTL0). This deasserts the DDR2_CKE pin to put the device into self-refresh mode. In this mode, the DDR2 memory DLL is put into reset in order to reduce power consumption.

If any of the two DDR2 clocks is not required in a system during self-refresh, they can be stopped by setting the DIS_DDR2CTL and DIS_DDR2CLK1 bit in the DDR2CTL0 control register. This reduces the power consumption in a system and is shown in the following code example.

```
ustat1 = dm(DDR2CTL0);
bit set ustat1 DDR2SRF;      /* enter self-refresh */
dm(DDR2CTL0) = ustat1;
nop;

ustat2 = dm(DDR2STAT0);
bit tst ustat2 DDR2SRA;     /* test self-refresh entry */
if not TF jump (pc,-2);

ustat1 = dm(DDR2CTL0);
```

```
bit set ustat1 DIS_DDR2CTL|DIS_DDR2CLK1;  /* freeze DDR2 clock */
dm(DDR2CTL0) = ustat1;
nop;
```

This requires careful software control because the DIS_DDR2CTL bit is set during runtime. Systems may become unstable if this bit is set too early because the system can lose control of the DDR2 memory device.

Self-Refresh Exit

The DDR2 remains in self-refresh mode for at least t_{RAS} period and until an access to DDR2 space occurs or SREF_EXIT bit in DDR2CTL0 is set. When exiting from self-refresh mode programs need to consider if this occurs during a read or write. If exiting during a read, additional latency occurs because the DDR2 memory DLL needs to be locked again.

When an access to DDR2 space occurs or when the SREF_EXIT bit is set in the DDR2CTL0 register, the controller:

1. Exits DDR2 from self-refresh mode by asserting DDR2CKE pin high
2. Waits to meet the t_{XSNR} specification ($t_{XSNR} = t_{RAS} + t_{RP}$)
3. Issues an auto-refresh command

   After the auto-refresh command, the controller waits for the t_{RFC} specification to be met before executing the activate command for the transfer that caused the DDR2 to exit self-refresh mode.

4. For reads, the t_{XSRD} time must be satisfied. When exiting self refresh, ODT must remain low until t_{XSRD} is satisfied. For example:

```
ustat1 = dm(DDR2CTL0);
bit clr ustat1 DIS_DDR2CTL|DIS_DDR2CLK1;
dm(DDR2CTL0) = ustat1;        /* release clock */
nop;
```

```
ustat2 = dm(DDR2STAT0);
bit tst ustat2 DDR2SRA;
if not TF jump (pc,-2);         /* test self-refresh */
dm(DDR2_ADDR) = r0;             /* exit self-refresh */
```

Precharge Power-Down Entry

The DDR2 controller supports DDR2 precharge power down mode. In this mode, the DDR2 memory DLL is disabled (like Self-refresh mode) to maximize power consumption.

When the DIS_DDR2CKE bit is set to 1 and the DDR2 controller enters an idle state, it issues a pre-charge command (if necessary) and then, after meeting the required timing specifications, pulls down the DDR2CKE signal. If an internal access is pending, the controller delays entering the power-down mode until it completes the pending DDR2 access and any subsequent pending access requests.

- DIS_DDR2CKE = 0 No effect.
- DIS_DDR2CKE = 1 Enter precharge power down.

Once the DDR device enters into power-down mode, the DDR controller asserts the DDR2PD bit in the DDR control status register (DDR2STAT0).

Unlike self-refresh mode, precharge power-down entry mode does not refresh the DDR2 device. Therefore, careful software control is required so as not to violate refresh conditions which leads to data corruption. The typical refresh interval of t_{REFI} can be extended up to $8 \times t_{REFI}$. Consult the DDR2 data sheet for complete information.

This mode is useful if the DDR2 operation is idling only for a *short* period of time. This time is limited by the JEDEC spec and is typically $9 \times t_{REFI}$. If for example $t_{REFI} = 7.8$ µs the maximum power-down time is 9×7.8 µs = 70 µs. According to the JEDEC standard eight burst refresh cycles are required before entering precharge power down mode.

When DDR2 memory pauses for a short period of time, systems should evaluate on a case by case basis whether or not self-refresh or precharge power-down should be used. This consideration will take into account that precharge power-down is limited to a timing window of approximately 70 μs (9 × t_{REFI}), and that self-refresh release requires 200 DDR2 cycles for the DLL to lock again.

Precharge Power-down Exit

The DDR2 device exits power-down mode only when the DIS_DDRCKE bit in the control register is cleared. The controller takes care of the power-down exit timing specifications t_{XP}, t_{XARD}, t_{XARDS} and t_{CKE} min.

No Operation/Command Inhibit

The no operation (NOP) command to the DDR2 has no effect on operations currently in progress. When the controller is actively accessing the DDR2 but needs to insert additional commands with no effect, the NOP command is given.

The command inhibit command is the same as a NOP command, except that the DDR2 is not chip-selected. When the controller is not accessing any DDR2 external banks, the command inhibit command is given.

Refresh Rate Control

The DDR2 refresh rate control register (DDR2RRC) provides a flexible mechanism for specifying the auto-refresh timing. The DDR2 controller provides a programmable refresh counter which has a period based on the value programmed into the RDIV field of this register, which coordinates the supplied clock rate with the DDR2 device's required refresh rate.

The delay (in number of DDR2_CLK cycles) desired between consecutive refresh counter time-outs must be written to the RDIV field. A refresh counter time-out triggers an auto-refresh command to the external DDR2 bank. Write the RDIV value to the DDR2RRC register before the DDR2 power-up sequence is triggered. Change this value only when the DDR2 controller is idle.

To calculate the value that should be written to the DDR2RRC register, use the following equation:
RDIV = (DDR2_CLK × t_{REFI}) – (t_{RAS} + t_{RP}) where:

- DDR2 Clock = DDR2 system clock frequency
- t_{REFI} = DDR2 maximum average auto refresh period (in us). (Note t_{REFI} = t_{REF}/Number of row addresses)
- t_{RAS} = Active to precharge time (DDR2_RAS bit in the DDR2CTL1 register) in number of clock cycles
- t_{RP} = RAS to precharge time (in the DDR2CTL1 register) in number of clock cycles

This equation calculates the number of clock cycles between the required distributed refreshes, and subtracts the required delay between bank activate commands to the same bank (t_{RC} = t_{RAS} + t_{RP}). The t_{RC} value is subtracted, so that in the case where a refresh time-out occurs while a DDR2 cycle is active, the refresh rate specification is guaranteed to be met. The result from the equation should always be rounded down to an integer.

The t_{RFC} field in (DDR2RRC) provides the row refresh cycle time. (Required time after receiving the refresh command and until row refresh done).

Below is an example of the calculation of RDIV for a typical DDR2 memory in a system with a 200 MHz clock.

DDR2_CLKx = 200 MHz
t_{REFI} = 7.8 μs
t_{RAS} = 9 cycles
t_{RP} = 3 cycles

The equation for RDIV yields:
RDIV = $(200 \times 10^{6} \times 7.8 \times 10^{-6}) - (9 + 3) = 1548$ clock cycles.

This means RDIV is 0x614 and the DDR2RRC register bits 13–0 should be written with 0x60C.

Note that the RDIV bit must be programmed to a non-zero value if the DDR2 controller is enabled. When RDIV = 0, operation of the controller is not supported and can produce undesirable behavior. Values for RDIV can range from 0x001 to 0x3FFF.

The refresh interval (t_{REFI}) may change with the application used. For consumer parts t_{REFI} =7.8 μs while for industrial and automotive parts t_{REFI} =3.9 μs.

Data Mask

The DDR2 controller provides two DDR2_DM1-0 pins. Both pins (for each byte) should be connected to the DDR2 DM pins.

The meaning of this pin is significant, based on the fact that the minimum burst length is 4 and a burst is not divisible. The DDR2_DM1-0 pins are used to mask the data on both edges of the DQS signal during writes in cases less than 4 sequential writes, for example a single write need to mask the data for the next sequential 3 writes.

> The DDR2_DM1-0 pins are useful for performance monitoring during write commands. When asserted they indicate that the controller is masking unwanted data writes that cause performance penalties. For reads, the controller does not latch the data from the burst.

Resetting the Controller

Like any other peripheral, the DDR2 controller can be reset by hard- or a soft reset. Both reset modes pull the DDR2_CKE pin asynchronously low. Since DDR2_CKE drops asynchronously and the PLL goes into bypass mode (hardware reset) immediately after reset, timing parameter cannot be met, causing data loss. The DDR2 device must be re-initialized and the DDR2 DLL must be re locked to use the DDR2 again.

> Running reset ($\overline{\text{RESETOUT}}$ pin as an input) does not reset the DDR2 controller.

Automated Initialization Sequence

DDR2 SDRAM must be powered up and initialized in a predefined manner. After the DDR2PSS bit is set in the DDR2CTL0 register, the DDR2 controller starts the power-up initialization sequence which occurs in the following order. Note that this procedure is performed by the DDR2 controller and user intervention is not required.

Table 4-14. DDR2 Initialization Sequence

Step	Action	Wait (Timing Specification)	Register Read
1	DDR2CKE high, drive a NOP command.		
2	Wait a minimum of 400 ns (with NOP or DESELECT commands).		
3	Issue a precharge all command.	t_{RP} period	
4	Issue a load EMR(2) command to initialize operating parameters.	t_{MRD} period	DDR2CTL4
5	Issue a load EMR(3) command to initialize operating parameters.	t_{MRD} period	DDR2CTL5
6	Issue a load EMR command. Enable memory DLL (clear bit 0), all other operating parameters cleared.	t_{MRD} period	Hard coded bit mask
7	Issue a load EMR command. Reset memory DLL (set bit 8), all other operating parameters cleared. Trigger a 200 cycle counter for memory DLL lock.	t_{MRD} period	Hard coded bit mask
8	Issue a precharge all command.	t_{RP} period	
9	Issue four auto refresh commands.	$4 \times t_{RFC}$ period	
10	Issue a load EMR command without resetting the memory DLL (bit 8), to initialize operating parameters.	t_{MRD} period	DDR2CTL2
11	Wait for the 200 cycle counter (step 7) to be expired		
12	Issue a load EMR command OCD default operation by setting bits A9–7. All other operating parameters are cleared.	t_{MRD} period	Hard coded bit mask
13	Issue a load EMR command OCD exit operation by clearing bits A9–7 and initialize operating parameters.	t_{MRD} period	DDR2CTL3
15	Start the DLL external bank calibration		EPCTL (ext bank select)
16	DDR2 ready for user access		

Initialization Time

After setting the power-up start bit, the controller starts internal and external calibration routines which are described below. The actual cycles may vary due to different timing specifications.

- Best case (one external DDR2 bank assigned). The entire power up requires 680 DDR2 initialization + 660 external bank calibration = around 1340 DDR2 cycles.
- Worst case (all external DDR2 banks assigned). Entire power up requires 680 DDR2 initialization + (4 x 660 external bank calibration) = around 3320 DDR2 cycles.

Internal DDR2 Bank Access

The following sections describe the different scenarios for DDR2 bank access.

Single Bank Access

The DDR2 controller keeps only one page open at a time if all subsequent accesses are to the same row or another row in the same bank.

Multibank Access

The processors are capable of supporting multibank operation, thus taking advantage of the DDR2 architecture.

Operation using single versus multibank accesses depends only on the address to be posted to the device, it is NOT an operation mode.

Any first access to DDR2 bank (A) forces an activate command before a read or write command. However, if any new access falls into the address space of the other banks (B, C, D, E, F, or H) the controller leaves bank (A) open and activates any of the other banks (B, C, D, E, F, or H). Bank (A) to bank (B) active time is controlled by t_{RRD}. This scenario is repeated until all eight banks (A–H) are opened and results in an effective page size of up to eight pages. This is because the absence of latency allows switching between these open pages (as compared to one page in only one bank at a time).

Any access to any closed page in any opened bank (A–H) forces a precharge command only to that bank. If, for example, two external port DMA channels are pointing to the same internal DDR2 bank, this always forces precharge and activation cycles to switch between the different pages. However, if the two external port DMA channels are pointing to different internal DDR2 banks, there is no additional overhead. See Figure 4-19.

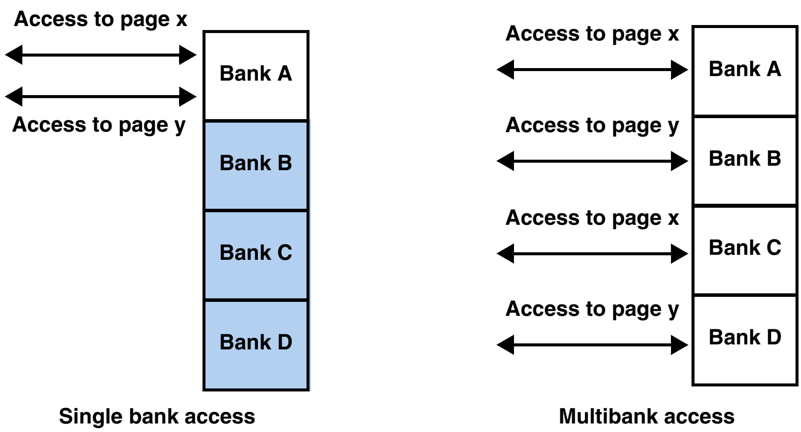

Figure 4-19. Single Versus Multibank Access

Force Activation Window

Traditionally, SDRAM has operated with a maximum of 4 internal banks. However, with DDR2 some higher-density devices will support 8 individual banks. For this reason, JEDEC has limited the number of banks that may be activated within a set period.

DDR2 devices support a new timing parameter called four active banks window (t_{FAW}). This is the minimum amount of time that must pass before more than four ACTIVE (ACT) commands may occur. It is acceptable to have more than 4 banks open simultaneously, but the additional ACT command(s) must be spaced out past the t_{FAW}(min) window. As shown in Figure 4-20, t_{RCD} for the fourth opened bank is complete at T8. To satisfy t_{FAW}(min), the fifth ACT command cannot occur until T11.

Furthermore the controller supports four external memory selects containing each DDR2. All external banks (`DDR2_CSx`) provide multibank support, so the maximum number of open pages is 8 x 4 = 32 pages.

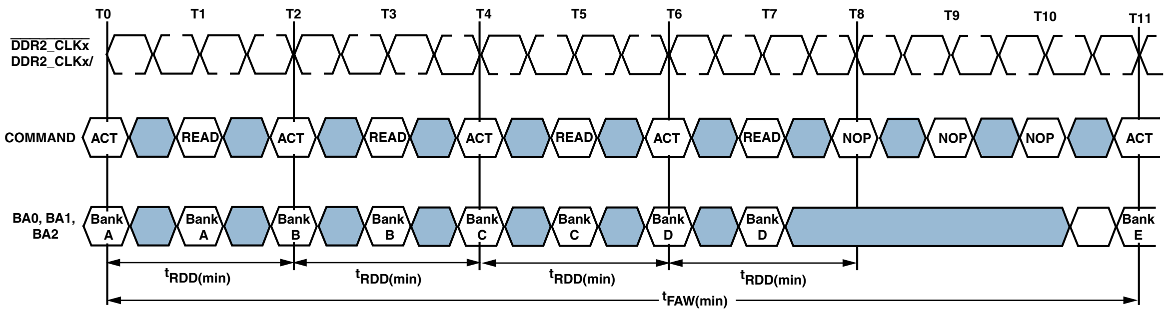

Figure 4-20. Bank Activation for a 8 Banked Device

Multibank access reduces precharge and activation cycles by mapping opcode/data among different internal DDR2 banks driven by the (`DDR2_BA2-0`) pins and external memory selects (`DDR2_CS3-0`).

Multi-Bank Operation with Data Packing

A logical address corresponds to 2 physical addresses. Consequently a physical address for example of 1024 x 16 page size translates into a logical address of 512 x 16 words to satisfy the packing. According to this all row addresses are shifted by 2.

A populated DDR2 of 8M x 16 x 8 with 1K words page size connected to external bank 0 has a logical mapping as follows.

Page Interleaving (DDR2ADDRMODE bit = 0):

```
0x200000 logical start address int bankA
0x2001FF logical end address int bankA
0x200200 logical start address int bankB
0x2003FF logical end address int bankB
0x200400 logical start address int bankC
0x2005FF logical end address int bankC
0x200600 logical start address int bankD
0x2007FF logical end address int bankD

0x200800 logical start address int bankE
0x2009FF logical end address int bankE
0x200A00 logical start address int bankF
0x200BFF logical end address int bankF
0x200C00 logical start address int bankG
0x200DFF logical end address int bankG
0x200E00 logical start address int bankH
0x201000 logical end address int bankH
```

Bank Interleaving (DDR2ADDRMODE bit = 1):

```
0x200000 logical start address int bankA
0x2001FF logical end address int bankA
0x600000 logical start address int bankB
0x6001FF logical end address int bankB
0xA00000 logical start address int bankC
0xA001FF logical end address int bankC
0xE00000 logical start address int bankD
0xE001FF logical end address int bankD

0x1200000 logical start address int bankE
0x12001FF logical end address int bankE
0x1600000 logical start address int bankF
0x16001FF logical end address int bankF
0x1A00000 logical start address int bankG
0x1A001FF logical end address int bankG
0x1E00000 logical start address int bankH
0x1E001FF logical end address int bankH
```

Fixed Timing Parameters

The timing specifications below are fixed by the controller.

- t_{MRD} (mode register delay). Required delay time to complete the mode register write. This parameter is fixed to 2 cycles.
- t_{RC} (row access cycle). Required delay time to open and close a single row. This parameter is fixed to $t_{RC} = t_{RAS} + t_{RP}$ cycles.
- t_{CCD} (column to column delay). Required delay between two column accesses (read/write). This parameter is fixed to 2 cycles.
- t_{XSNR} (exit self-refresh with non-read). Required delay to exit the self-refresh mode with a non read command. This parameter is fixed to $t_{XSNR} = t_{RFC} + 4$ cycles.
- t_{XSRD} (exit self-refresh with read). Required delay to exit the self-refresh mode with a read command. This parameter is fixed to $t_{XSRD} = 200$ cycles.

The DDR2 controller controls the following ODT related timing parameters, no user programming is required.

- t_{ANPD} (ODT to power-down entry latency)
- t_{AXPD} (ODT to power down exit latency)
- t_{AOND} (ODT turn on delay)
- t_{AOFD} (ODT turn off delay)
- t_{AON} (ODT turn on time)
- t_{AOF} (ODT turn off time)

16-Bit Data Storage and Packing

The processors use logical addressing when an external memory smaller than 32 bits is used. Logical addresses require multiple external addresses seen by the memory correspond to a single internal address, depending on the width of the memory being accessed.

The external physical address map is shown in Table 4-15.

Table 4-15. DDR2 Address Memory Map

Bus Width	External Memory Bank	Internal Logical Address (supported memory map)	External Physical Address (on ADDR23–0)
16-bit	0	0x0020_0000 – 0x007F_FFFF	0x40_0000 – 0xFF_FFFF
16-bit	1, 2, 3	0x0400_0000 – 0x047F_FFFF 0x0800_0000 – 0x087F_FFFF 0x0C00_0000 – 0x0C7F_FFFF	0x00_0000 – 0xFF_FFFF

External Instruction Fetch

For more information, see “External Instruction Fetch” on page 4-44.

Address For DDR2 Types

Table 4-16 provides addressing for various sizes of DDR2 DRAM memory.

Table 4-16. Translation of Logical to Physical Addressing for DDR2

DDR2 Device	Physical Address Range Mapped to Memory Device	Mapping Between External Port Address Range and Memory Device
256 Mb (x16)	0x60 0000 – 0x15F FFFF	0x20 0000 – 0xFF FFFF 0x00 0000 – 0x1F FFFF
512 Mb (x16)	0x60 0000 – 0x25F FFFF	0x020 0000 – 0x1FF FFFF 0x000 0000 – 0x01F FFFF
1 Gb (x16)	0x60 0000 – 0x45F FFFF	0x020 0000 – 0x3FF FFFF 0x000 0000 – 0x01F FFFF

Operating Modes

The following sections provide on the operating modes of the DDR2 interface.

Address Mapping

To access DDR2, the DDR2 controller multiplexes the internal 32-bit non-multiplexed address into three portions:

- Row address bits
- Column address bits
- Bank address bits

The non multiplexed address that is seen from the core/DMA is referred to as IA31–0 in the following sections.

Address Translation Options

To provide flexible addressing, `DDR2ADDRMODE` (bit 14 in the `DDR2CTL0` register) is used to select the address mapping scheme—page interleaving (default) or bank interleaving.

Page Interleaving Map

Programming the `DDR2ADDRMODE` bit to 0 selects the page interleaving scheme. In this scheme consecutive pages fall in consecutive banks. The bank address bits follow the most significant column address bits. This is shown in Figure 4-21.

31			0
Unused	**Row Address**	**Bank Address**	**Column Address**

Figure 4-21. Core Address Mapping to Row, Column Addresses (Page)

One advantage of the page interleaving is that the effective page size is up to four pages (assuming four banks activated) and all the addresses are sequential. If using delay line DMA mode, the addresses for a long delay line are all sequential, simplifying the addressing. Moreover, DDR2 sequential addressing provides maximum performance.

Bank Interleaving Map

Programming the `DDR2ADDRMODE` bit to 1 selects the bank interleaving scheme. In this scheme consecutive pages sit in the same bank. The bank address bits follow most significant row address bits. This is shown in Figure 4-22.

31			0
Unused	Bank Address	Row Address	Column Address

Figure 4-22. Core Address Mapping to Row, Column Addresses (Bank)

One advantage of bank interleaving is that the effective page size is also up to four pages (assuming four banks activated) but the addresses of the four pages are not sequential. If the program uses two external port DMAs pointing to the DDR2 space, this scheme has advantages since every bank has its one DMA buffer addressing.

Address Width Settings

Number Internal Banks (DDR2BC). The controller assumes the DDR2 is comprised of eight bank devices. However, DDR2 can use four bank devices by not connecting the DDR2_BA2 pin and programming the DDR2BC bits in the DDR2CTL0 register. The external bank addresses are decoded as shown in Table 4-17.

Table 4-17. External Memory Address Bank Decoding

IA[27]	IA[26]	External Bank
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3

Row Address Width (DDR2RAW). These bits in the DDRCTL0 register determine the row width of the DDR. The DDR2RAW bits can be programmed for row widths of 8 to 15.

Column Address Width (DDR2CAW). The DDR2 memory control register also includes external bank specific programmable parameters. The external bank can be configured for a different DDR2 size. The DDR controller determines the internal DDR2 page size from the X16DE and DDR2CAW parameters. Page sizes of 256, 512, 1K, 2K and 4K words are supported.

The mapping of the addresses depends on the row address width (DDR2RAW), column address width (DDR2CAW), and the address mode bit (DDR2ADDRMODE) setting.

16-Bit Address Mapping

Even if the external data width is 16 bits, the processor supports only 32-bit data accesses. The DDR2 controller performs two 16-bit accesses to get and place 32-bit data. The controller takes the IA address and appends one extra bit to the LSB to generate the address externally.

For example, if the processor core requests address 0x20 0000 for a 32-bit access, the controller performs two 16-bit accesses at 0x40 0000 and 0x40 0001, using MS0 to get one 32-bit data word.

The X16DE bit must always be set.

Address Map Tables

The row address and column address mappings for 16-bit addresses are shown in Table 4-18 through Table 4-21. The row, bank and column addresses are multiplexed to the A14–A0 and BA2–BA0 pins of the processor.

Table 4-18 through Table 4-21 also show the mapping of the internal address [IA] to the external address. The mapping of the address depends on row address width, column address width, the number of internal banks, and the external I/O width.

Table 4-18 shows DDR2ADDRMODE = 0, DDR2RAW = 100 (12), DDR2CAW = 10 (10), DDR2BC = 10.

Table 4-18. 16-bit Address Mapping (8 Banks, Page Interleaving)

SHARC Pin	Column Address	Row Address	Bank Address	DDR2 Pin
DDR2_BA2			IA[11]	BA[2]
DDR2_BA1			IA[10]	BA[1]
DDR2_BA0			IA[9]	BA[0]
DDR2_ADDR[12]				A[12]
DDR2_ADDR[11]		IA[23]		A[11]

Table 4-18. 16-bit Address Mapping (8 Banks, Page Interleaving) (Cont'd)

SHARC Pin	Column Address	Row Address	Bank Address	DDR2 Pin
DDR2_ADDR[10]		IA[22]		A[10]
DDR2_ADDR[9]	IA[8]	IA[21]		A[9]
DDR2_ADDR[8]	IA[7]	IA[20]		A[8]
DDR2_ADDR[7]	IA[6]	IA[19]		A[7]
DDR2_ADDR[6]	IA[5]	IA[18]		A[6]
DDR2_ADDR[5]	IA[4]	IA[17]		A[5]
DDR2_ADDR[4]	IA[3]	IA[16]		A[4]
DDR2_ADDR[3]	IA[2]	IA[15]		A[3]
DDR2_ADDR[2]	IA[1]	IA[14]		A[2]
DDR2_ADDR[1]	IA[0]	IA[13]		A[1]
DDR2_ADDR[0]	1/0	IA[12]		A[0]

Table 4-17 shows DDR2ADDRMODE = 0, DDR2RAW = 100 (12), DDR2CAW = 11 (11), DDR2BC = 01(four banks).

Table 4-19. 16-bit Address Mapping (4 Banks, Page Interleaving)

SHARC Pin	Column Address	Row Address	Bank Address	DDR2 Pin
DDR2_BA1			IA[11]	BA[1]
DDR2_BA0			IA[10]	BA[0]
DDR2_ADDR[13]				
DDR2_ADDR[12]				A[12]
DDR2_ADDR[11]	IA[9]	IA[23]		A[11]
DDR2_ADDR[10]		IA[22]		A[10]
DDR2_ADDR[9]	IA[8]	IA[21]		A[9]
DDR2_ADDR[8]	IA[7]	IA[20]		A[8]
DDR2_ADDR[7]	IA[6]	IA[19]		A[7]

Table 4-19. 16-bit Address Mapping (4 Banks, Page Interleaving) (Cont'd)

SHARC Pin	Column Address	Row Address	Bank Address	DDR2 Pin
DDR2_ADDR[6]	IA[5]	IA[18]		A[6]
DDR2_ADDR[5]	IA[4]	IA[17]		A[5]
DDR2_ADDR[4]	IA[3]	IA[16]		A[4]
DDR2_ADDR[3]	IA[2]	IA[15]		A[3]
DDR2_ADDR[2]	IA[1]	IA[14]		A[2]
DDR2_ADDR[1]	IA[0]	IA[13]		A[1]
DDR2_ADDR[0]	1/0	IA[12]		A[0]

Table 4-20 shows `DDR2ADDRMODE` = 1, `DDR2RAW` = 100 (12), `DDR2CAW` = 10 (10), `DDR2BC` = 10 (eight banks).

Table 4-20. 16-bit Address Mapping (8 Banks, Bank Interleaving)

SHARC Pin	Column Address	Row Address	Bank Address	DDR2 Pin
DDR2_BA2			IA[23]	BA[2]
DDR2_BA1			IA[22]	BA[1]
DDR2_BA0			IA[21]	BA[0]
DDR2_ADDR[12]				A[12]
DDR2_ADDR[11]		IA[20]		A[11]
DDR2_ADDR[10]		IA[19]		A[10]
DDR2_ADDR[9]	IA[8]	IA[18]		A[9]
DDR2_ADDR[8]	IA[7]	IA[17]		A[8]
DDR2_ADDR[7]	IA[6]	IA[16]		A[7]
DDR2_ADDR[6]	IA[5]	IA[15]		A[6]
DDR2_ADDR[5]	IA[4]	IA[14]		A[5]
DDR2_ADDR[4]	IA[3]	IA[13]		A[4]
DDR2_ADDR[3]	IA[2]	IA[12]		A[3]

Table 4-20. 16-bit Address Mapping (8 Banks, Bank Interleaving) (Cont'd)

SHARC Pin	Column Address	Row Address	Bank Address	DDR2 Pin
DDR2_ADDR[2]	IA[1]	IA[11]		A[2]
DDR2_ADDR[1]	IA[0]	IA[10]		A[1]
DDR2_ADDR[0]	1/0	IA[9]		A[0]

Table 4-21 shows DDR2ADDRMODE = 1, DDR2RAW = 100 (12), DDR2CAW = 11 (11), DDR2BC = 01(four banks).

Table 4-21. 16-bit Address Mapping (4 Banks, Bank Interleaving)

SHARC Pin	Column Address	Row Address	Bank Address	DDR2 Pin
DDR2_BA1			IA[23]	BA[1]
DDR2_BA0			IA[22]	BA[0]
DDR2_ADDR[13]				
DDR2_ADDR[12]				A[12]
DDR2_ADDR[11]	IA[9]	IA[21]		A[11]
DDR2_ADDR[10]		IA[20]		A[10]
DDR2_ADDR[9]	IA[8]	IA[19]		A[9]
DDR2_ADDR[8]	IA[7]	IA[18]		A[8]
DDR2_ADDR[7]	IA[6]	IA[17]		A[7]
DDR2_ADDR[6]	IA[5]	IA[16]		A[6]
DDR2_ADDR[5]	IA[4]	IA[15]		A[5]
DDR2_ADDR[4]	IA[3]	IA[14]		A[4]
DDR2_ADDR[3]	IA[2]	IA[13]		A[3]
DDR2_ADDR[2]	IA[1]	IA[12]		A[2]
DDR2_ADDR[1]	IA[0]	IA[11]		A[1]
DDR2_ADDR[0]	1/0	IA[10]		A[0]

Parallel Connection of DDR2s

To specify a DDR2 system, multiple possibilities are given based on the different memory sizes. For a 16-bit I/O capability, the following memory sizes can configured.

- 1 x 16-bit/page 512 words
- 2 x 8-bit/page 1k words
- 4 x 4-bit/page 2k words

The DDR2's page size is used to determine the system you select. All three systems have the same external bank size, but different page sizes. Note that larger page sizes, allow higher performance but larger page sizes require more complex hardware layouts.

Even if connecting DDR2s in parallel, the controller always considers the cluster as one external DDR2 bank because all address and control lines feed the parallel parts.

Buffering Controller for Multiple DDR2s

If using multiples DDR2s or modules, the capacitive load will exceed the controller's output drive strength. In order to bypass this problem an external register (SSTL18 class) can be used for decoupling by setting bit 24 in `DDR2CTL0` register. This adds a cycle of data buffering to read and write accesses.

Read Optimization

The best throughput numbers for reads are achievable only when the `DDR2OPT` bit in the `DDR2CTL0` register is set. To achieve better performance for reads, predictive addresses need to be given to the DDR memory. The predictive address given to the memory depends on the `DDR2MODIFY` bit setting. If the DDR2MODIFY value is 2 then the address + 2 is given predictively on the DDR address pins. Programs have the option whether to use read optimization or not.

It is advisable to use read optimization for core and DMA transfers, with a constant modifier to achieve better performance. With multiple channels running with ping-pong accesses, use arbitration freezing to get better throughput.

For SIMD accesses, if optimization is enabled and the modifier is set to 2 (even if the modifier is changed, it remains at 2). The throughput is at maximum if optimization is enabled for sequential accesses. But in the case of non sequential accesses, throughput is affected by enabling optimization.

Read Optimization Modifier

The predictive address given to the memory depends on the DDR2MODIFY bit values. For example, if the DAG modifier = 2, the DDR2MODIFY value should also be 2, in which case the address + 2 is the predictive value provided to the DDR2 address pins. Programs may choose to determine whether read optimization is used or not. If read optimization is disabled, then each read takes 11 DDR2CLK cycles for a CAS latency of 3, even for sequential reads.

With read optimization enabled, 32 sequential reads, with offsets ranging from 0 to 15, take only 42 DDR2CLK cycles. Read optimization should not be enabled while reading at the external bank boundaries. For example, if DDR2MODIFY = 1, then 32 locations in the boundary of the external banks should not be used. These locations can be used without optimization enabled. If DDR2MODIFY = 2, then 64 locations cannot be used at the boundaries of the external bank (if it is fully populated).

It should be noted that read optimization always improves the read performance (if the access modifiers are deterministic and accesses are not interrupted) by a factor of approximately 4–5.

Optimization extracts the data from the incoming burst as described below.

- For modifier = 1 the controller does not need to extract the correct data from the burst of 4 and has the best data throughput.
- For modifier = 2 the controller extracts every second data from the burst of 4 and has the second best data throughput.

By default, the read optimization is enabled (SDROPT = 1) with a modifier of 1 (DDR2MODIFY = 1). Read optimization assumes that the DDR2 pointer has a constant modifier. For non-sequential accesses, turning off optimization provides better results.

Core Accesses

Any break of sequential reads of full page accesses can cause a throughput loss due to a maximum of eight extra reads. Listing 4-4 shows how to achieve maximum throughput using core accesses. Any cycle between consecutive reads to an DDR2 address results in non-sequential reads.

Listing 4-4. Maximum Throughput Using Sequential Reads

```
ustat1=dm(DDR2CTL0);
bit set ustat1 DDR2OPT|DDR2MODIFY1;
dm(DDR2CTL0)=ustat1;
nop;
I0 = DDR2_addr;
M0 = 1;
Lcntr = 1024, do(PC,1) until lce;
R0 = R0 + R1, R0 = dm (I0, M0);
```

The example shows read optimization can be used efficiently using core accesses. All reads are on the same page and it takes 2088 core cycles (core to DDR2 clock ratio 2:1) to perform 1024 reads.

Without read optimization, 1024 reads use 5125 core cycles if all of the reads are on the same page, non-sequential reads takes 9220 core cycles. With read optimization (Listing 4-5), 1024 reads take 10262 core cycles, due to the breaking of sequential reads.

Listing 4-5. Interrupted Reads With Read Optimization

```
ustat1=dm(DDR2CTL0);
bit set ustat1 DDR2OPT|DDR2MODIFY2;
dm(DDR2CTL0)=ustat1;
nop;
I0 = DDR2_addr;
M0 = 2;
Lcntr = 1024, do(PC,2) until lce;
R0 = R0 + R1, R0 = dm (I0, M0);
NOP;
```

Note the above mentioned cycles may vary based on different latency and timing parameters programmed.

DMA Access

Listing 4-6 shows an example of external port DMA using read optimization.

Listing 4-6. EPDMA With Read Optimization

```
ustat1=dm(DDR2CTL0);
bit set ustat1 DDROPT|DDRMODIFY2;
dm(DDR2CTL0)=ustat1;
nop;
r0=DFLSH;
dm(DMAC1)=r0;
r0=intmem;     dm(IIEP1)=r0;
r0=2;          dm(IMEP1)=r0;
```

```
r0=N;           dm(ICEP1)=r0;
r0=2;           dm(EMEP1)=r0;
r0=extmem;      dm(EIEP1)=r0;
r0=DEN;
dm(DMAC1)=r0;
```

Notes on Read Optimization

The core and the DMA engine take advantage of the major improvements during reads using read optimization. However, in situations where both the core and DMA need to read from different internal memory banks with different modifiers at the same time, programs need to choose whether or not to use optimization. Note that from a throughput prospective, external port arbitration also is a factor. A good rule is that the requester with the higher priority should have the same modifier as DDR2-MODIFY. In other words, if DMA has a higher priority over the core, then the DMA modifier should match the DDR2MODIFY setting.

Read optimization is only effective if the accesses are uninterrupted.

Self-Refresh Mode

This mode causes refresh operations to be performed internally by the DDR2, without any external control. This means that the controller does not generate any auto-refresh cycles while the DDR2 is in self-refresh mode.

Self-refresh entry—Self-refresh mode is enabled by writing a 1 to the DDR2SRF bit of the DDR2 memory control register (DDR2CTL0). This deasserts the DDR2CKE pin and puts the DDR2 in self-refresh mode if no access is currently underway. The DDR2 remains in self-refresh mode for at least t_{RAS} and until an internal access (read/write) to DDR2 space occurs or the SREF_EXIT bit in the DDR2CTL0 register is set.

The self-refresh entry command automatically disables the DDR2 memory DLL. Therefore its release command (exit) requires additional stall cycles until the DLL has re-locked.

Self-refresh exit. When any DDR2 access occurs, the controller asserts DDR2CKE high which causes the DDR2 to exit from self-refresh mode. The controller waits to meet the t_{XSNR} specification (exit with no read command) or the t_{XSRD} specification (exit with read command). Here is a significant difference; releasing with a read command requires 200 DDR2 cycles (since the memory DLL needs to re read memory).

System clock during self-refresh mode. Note that the DDR2CLK is not disabled by the controller during self-refresh mode. However, software may disable the clocks by setting the DIS_DDR2CTL bit in the DDR2CTL0 register. Programs should ensure that all applicable clock timing specifications are met before the transfer to DDR2 address space (which causes the controller to exit the self-refresh mode). If a transfer occurs to DDR2 address space when the DIS_DDR2CTL bit is cleared, an internal bus error is generated, and the access does not occur externally.

The following steps are required when using self-refresh mode.

1. Set the DDR2SRF bit to enter self-refresh mode.
2. Poll the DDR2SRA bit in the DDR2 status register (DDR2STAT) to determine if the DDR2 has already entered self-refresh mode.
3. Optionally: set the DIS_DDR2CTL bit to freeze DDR2_CLK.
4. Optionally: clear the DIS_DDR2CTL bit to re-enable DDR2_CLK.

DDR2 access occurs and the DDR2 exits from self-refresh mode.

The minimum time between a subsequent self-refresh entry and exit command is the t_{RAS} cycle. If a self-refresh request is issued during any external port DMA, the DDR2 controller grants the request with the t_{RAS} cycle and continues DMA operation afterwards.

Single-Ended Data Strobe

DDR2 data strobe mode is specified for either single ended or differential mode, depending on the setting of the EMR register enable DDR2_DQS mode bit. The timing advantages of differential mode are realized in system design.

The method by which the DDR2 pin timing is measured is mode dependent. In single ended mode, timing relationships are measured relative to the rising or falling edges of DDR2_DQS crossing at VREF. In differential mode, these timing relationships are measured relative to the crosspoint of DDR2_DQSS and its complement, $\overline{\text{DDR2\_DQS}}$. This distinction in timing methods is guaranteed by design and characterization. When differential data strobe mode is disabled via the EMR register, the complementary pin, $\overline{\text{DDR2\_DQS}}$, must be tied externally to VSS through a 20 Ω to 10 kΩ resistor to insure proper operation.

On Die Termination (ODT)

The DDR2 controller contains a separate pin (DDR2_ODT) that controls on-die termination. By default this pin is deasserted. If during power-up, the ODT register field in the DDR2CTL3 register is programmed with any Rtt value, the ODT pin is asserted after the power-up sequence has finished.

The level can be changed by forcing another power-up sequence which disables Rtt resistance in the ODT field. After completion, the ODT pin is deasserted. Note that the ODT pin control is independent on the DDR2 data access directions (read or write).

Additive Latency

Posted CAS operation helps maintain efficient and sustainable bandwidths in DDR2 SDRAM on the command and data bus. In this operation, the DDR2 SDRAM allows a CAS read or write command to be issued immediately after the RAS bank activate command (or any time during the RAS-CAS-delay time, t_{RCD}, period). The command is held for the time of the additive latency (AL) before it is issued inside the device. The read latency (RL) is controlled by the sum of AL and the CAS latency (CL).

Therefore if a program wants to issue a read/write command before the t_{RCD}min, then AL (greater than 0) must be written into the EMR(1) register. The write latency (WL) is always defined as RL – 1 (read latency – 1) where read latency is defined as the sum of additive latency plus CAS latency (RL = AL + CL). Read or write operations using AL allow seamless bursts (refer to seamless operation timing diagram examples in read burst and write burst section). Note that while the controller supports this feature, performance has nothing to do with the AL settings written to EMR1.

Forcing DDR2 Commands

The controller has some specific bits which can be used to aid in debug and in specific system solutions. If for example the part enters precharge power-down mode, explicit auto refresh commands need to be triggered (JEDEC standard). Or if during runtime the mode register settings and clock speed are changed.

By setting the SDPSS bit after reset, all mode registers are automatically updated. The SDPSS bit should be cleared when using forced commands.

Force Precharge All

Whenever an auto-refresh or a mode register set command is issued, the internal banks are required to be in idle state. Setting bit 21 (=1) forces a precharge all command to accomplish this. If the precharge all command is not issued, the auto-refresh and mode register set commands can be illegal depending on the current state.

Note that it is a good practice always to perform a force precharge all command before a forced refresh/mode register command.

Force Load Mode Register

Programs can use the Force LMR command by setting bit 22 (=1) in the `DDR2CTL0` register. The Force LMR bit allows changes to the `MODE` register based settings during runtime. These settings include bit 22 (=1) for MR command (settings `DDR2CTL2` register).

Force Auto-Refresh

Bit 20 (=1) in the `DDR2CTL0` register forces the auto refresh to be immediately executed (not waiting until the refresh counter has expired). This is useful for test purposes but also to synchronize the refresh time base with a system relevant time base.

Force Extended Mode Register 1–3

Programs use the Force extended mode register 1–3 commands (`DDR2CTL0` register) by setting:

bit 23 (=1) for EMR1 command (settings `DDR2CTL3` register)
bit 12 (=1) for EMR2 command (settings `DDR2CTL4` register)
bit 17 (=1) for EMR3 command (settings `DDR2CTL5` register)

This allows programs to initialize or change the content of the `EMR` register.

Force DLL External Bank Calibration

The last step during power up is the post calibration of the external DDR2 banks. This command is enabled by setting bit 13 (=1) in the DDR2CTL0 register. If enabled the DDR2 controller posts 300 dummy reads for calibration between the internal DDR2 clock and the DDR2_DQS1-0 pins which are driven during the read. Note the calibration is done separately for each assigned external bank.

Shared Memory Interface (ADSP-2146x)

The ADSP-2146x processor supports connections to a common shared external memory of up to two other ADSP-2146x processors. These connections create shared external bus processor systems.

Features

- Shared memory space for all four external DDR2 banks
- Supports shared data or instruction fetch
- Distributed, on-chip arbitration for the shared DDR2 bus
- Bus lock feature support for semaphore implementation
- Bus master time-out for arbitration fairness

Figure 4-23 illustrates a basic shared memory system. In a system with several processors sharing the external bus, any of the processors can become the bus master. The bus master has control of the bus, which consists of the DDR2 control and address/data signals and associated control lines.

Note that the ADSP-2146x owns two separate and independent external port buses, one for the AMI and the other for the DDR2.

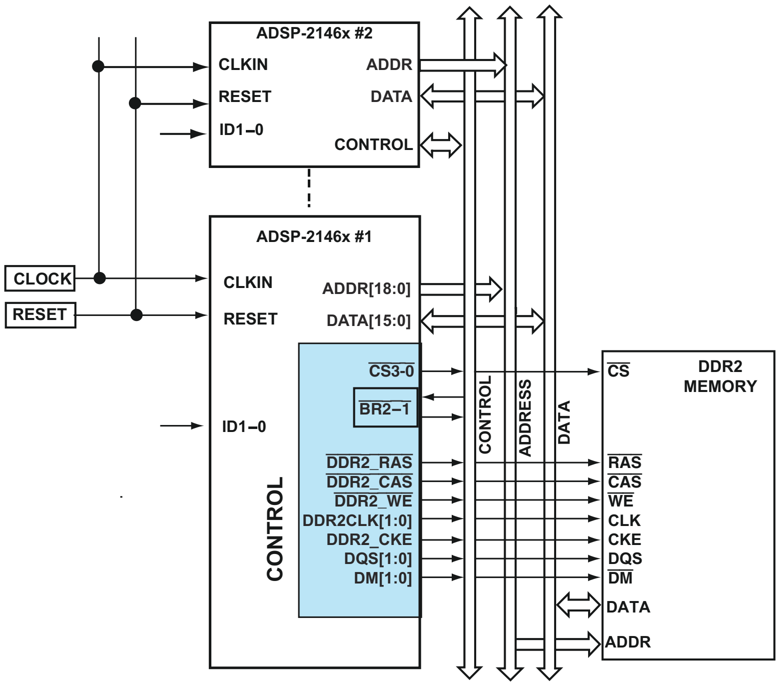

Figure 4-23. Shared DDR2 Memory System

Pin Descriptions

The pins used by the shared external memory interface are described in the *ADSP-21467/ADSP-21469 Processor Data Sheet*.

Functional Description

Multiple processors can share the external bus with no additional arbitration logic as shown in Figure 4-23. Arbitration logic is included on chip to allow the connection of up to two ADSP-2146x processors.

The processor accomplishes bus arbitration through the `BR2-1` signals which arbitrate between the two processors.

The ID2-1 pins provide a unique identity for each processor in a multiprocessing system. The first processor should be assigned ID = 1, the second should be assigned ID = 2. One of the processors must be assigned ID = 1 in order for the bus synchronization scheme to function properly.

The processor with ID = 1 holds the external bus control lines stable (pull-up enabled) during reset.

A processor in a shared memory system can determine which processor is the current bus master by reading the CRBM bits of the SYSTAT register. These bits provide the values of the ID2-1 inputs of the current bus master.

Only DDR2CKR ratios of 1:2 and 1:4 are supported in multiple processor system. Other ratios do not work because of unaligned clocks. The PCLK and CLKIN clocks are used in the arbitration logic for the shared external bus. The multiprocessor logic requires that these clocks need to be rising edge aligned to function properly. Therefore, not all core to DDR2 clock ratios are allowed in multiple processor system. The PLL bit settings PLLM/PLLD in PMCTL register need to be programmed such that the PLLM/PLLD ratio is integer (for example 15/2=7.5 fractional, is not allowed).

Bus Transition Cycle

The bus request ($\overline{\text{BR1-0}}$) pins are connected between each processor in a shared memory system, where the number of $\overline{\text{BRx}}$ lines used is equal to the number of processors in the system. Each processor drives the $\overline{\text{BRx}}$ pin that corresponds to its ID2-1 inputs and monitors all others.

When the slave processor needs to perform an access to the shared memory space, it needs to become bus master, it automatically initiates the bus arbitration process by asserting its $\overline{\text{BRx}}$ line at the beginning of the cycle. Later in the same cycle, the processor samples the value of the other $\overline{\text{BRx}}$ lines.

The cycle in which mastership of the bus is passed from one processor to another is called a *bus transition cycle* (BTC). A BTC occurs when the current bus master's $\overline{\text{BRx}}$ pin is deasserted and the slave's $\overline{\text{BRx}}$ pin is asserted. The bus master can retain bus mastership by keeping its $\overline{\text{BRx}}$ pin asserted.

By observing all of the $\overline{\text{BRx}}$ lines, each processor can detect when a bus transition cycle occurs and which processor has become the new bus master. A bus transition cycle is the only time that bus mastership is transferred.

The actual transfer of bus mastership, shown in Figure 4-24, is accomplished by the current bus master three-stating the DDR2 bus signals—at the end of the bus transition cycle and the new bus master driving these signals at the beginning of the next cycle.

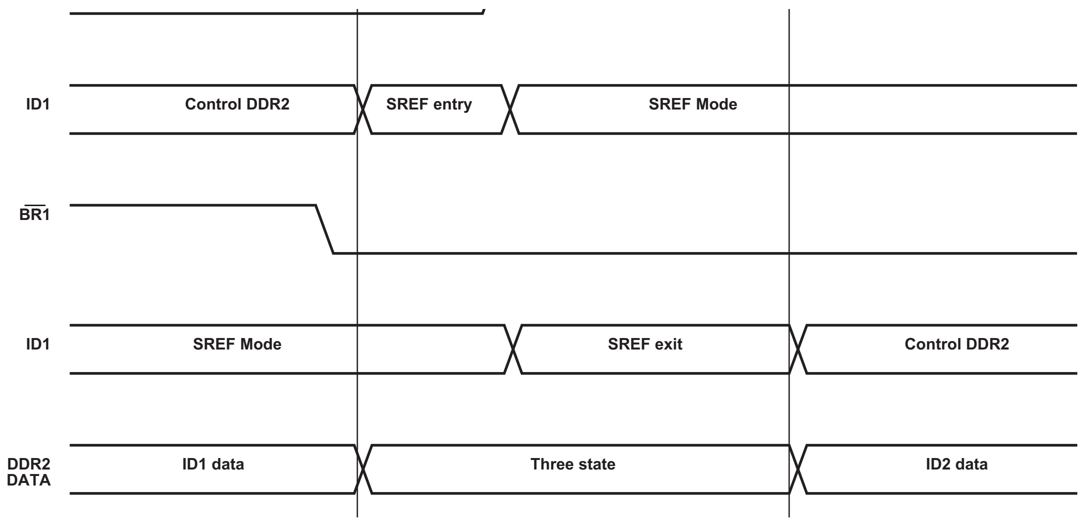

Figure 4-24. DDR2 Bus Mastership Transfer

During bus transition cycle delays, execution of external accesses are delayed. When one of the slave processors needs to perform a read or write to the shared memory space, it automatically initiates the bus arbitration process by asserting its $\overline{\text{BRx}}$ line. This read or write is delayed until the processor receives bus mastership. If the read or write was generated by the processor's core (not the DMA controller), program execution stops on that processor until the instruction is completed.

Any slave requester of an ADSP-2146x can't interrupt a current DMA to the shared memory, it has to wait until the DMA has completed.

The following steps occur as a slave acquires bus mastership and performs an external read or write over the DDR2 bus.

1. The slave determines that it is executing an instruction which requires an off-chip access. It asserts its $\overline{\text{BRx}}$ line at the beginning of the cycle. Extra cycles are generated by the core processor (or DMA controller) until the slave acquires bus mastership.

2. To acquire bus mastership, the slave waits for a bus transition cycle in which the current bus master deasserts its $\overline{\text{BRx}}$ line. The slave becomes bus master in the next cycle.

3. At the end of the BTC, the current bus master releases the bus and the new bus master starts driving.

During the CLKIN cycle in which the bus master deasserts its $\overline{\text{BRx}}$ output, it three-states its outputs in case another bus master wins arbitration and enables its drivers in the next CLKIN cycle. If the current bus master retains control of the bus in the next cycle, it enables its bus drivers, even if it has no bus operation to run.

The fundamental clock for the bus arbitration is the CLKIN input. Therefore all bus members must share the same CLKIN oscillator. Note the higher CLKIN the higher the bus arbitration speed.

When the bus master stops using the bus, its $\overline{\text{BRx}}$ line is deasserted, allowing other processors to arbitrate for mastership if they need it. If no other processor is asserting its $\overline{\text{BRx}}$ line when the master deasserts its $\overline{\text{BRx}}$, the master retains control of the bus and continues to drive the memory control signals until:

1. it needs to use the bus again
2. another processor asserts its $\overline{\text{BRx}}$ line

DDR2 Bus Mastership Transfer

The DDR2 memory is shared among two ADSP-2146x processors.

Both processors must have the same configured DDR2 frequency, and the same core to DDR2 clock ratio. This implies that both processors must use the same controller settings in their respective control (DDR2CTL5-0) and refresh rate (DDR2RRC) registers.

The bus master ownership is switched between the processors by using additional self-refresh entry and exit commands to perform the bus transition cycle. The DDR2 memory is automatically entered in self-refresh mode before releasing the bus and its clock is three-stated for one DDR2CLK cycle. The new bus master executes a self-refresh exit command immediately after acquiring mastership. Since the DDR2 memory is in self-refresh mode during the BTC cycle the command bus state is undefined preventing it from latching invalid commands due to glitches on clock.

The slave processor does not track DDR2 commands on the bus. After getting bus mastership the processor clears the current refresh counter value (DDR2RRC) for auto-refresh and issues an auto-refresh command. This simplifies the design and avoids maintaining the refresh counters in sync on both processors. The bus master transfer is executed in the five phases listed below.

1. Memory access of current bus master, slave asserts its $\overline{\text{BR}}$ signal
2. Current bus master enters SREF mode and de-asserts $\overline{\text{BR}}$ signal
3. BTC cycle
4. New bus master releases SREF mode
5. Memory access of new bus master

The time for the entire bus mastership transfer is:
SREF Entry (bus release) + BTC + SREF Exit (bus request) cycles

Where:

SREF Entry = 1 CLKIN + t_{RP} cycles

SREF Exit = 1 CLKIN + 8 DDR2CLK + 2x t_{XSNR} (for non read command after self refresh exit)

SREF Exit = 1 CLKIN + 4 DDR2CLK + t_{XSRD} (for read command after self refresh exit)

Bus Synchronization After Reset

When a shared memory system comes out of reset (after $\overline{\text{RESET}}$ is de-asserted), the bus arbitration logic on each processor must synchronize, ensuring that only one processor drives the external bus. One processor must become the bus master, and all other processors must recognize it before actively arbitrating for the bus. The bus synchronization scheme also lets the system safely bring individual processors into and out of reset.

One of the processors in the system must be assigned ID = 1 in order for the bus synchronization scheme to function properly. This processor also holds the external bus control lines stable during reset.

Bus arbitration and synchronization are disabled if the processor is in a single processor system (ID = 0).

To synchronize their bus arbitration logic and define the bus master after a system reset, the multiple processors obey the following rules:

- The processor with ID = 2 de-assert its $\overline{\mathrm{BRx}}$ line during reset. It keeps its $\overline{\mathrm{BRx}}$ deasserted for at least two cycles after reset and until their bus arbitration logic is synchronized.
- After reset, a processor considers itself synchronized when it detects a cycle in which only one $\overline{\mathrm{BRx}}$ line is asserted. The processor identifies the bus master by recognizing which $\overline{\mathrm{BRx}}$ is asserted and updates its internal record to indicate the current master.
- The processor with ID = 1 asserts its $\overline{\mathrm{BRx}}$ during reset and for at least two cycles after reset. If the other $\overline{\mathrm{BRx}}$ line is asserted during these cycles, the processor with ID = 1 drives the memory control signals to prevent glitches. Although the processor with ID = 1 is asserting its $\overline{\mathrm{BRx}}$ and driving the memory control signals during these cycles, this processor does not perform reads or writes over the bus.
- While in reset, the processor with ID = 1 attempts to gain control of the bus by asserting $\overline{\mathrm{BR1}}$.
- While in reset, the processor with ID = 1 drives the DDR2 signals only if it determines that it has control of the bus. For the processor to decide it has control of the bus: 1) its $\overline{\mathrm{BR1}}$ signal must be asserted and 2) in the previous cycle, no other processor's $\overline{\mathrm{BRx}}$ signals were asserted.

The processor with ID = 1 continues to drive the DDR2 signals for two cycles after reset, as long as other $\overline{\mathrm{BRx}}$ lines are asserted.

If the processor with ID = 1 is synchronized by the end of the two cycles following reset, it becomes the bus master. If it is not synchronized at this time, it deasserts its $\overline{\mathrm{BRx}}$ signal and stops driving the memory control signals and does not arbitrate for the bus until it becomes synchronized.

When a processor has synchronized itself, it sets the BSYN bit in the SYSTAT register. Note that the BSYN bit is set after de-assertion of the RESETOUT pin for a minimum delay of 1 CLKIN cycle or more.

If one processor comes out of reset after the other has synchronized and started program execution, that processor may not be able to synchronize immediately (for example, if it detects more than one BRx line asserted). If the non-synchronized processor tries to execute an instruction with an off-chip read or write, it cannot assert its BRx line to request the bus and execution is delayed until it can synchronize and correctly arbitrate for the bus.

The FSYNC bit in SYSCTL register is provided to force synchronizing the cluster system. When set this bit enables synchronization and when cleared disables synchronization of the system.

The current bus master during regular operation should not be reset (hardware reset), as this would result in system synchronization problems.

Operating Modes

The following sections describe the operating modes that can be used with shared memory.

Bus Mastership Time-Out

Systems may need to limit how long a bus master can retain the bus. This is accomplished by forcing the bus master to deassert its BRx line after a specified number of CLKIN cycles and giving the other processors a chance to acquire bus mastership.

To set up a bus master time-out, a program must load the bus time-out maximum (BMAX register, 16-bit) with the maximum number of CLKIN cycles (minus 2) that allows the processor to retain bus mastership. This equation is shown below.

BMAX = (maximum number of bus mastership CLKIN cycles) – 2

The minimum value for BMAX is 2, which lets the processor retain bus mastership for four CLKIN cycles. Setting BMAX=1 is not allowed. To disable the bus master time-out function, set BMAX=0.

Each time a processor acquires bus mastership, its bus time-out counter (BCNT register, 16-bit) is loaded with the value in BMAX. The BCNT is then decremented in every CLKIN cycle in which the master performs a read or write over the bus and any other (slave) processors are requesting the bus. Any time the bus master deasserts its $\overline{\text{BRx}}$ line, BCNT is reloaded from BMAX.

When BCNT decrements to zero, the bus master first completes its off-chip read/write and then deasserts its own $\overline{\text{BRx}}$ (any new off-chip accesses are delayed), which allows transfer of bus mastership.

If BCNT reaches zero while bus lock is active, the bus master does not deassert its $\overline{\text{BRx}}$ line until bus lock is removed. Bus lock is enabled by BUSLK (bit 29 of SYSCTL register). For more information, see “Bus Transition Cycle” on page 4-110.

During any access (core/external port DMA), the new master requesting the bus must wait until the present master releases the bus. The new master can not interrupt current bus master transfers. If the current master doesn’t have an external transfer, it releases the bus (even before BCNT = 0).

If BCNT reaches zero while bus lock bit is set, the bus master does not de-assert its $\overline{\text{BRx}}$ line until bus lock bit is cleared.

Bus Lock

With the use of its bus lock feature, the processor has the ability to read and modify a semaphore in a single indivisible operation – a key requirement of multiple processor systems.

Semaphores can be used in multiple processor systems to allow the processors to share resources such as memory. A semaphore is a flag that can be read and written by any of the processors sharing the resource. The value of the semaphore tells the processor when it can access the resource. Semaphores are also useful for synchronizing the tasks being performed by different processors in a system.

Read-modify-write operations on semaphores can be performed if all of the processors obey two simple rules.

- A processor must not write to a semaphore unless it is the bus master.
- When attempting a read-modify-write operation on a semaphore, the processor must have bus mastership for the duration of the operation.

Both of these rules apply when a processor uses its bus lock feature, which retains its mastership of the bus and prevents the other processors from simultaneously accessing the semaphore.

Bus lock is requested by setting the `BUSLK` bit in the `SYSCTL` register. When this happens, the processor initiates the bus arbitration process by asserting its $\overline{\texttt{BRx}}$ line. When it becomes bus master, it locks the bus by keeping its $\overline{\texttt{BRx}}$ line asserted even when it is not performing an external read or write. When the `BUSLK` bit is cleared, the processor gives up the bus by de-asserting its $\overline{\texttt{BRx}}$ line.

While the `BUSLK` bit is set, the processor can determine if it has acquired bus mastership by executing a conditional instruction with the Bus Master (BM) or Not Bus Master (Not BM) condition codes, (Refer to *SHARC Processor Programming Referenc*e.) For example:

```
IF NOT BM JUMP(PC,0); /* Wait for bus mastership */
```

If it has become the bus master, the processor can proceed with the external read or write. If not, it can clear its BUSLK bit and tries again later. A read-modify-write operation is accomplished with the following steps.

1. Request bus lock by setting the BUSLK bit.
2. Wait for bus mastership to be acquired.
3. Read the semaphore, test it, then write to it.

Note that locking the bus prevents the other processor from writing to the semaphore while the read-modify-write is occurring.

Data Transfer

The external port has two buffers, the AMI which requires two buffer for packing/unpacking 8/16-bit to 32-bit data. The DMA has a data buffer for each of the DMA channels.The AMI can access data from both the core and through DMA. The following sections describe these options.

Data Buffers

The asynchronous memory interface has two 1 deep data buffers, one each for the transmit and receive operations. These buffers pack data or instruction during external port boot.

Receive Buffer Unpacking

Reads from external memory are done through the 1 deep receive packing buffer (AMIRX). When an external address that is mapped to the AMI in the EPCTL register is accessed, it receives 8/16-bit data and packs the data based on the packing and control modes in the AMI control register (AMICTLx). Once a full packed word is received, the internal status signal is deasserted and new reads are allowed.

The AMI provides the interface to the external data pins as well as to the processor core or to the internal DMA controller. When the AMI receives data, it is passed by internal hardware to the DMA controller or to the external port control bus, depending on which entity requested the data.

Transmit Buffer Unpacking

Writes to external memory are done through the 1 deep transmit packing buffer (AMITX). When an external address that is mapped to the AMI in the EPCTL register is accessed, it receives data from internal memory using the DMA controller or through direct core writes.

Once a full word is transferred out of the AMI, the internal status signal is deasserted and new writes are allowed. No more external transfers can start while the AMI module is not empty.

Whenever the AMITX buffer is empty, the DMA controller or a direct access from the processor core can write new data into the AMI. If the register is full, further writes from the core (or DMA controller) are stalled.

> For core and DMA access, the received data is also unpacked, depending on the setting of the PKDIS bit. The order of unpacking is dependent on the MSWF bit in AMICTLx registers.

External Port DMA Buffer

The external port supports two DMA channels, each populated with a data buffer (DFEP1-0). Each data buffer is 6 locations deep and its status can be read in the DMACx register. Note the DMA channels are valid for AMI, SDRAM or DDR2 transfers. For more information, see “External Port DMA” on page 4-125.

Buffer Status

The entire path form a 6-stage buffer. Six writes/reads can occur to the transmit/receive buffer by the DMA before it signals a full/empty condition. Full/empty status for the DMA buffer is read by the `DFS` bits in the `DMACx` register.

Flush Buffer

The AMI and the external port DMA buffers are flushed when the `FLSH` bit (AMI) and `DFLSH` bit (external port) are set.

Core Access

For core-driven external port transfers, the instruction needs to read or write from a valid external port address.

External Port Dual Data Fetch

The dual data fetch instruction (Type 1) allows the processor to access external data from both DAGs. In such an instruction, the accesses are executed sequentially (not simultaneously as in internal memory). For example:

```
r4=r2+r3, r2=dm(i6,m6), r3=pm(i10,m10);
```

The DAG1 access (operand `r2`) is executed first followed by the second DAG2 access (operand `r3`).

Conditional Instructions

On the SHARC processors, almost all instruction types can be conditional. Access to external data based on a conditional instructions are allowed. For example:

```
r10=pass r9;
If EQ r4=r2+r3, r2=dm(i6,m6);
```

The instruction is only executed if the condition is true.

SIMD Access

The SHARC processor supports SIMD data access from external memory for SDRAM and DDR2 memory space in normal space only.

In SIMD mode, the core expects 64-bit data on a single read request and drives 64-bit data for write requests. The controller decodes the access request and if it is a SIMD read from a location N, the controller fetches data from N and N+1, irrespective of whether N is an odd or an even address.

The memory controller then packs the data into 64 bits and sends it back along the core buses. For a SIMD write, the controller unpacks the 64-bit data given by the core and writes it to N and N+1 memory locations.

The behavior of SIMD access to/from external memory is similar to the internal processor memory. The only difference is that it is supported in normal word (32-bit) address space only. Unlike internal memory access, SIMD access from external memory may have a different latency, the explicit transfer terminate first followed by the implicit transfer.

SDRAM

SIMD mode transfers are performed within 2 core accesses. The first access performs an explicit 32-bit access (which results in the physical space in 2x16-bit words) while the 2nd 32-bit access performs the implicit transfer. In total there are four read or write commands as shown in Table 4-22.

Table 4-22. SDRAM SIMD Access

Access	Logical x32	Physical x16	Comment
Explicit	0x20 0000	0x40 0000 = LS word 0x40 0001 = MS word	No Masking
Implicit	0x20 0001	0x40 0002 = LS word 0x40 0003 = MS word	No Masking

SIMD mode access is not supported in the asynchronous memory interface (AMI).

DDR2

For a SIMD transfer, the controller can burst the 64-bit data given by the core and writes it to N and N+1 memory locations. Since the burst length is 4, SIMD mode transfers can be performed within one burst access.

The first access performs the explicit 32-bit access (which results in the physical space in 2x16-bit words) while the 2nd 32-bit access performs the implicit transfer. In total there is one read or write command per burst of 4x16-bit data as shown in Table 4-23.

Table 4-23. DDR2 SIMD Burst Access

Access	Logical x32	Physical x16	Comment
Explicit	0x20 0000	0x40 0000 = LS word 0x40 0001 = MS word	No masking
Implicit	0x20 0001	0x40 0002 = LS word 0x40 0003 = MS word	No masking

Bursts are not divisible. During reads, all DDR2 data are received and on-chip masked by the DDR2 controller. For single write access in SISD mode, the 3rd and 4th data needs to be masked. The data masking (`DDR2_DM1-0` signal) is only performed during write operations as shown in Table 4-24.

Table 4-24. DDR2 SISD Access

Access	Logical x32	Physical x16	Comment
Explicit only	DM(0x200000) = R0;	0x40 0000 = LSW 0x40 0001 = MSW 0x40 0002 = LSW 0x40 0003 = MSW	No masking Masking required (burst)

SIMD write access to the DDR2 memory should be even address aligned. If odd address aligned, the throughput is reduced by a factor of 2. This does not apply to SIMD reads or any SISD mode. For more information on SIMD access, see *SHARC Processor Programming Reference*.

External Port DMA

The external port has two DMA channels that can use either the SDRAM/DDR2 controller or the asynchronous memory interface (AMI). The AMI controller supports DMA with an external data width of 8 or 16 bits. The SDRAM/DDR2 controllers support DMA with an external data width of 16-bits.

Features

The external port has the following features and capabilities.

- Two DMA channels
- Standard mode
- Chained Mode with direction on the fly
- Tap List Mode (Scatter/Gather)
- Delay Line Mode (Write to Read)
- All these modes can operate in circular fashion
- In circular operation some modes allow write back of index pointer for correct addressing of next transfer control block (TCB)
- Addressing from internal to external or internal to internal

DMA Parameter Registers

These registers are used to set up and control DMA through the processor's external port. For information on these registers and on how to set up DMA transfers, see "General Procedure for Configuring DMA" on page 3-50. The registers that control external port DMA are described Table 4-25.

Table 4-25. DMA Parameter Registers

Register	Description	Comment
IIEPx	Internal Index	Internal Start Address. For delay line DMA, it serves as the delay line write index; for example, the start address of the internal memory buffer for the external write data.
IMEPx	Internal Modifier	Internal address modifier.
ICEPx	Internal Count	For delay line DMA, it serves as count for delay line writes, write block size.
EIEPx	External Index	External start address. In delay line DMA this address is written back to internal memory once the writes completes (ICEP=0)
EMEPx	External Modifier	External address modifier.
ECEPx	External Count	External memory count, read only (alias of ICEPx)
CPEPx	Chain Pointer	Contains address of the next descriptor in internal memory.

Table 4-26. Enhanced DMA Parameter Registers

Register	Description	Comment
ELEPx	Circular Buffer Length	Hold circular buffer length for circular, delay line DMA, scatter/gather DMA.
EBEPx	External Base	Hold circular start address for circular, delay line DMA, scatter/gather DMA.
RIEPx[1]	Read Internal Index	Contains start address of internal memory buffer to which the data read from external memory during delay line DMA reads are to be written into (alias of IIEPx during delay line read DMA).
RCEPx[1]	Read Internal Count	Contains number of reads from each tap list, read block size (alias of ICEPx during delay line read DMA).
RMEPx[1]	Read External Modifier	Contains external modifier to be used for delay line reads (alias of EMEPx during delay line read DMA).

Table 4-26. Enhanced DMA Parameter Registers (Cont'd)

Register	Description	Comment
TCEPx	Tap Count	Holds the length of the tap list, number of taps. Applies to delay line DMA, scatter/gather DMA.
TPEPx	Tap List Pointer	Holds address of an array in internal memory which holds offsets to be used when accessing delay line DMA in external memory. The offset represents the first address of each read block. Applies to delay line DMA, scatter/gather DMA

1 These registers are only accessible through the TCB loading.

Functional Description

The external port DMA supports two different DMA channels with different addressing types and DMA modes described below.

DMA Addressing

Besides the traditional internal to external addressing type, the DMA module also supports internal to internal transfers. This is accomplished by indexing all external parameter registers with internal addresses. The DMA controller recognizes the transfer by addresses and not by an additional control bit setting.

Note that the DMA channel priority changes if using internal vs. external index addresses.

The SHARC supports another internal to internal DMA module (MTM) which has higher default priority but only supports standard DMA mode. For more information, see Chapter 6, "Memory-to-Memory Port DMA".

Operating Modes

This section and Table 4-27 highlight the different DMA modes which can be used with the external port. The complete register bit descriptions are in "External Port DMA Control Registers (DMACx)" on page A-23.

Table 4-27. DMACx Register Bit to Operating Modes

<table>
<tr><th>Bit (Name)</th><th>Standard</th><th>Chained</th><th>Scatter/Gather</th><th>Delay Line</th></tr>
<tr><td colspan="5">Control Bits</td></tr>
<tr><td>0 (DEN,)</td><td colspan="4">Used</td></tr>
<tr><td>1 (TRAN)</td><td colspan="3">Used</td><td>Reserved</td></tr>
<tr><td>2 (CHEN)</td><td colspan="4">Used</td></tr>
<tr><td>3 (DLEN)</td><td colspan="3">Reserved</td><td>Used</td></tr>
<tr><td>4 (CBEN)</td><td colspan="4">Used</td></tr>
<tr><td>5 (DFLSH)</td><td colspan="4">Used</td></tr>
<tr><td>6</td><td colspan="4">Reserved</td></tr>
<tr><td>7 (WRBEN)</td><td>Reserved</td><td>Used</td><td>Reserved (=0)</td><td>Reserved (=1)</td></tr>
<tr><td>8 (OFCEN)</td><td colspan="3">Used</td><td>Reserved</td></tr>
<tr><td>9 (TLEN)</td><td colspan="2">Reserved</td><td>Used</td><td>Reserved</td></tr>
<tr><td>11–10</td><td colspan="4">Reserved</td></tr>
<tr><td>12 (INTIRT)</td><td colspan="4">Used</td></tr>
<tr><td>15–13</td><td colspan="4">Reserved</td></tr>
<tr><td colspan="5">Status Bits</td></tr>
<tr><td>17–16 (DFS)</td><td colspan="4">Used</td></tr>
<tr><td>19–18</td><td colspan="4">Reserved</td></tr>
<tr><td>20 (DMAS)</td><td colspan="4">Used</td></tr>
<tr><td>21 (CHS)</td><td>Reserved</td><td>Used</td><td>Reserved</td><td>Used</td></tr>
<tr><td>22 (TLS)</td><td colspan="2">Reserved</td><td>Used</td><td>Reserved</td></tr>
<tr><td>23 (WBS)</td><td colspan="3">Reserved</td><td>Used</td></tr>
</table>

Table 4-27. DMACx Register Bit to Operating Modes (Cont'd)

Bit (Name)	Standard	Chained	Scatter/Gather	Delay Line
24 (EXTS)	Used			
25 (DIRS)	Used			
31–26	Reserved			

The additional bit-field information for the reserved field indicate how the hardware operates internally. Reading these "reserved" bits however does not generate a meaningful result.

Standard DMA

This DMA type resembles the traditional DMA type to initialize the different internal and external parameters (index, modify and count) registers and configuration of the DMA control registers.

Note that the ECEP parameter register (read only) is a copy of the ICEP register. If ICEP is written, the ECEP register is updated automatically (Figure 4-25).

Circular Buffered DMA

Circular buffered DMA (Figure 4-26, Figure 4-27) resembles the traditional core DAG circular buffered mode by using registers for circular buffering. In this mode the DMA needs two additional registers (base and length) to support reads and writes to a circular buffer.

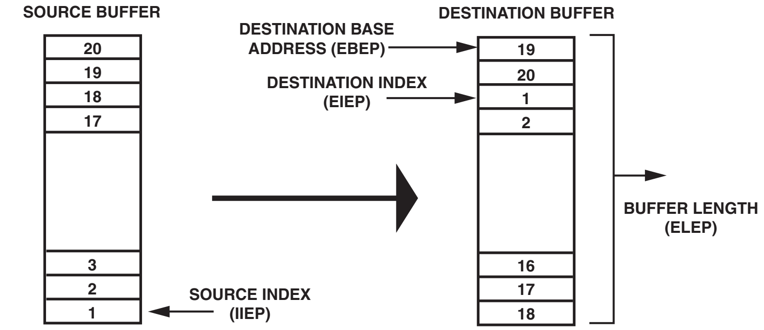

Figure 4-26. Circular Buffering Write DMA

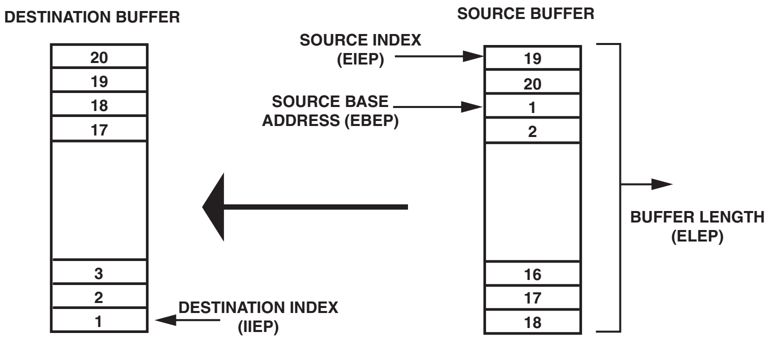

Figure 4-27. Circular Buffering Read DMA

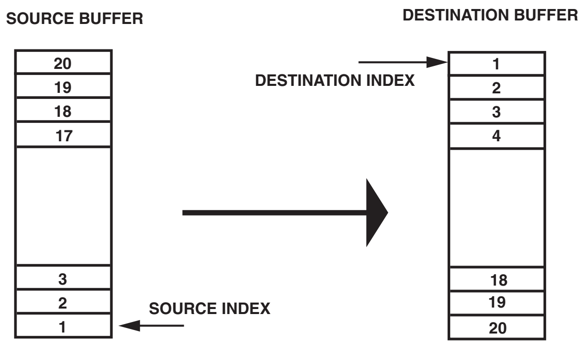

Figure 4-25. Standard Write

Note circular operation is available for all operating modes (standard, Chained, tap list and delay line DMA).

Chained DMA Mode

Chained DMA is used to support automated access by a linked list (repetitive reads and writes to a defined location defined by the individual TCBs). Setting the CHEN bit, the corresponding TCB storage must be selected (for non-circular or circular mode). See "TCB Memory Storage" on page 3-32. Note that for the delay line DMA the CHEN bit must be set (not optional).

Data Direction on the Fly

The SHARC processors allow a change of external port data direction for each individual TCB in a chain sequence.

As shown in Listing 4-7, the CPDR bit of the external port chain pointer register (CPEPx) changes the data flow direction. If CPDR is cleared (=0) writes to internal memory are performed, if CPDR is set (=1), internal memory reads are performed. This works similar to the PCI bit. The OFCEN and CHEN bits in the DMACx register must be set (=1) to enable this functionality.

Listing 4-7. Changing DMA Direction

```
.section/pm seg_dmda;
/* EP TCB storage order CP-EM-EI-C-IM-II */
.var TCB1[6] = 0 , M , extbuffer , N , M , buffer;
.var TCB2[6] = 0 , M , extbuffer , N , M , buffer;

.section/pm seg_pmco;
R0=0;
dm(CPEP0)=R0;              /* clear CPx register */

r0 = DEN|CHEN|OFCEN;       /* enable DMA channel */
```

```
dm(DMAC0)=r0;

R2=(TCB1+5) & 0x7FFFF;    /* load IIx address of next TCB and
                              mask address */
R2=bset R2 by 19;           /* set PCI bit */
dm(TCB2)=R2;              /* write address to CPx location of
                              current TCB */
R2=(TCB2+5) & 0x7FFFF;      /* load IIx address of next TCB and
                                mask address*/

R2=bset R2 by 19;             /* clear PCI bit */
R2=bset R2 by 20;             /* set CPDR bit */
dm(TCB1)=R2;                /* write address to CPx location of
                                current TCB */
dm(CPEP0)=R2;               /* write IIx address of TCB1 to CPx
                               register to start chaining*/
```

If chaining is enabled with the OFCEN bit set then the TRAN bit has no effect, and direction is determined by the CPDR bit in the CPEP register.

Write Back Circular Index Pointer

Operating the DMA in circular mode requires some special considerations. The index pointer of start address within the buffer may wrap around for the case if IC × IM > EL or does not finish if IC × IM < EL. In both cases the TCB start address is no longer valid.

Setting the WRBEN bit writes (at the end of current TCB block) the current index address + 1 into the TCB memory which is the start address for the next TCB. This bit is only selectable for chained DMA mode, for tap list and delay line modes this bit is hardwired to 0 or 1.

Scatter/Gather DMA

The purpose of scatter/gather DMA (Table 4-28, and Figure 4-28 through Figure 4-31 on page 4-138) is the transfer of data from/to non contiguous memory blocks.

The scatter/gather DMA type is a fixed block size scatter/gather DMA that relies on tap list entries in internal memory to calculate the external address to scatter/gather the DMA. If the DMA direction is external write (TRAN = 1) then it is a scatter DMA. If TRAN = 0 then it is a gather DMA. This mode also supports chained and circular buffer chained DMAs.

Table 4-28. Read/Write Index Pre-Modify (Scatter/Gather DMA)

Pre-Modify Address Equation	**Result**	
EIEP + TPEP[TCEP] + (EMEPx ICEP)	**Blocksize**	**Tap**
EIEP + TPEP[0] + EMEPx1	N	0
EIEP + TPEP[0] + EMEPx2		
EIEP + TPEP[0] + EMEPx3		
EIEP + TPEP[0] + EMEPxN		
EIEP + TPEP[1] + EMEPx1	N	1
EIEP + TPEP[1] + EMEPx2		
EIEP + TPEP[1] + EMEPx3		
EIEP + TPEP[1] + EMEPxN		
EIEP + TPEP[M] + EMEPx1	N	M
EIEP + TPEP[M] + EMEPx2		
EIEP + TPEP[M] + EMEPx3		
EIEP + TPEP[M] + EMEPxN		

Pre Modified Read/Write Index

For scatter/gather DMA, the tap list modifiers are employed and the number of taps is determined by the tap list count register (TCEPx). The number of sequential reads (block size) from every tap is determined by the internal count register (ICEPx), and is the same for every tap. The read/write pointer in external index register (EIEPx) serves as the index address for these read/writes.

TL[N] is the first tap list entry in the internal memory as pointed by the TPEP, the tap list pointer. The tap list entries are 27-bit signed integers. Therefore, for each read/write block, the DMA state machine fetches the offset from the tap list. The offset is added to the EIEP value to get the start address of the next block. The external addresses are circular buffered if circular buffering is enabled (Figure 4-30, Figure 4-31).

Once the ICEP register for the final tap decrements to zero (both TCEP and ICEP are zero), then the tap list DMA access is complete and the DMA completion interrupt is generated (if chaining is enabled the interrupt depends on the PCI bit setting).

The write back mode (WRBEN bit) is hardwired to zero for tap list based DMA (as the addressing is pre-modify, and therefore the EIEP value coincides with the TCB value even at the end of the DMA).

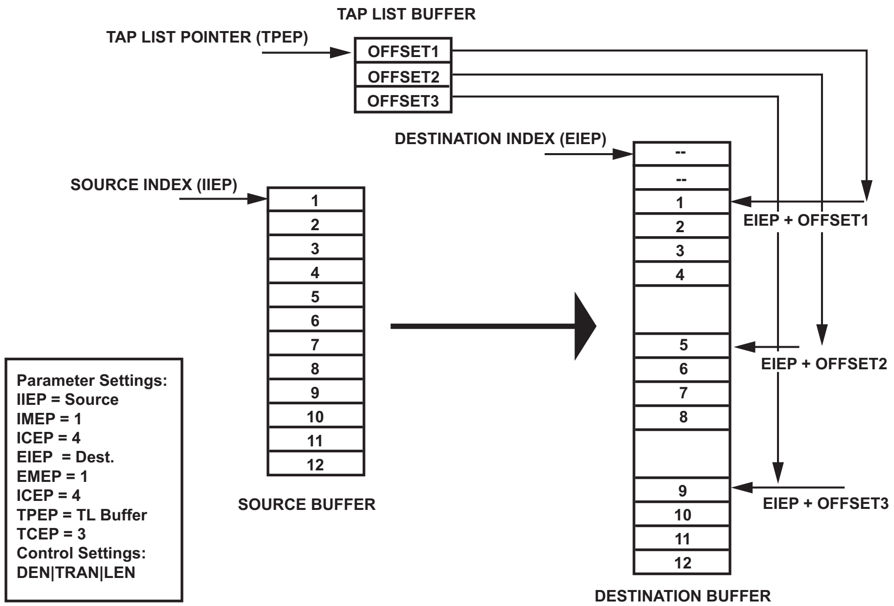

Figure 4-28. Scatter DMA (Writes)

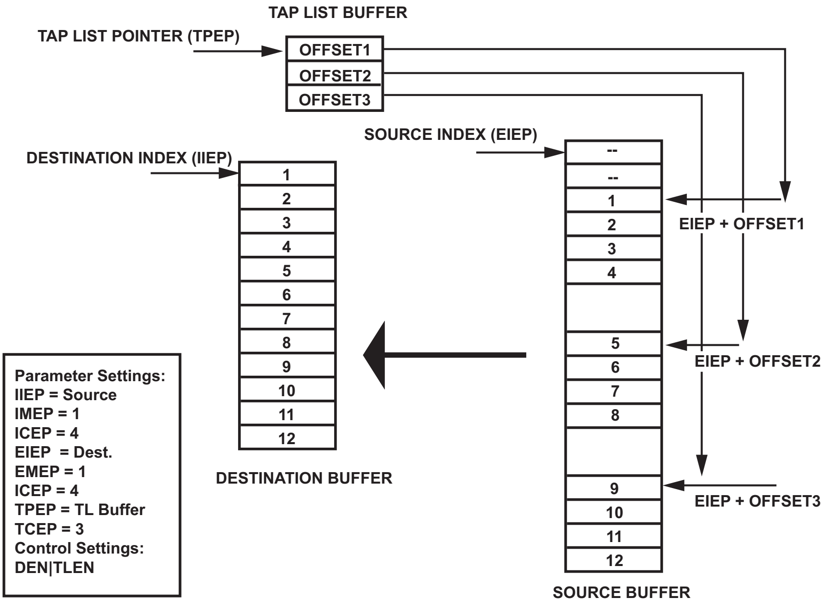

Figure 4-29. Gather DMA (Reads)

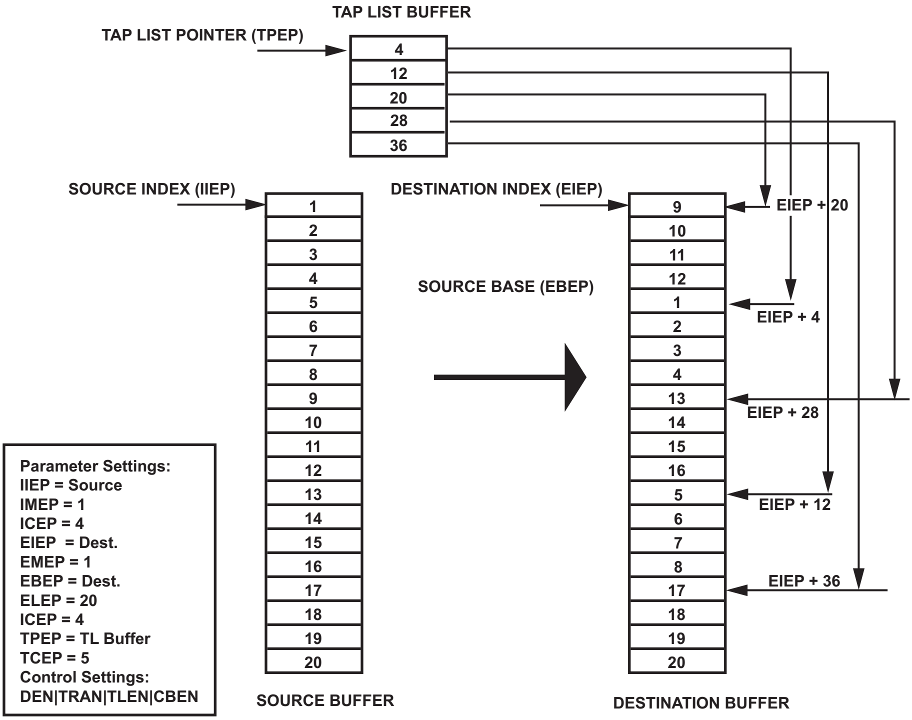

Figure 4-30. Circular Buffering Scatter DMA (Writes)

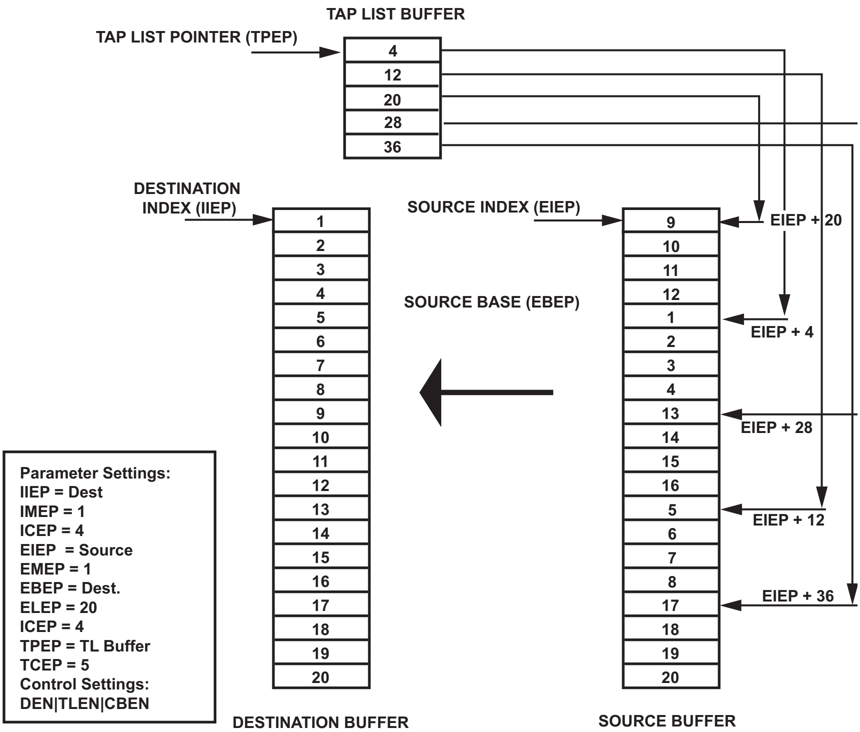

Figure 4-31. Circular Gather DMA (Reads)

Delay Line DMA

Delay line DMA is used to support reads and writes to external delay line buffers with limited core interaction. In this sense, delay line DMA is basically a quantity of integrated writes followed by reads from external memory-called a *delay line DMA access*. Delay line DMA is described in the following sections.

Delay line DMA can only operate by using chained DMA mode (`CHEN` bit set). In order to use delay line DMA for a single DMA sequence, initialize the `CPEP` register to zero in the TCB.

Delay Line DMA operates using the following five steps:

1. Load first half TCB for write (7 parameters).
2. DMA writes to the delay line buffer until IC = 0.
3. Update EI index pointer if circular mode is enabled.
4. Load second half TCB for read (6 parameters).
5. DMA tap based read from delay line buffer until RC = 0.

   Jump to step 1.

Writes to delay line, Figure 4-32. The number of writes is determined by the ICEP register. The data is fetched from the IIEP register and the IMEP register is used as the internal modifier. The EIEP register serves as the external index and is incremented by the EMEP register after each write. These writes are circular buffered if circular buffering is enabled.

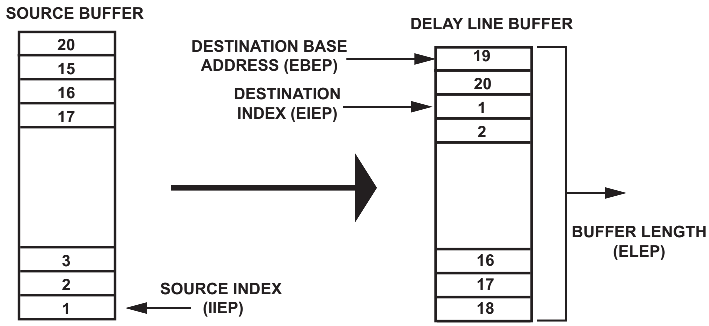

Figure 4-32. Write to Delay Line Buffer

When the writes are complete, (ICEP = 0) the EIEP register, which serves as the write pointer of the delay line, is written back (WRBEN is hardwired to 1) to the internal memory TCB location from where it was fetched.

Reads from the delay line, Figure 4-33. For reads, the tap list (TL) modifiers are used and the number of reads is determined by the RCEP register. The write pointer in the RIEP register serves as the index address for these reads (reads start from where writes end). The RIEP register, along with tap list modifiers, are used in a pre-modify addressing mode to create the external address for the reads. Therefore, for each read, the DMA controller fetches the external modifier (TCEP register) from the tap list and the reads are circular buffered (if enabled). Therefore, for each read, the DMA controller fetches the external modifier from the tap list and the reads are circular buffered (if enabled).

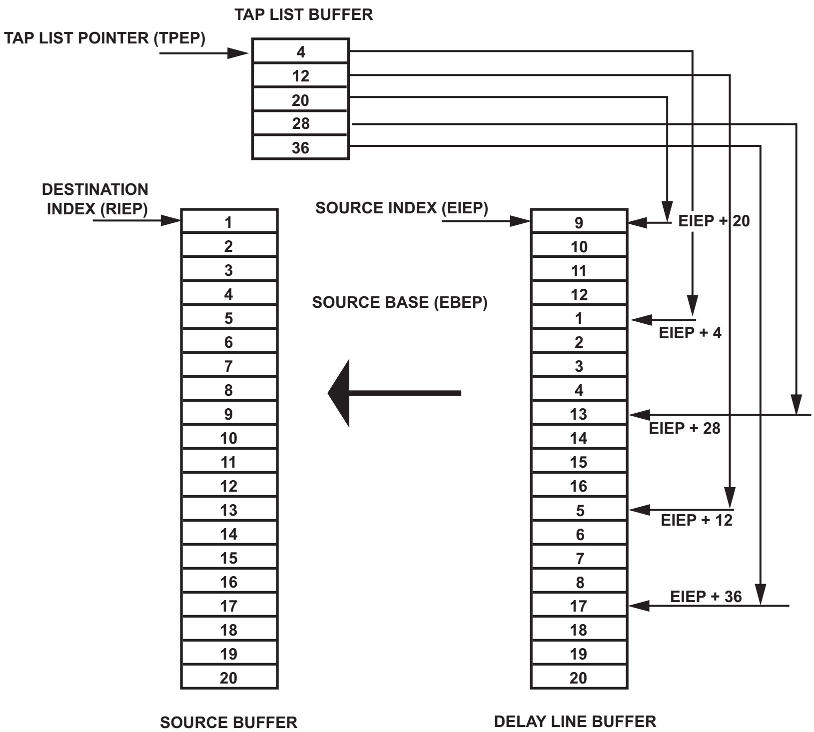

Figure 4-33. Read From Delay Line Buffer

Pre-Modified Read Index

Note that TL[N] is the first tap list entry in internal memory pointed to by the tap list pointer register (TPEP). Tap list entries are 27-bit signed integers. Therefore, for each read-block, the DMA state machine fetches the offset external modifier from the tap list. The reads are circular buffered if circular buffering is enabled.

The external address generation follows pre-modify addressing for reads in delay line DMA and therefore the EIEP register values are not updated. Also the EMEP register does not have any effect during these delay line reads. Once the read count completes, the ICEP register decrements to zero (both ICEP and TCEP are zero) for the final tap. Finally, the delay line DMA access completes and the DMA completion interrupt is generated. If chaining is enabled, the interrupt is dependent on the PCI bit setting. The delay line DMA can only be initialized using the TCB. In order to use the delay line DMA for a single DMA sequence, initialize the CPEP register to zero in the TCB.

For each 32-bit tap read, the external read index is shown in Table 4-29. Note that one tap list entry starts multiple reads.

Table 4-29. Read/Write Index Pre-Modify (Scatter/Gather DMA)

Pre-Modify Address Equation	Result	
RIEP + TPEP[TCEP] + (RMEPxRCEP)	**Blocksize**	**Tap**
RIEP + TPEP[0] + RMEPx1	N	0
RIEP + TPEP[0] + RMEPx2		
RIEP + TPEP[0] + RMEPx3		
RIEP + TPEP[0] + RMEPxN		

Table 4-29. Read/Write Index Pre-Modify (Scatter/Gather DMA) (Cont'd)

Pre-Modify Address Equation	Result	
RIEP + TPEP[TCEP] + (RMEPxRCEP)	Blocksize	Tap
RIEP + TPEP[1] + RMEPx1	N	1
RIEP + TPEP[1] + RMEPx2		
RIEP + TPEP[1] + RMEPx3		
RIEP + TPEP[1] + RMEPxN		
RIEP + TPEP[M] + RMEPx1	N	M
RIEP + TPEP[M] + RMEPx2		
RIEP + TPEP[M] + RMEPx3		
RIEP + TPEP[M] + RMEPxN		

External Port DMA Group Priority

The external port has two DMA channels. When the channels have data ready, the channel arbitrates by a fixed or rotating method (which is the first arbitration stage). The winning channel requests the DMA bus arbiter to get control of the external port DMA bus (2nd stage of arbitration). In fixed priority, channel 0 has highest priority.

For fixed priority, if channel 0 performs internal to internal memory transfers, then channel 1 has the higher priority.

For the I/O processor, the two DMA channels are considered as a group with one arbitration request. For more information, see "External Port DMA Arbitration" on page 3-39.

Interrupts

There are two external port DMA channels. The following sections describe the two ways of triggering interrupts. Table 4-30 provides an overview of external port interrupts.

Table 4-30. External Port Interrupt Overview

Default Programmable Interrupt	Sources	Masking	Service
EPDMA0I = P9I EPD-MA1I = P13I	DMA complete Internal transfer completion Access completion	N/A	RTI instruction

Sources

Each external port DMA module generates one interrupt signal. The external port DMA can generate interrupts under the conditions described in the following sections.

Delay Line DMA

For the delay line DMA, the DMA complete interrupt is generated when the delay line reads are completed (after the write access).

Scatter Gather DMA

With scatter/gather DMA, the DMA complete interrupt is generated only after all tap list reads/writes are complete.

Internal Transfer Completion

This mode of interrupt generation is enabled when the INTIRT bit is set in the DMA control register and resembles traditional SHARC DMA interrupt generation. This mode is provided for backward compatibility. This interrupt is generated once the DMA internal transfers (transmit or receive) are completed. For external transmit DMA, there may be still external access pending at the external DMA interface when the completion interrupt is generated. Therefore, the DMA may be disabled on the DMA complete interrupt only if the external interface is idle (for example, EXTS = 0).

Access Completion

This is the default mode of interrupt generation where the DMA complete interrupt is generated when accesses are completed.

- For external write DMA, the DMA complete interrupt is generated only after external writes on the DMA external interface are done.
- For external read DMA, the DMA complete interrupt is generated when the internal DMA writes complete.

In this mode, the DMA interface can be disabled as soon as the interrupt is received, (there is no need to check the EXTS bit before disabling the DMA interface).

ⓘ The DMA interface can be disabled based on a DMA complete interrupt. However, the external device interfaces—AMI/SDRAM/DDR2 may still be performing writes of the DMA data. Prior to disabling any of these devices, programs should check their respective status bits.

ⓘ If DMA is disabled in the middle of data transfers, the DMA interrupts should not be used.

Chained DMA

For chained DMA, if the PCI bit is cleared (= 0), the DMA complete interrupt is generated only after the entire chained DMA access is complete. If the PCI bit is set (= 1), then a DMA interrupt is generated for each TCB.

In a chained delay line DMA, the PCI bit determines if each delay line TCB generates an interrupt or not. For scatter/gather DMA, the PCI bit setting determines if each tap list TCB generates an interrupt in a chained access.

Masking

The EPDMA0I and EPDMA1I signals are routed by default to programmable interrupts as follows.

- To service the EPDMA0I, unmask (set = 1) the P9IMSK bit in the LIRPTL register.
- To service the secondary EPDMA1, unmask (set = 1) the P13IMSK bit in the LIRPTL register.

For example:

```
bit set LIRPTL P9IMSK;    /* unmasks P9I interrupt */
bit set LIRPTL P13IMSK;    /* unmasks P13I interrupt */
```

Service

Interrupts are serviced with a RTI (return from interrupt) instruction.

Interrupt Dependency on DMA Mode

Interrupt generation varies, depending on the DMA mode used. The INTIRT bit determines whether the interrupt is generated on internal completion or access completion. The following also effect interrupt generation.

- For standard chained DMA, if the PCI bit is cleared (= 0), the DMA complete interrupt is generated only after the entire chained DMA access is complete. If the PCI bit is set (= 1), then a DMA interrupt is generated for each TCB.

- For the delay line DMA, the DMA complete interrupt is generated when both the write access and the delay line reads are completed. In a chained delay line DMA, the `PCI` bit determines if each delay line TCB generates an interrupt or not.
- With scatter/gather DMA, the DMA complete interrupt is generated only after all tap list reads/writes are complete. As in the delay line DMA, the `PCI` bit setting determines if each tap list TCB generates an interrupt in a chained access.

If DMA is disabled in the middle of data transfers, the DMA interrupts should not be used.

External Port Throughput

The following sections provide information on the throughput of the external port interfaces (AMI, SDRAM).

Data Throughput

Table 4-31 provides information needed to configure the SDRAM interface for the desired throughput.

Table 4-31. SDRAM 16-bit SISD Data Throughput

Access	Page	Throughput per SDCLK (16-Bit Data)
Sequential uninterrupted reads	Same	One word per two cycles[1]
Any writes	Same	One word per two cycles
Non sequential uninterrupted reads	Same	One word per seven cycles (CL=2) One word per eight cycles (CL=3)

1 Read Optimization enabled, first data of a sequential read takes 7 cycles for CL =2 and 8 cycles for CL = 3, thereafter it is one word per two cycles.

Table 4-32 provides information needed to configure the DDR2 interface for the desired throughput.

Table 4-32. DDR2 16-bit SISD Data Throughput

Access	Page	Throughput per DDR2CLK (16-Bit Data)
Sequential uninterrupted reads	Same	One word per two cycles[1]
Any writes	Same	One word per cycle
Non sequential uninterrupted reads	Same	One word per 10 cycles (CL=3)
Non sequential interrupted writes	Same	One word per two cycles

1 Read Optimization enabled, first data of a sequential read takes 10 cycles for CL = 3, thereafter it is one word per cycle.

The AMI data throughput is shown in Table 4-33.

Table 4-33. AMI Read/Write Throughput

Access[1]	8-Bit I/O	16-Bit I/O
Write	32-bit word per 12 cycles	32-bit word per 6 cycles
Read	32-bit word per 12 cycles	32-bit word per 6 cycles

1 Throughput for minimum wait states of 2 with no idle and hold cycles.

DMA Throughput

Table 4-34 provides approximate throughput information with the processor core running at 400 MHz for DMA-driven reads and writes of external DDR2 memory. The throughput numbers shown are measured by running chained DMA with four TCBs (with 256 32-bit words per transfer block).

For the analysis, 16 bit DDR2 is used (t_{FAW}=10, t_{RRD}=2, t_{RTP}=2, t_{RCD}=3, t_{WTR}=1, t_{RP}=3, t_{RAS}=8, CL=4, AL=4, t_{WR}=4).

Throughput is calculated by measuring time between the instant when DMA is enabled and instant when DMA completion ISR is entered.

Table 4-34. DMA Throughput, 400 MHz Core Clock

Operation	DDR2 Clock	Clock Ratio	Throughput
DMA Reads			
	133 MHz	1:3	473M bytes/sec.
	200 MHz	1:2	700M bytes/sec.
DMA Writes			
	133 MHz	1:3	481M bytes/sec.
	200 MHz	1:2	732M bytes/sec.

Core Throughput

Table 4-35 provides approximate throughput information with the processor core running at 400 MHz for core-driven reads and writes of external DDR2 memory. The throughput numbers shown are measured by running a loop of 1024 read/writes (512 in case of SIMD reads/writes).

For the analysis, 16-bit DDR2 is used (t_{FAW}=10, t_{RRD}=2, t_{RTP}=2, t_{RCD}=3, t_{WTR}=1, t_{RP}=3, t_{RAS}=8, CL=4, AL=4, t_{WR}=4).

Throughput is calculated from start of the first iteration of the loop to the end of the last iteration of the loop.

Table 4-35. Core Throughput, 400 MHz Core Clock

Operation	DDR2 Clock	Clock Ratio	Throughput
Core Reads (SISD/SIMD)			
	133 MHz	1:3	495M bytes/sec.
	200 MHz	1:2	742M bytes/sec.
Core Writes (SISD)			
	133 MHz	1:3	529M bytes/sec.
	200 MHz	1:2	793M bytes/sec.
Core Writes (SIMD)			
	133 MHz	1:3	531M bytes/sec.
	200 MHz	1:2	796M bytes/sec.

DDR2 Read Optimization

Listing 4-8 through Listing 4-13 on page 4-152 provide different scenarios of core read accesses. The timing settings for these examples are: (CL = 4, AL = 0, t_{RRD} = 3, t_{RTP}= 2, t_{RCD} = 4, t_{RP} = 4, t_{RAS} = 9)

Listing 4-8 shows how read optimization can be used efficiently using core accesses. All reads are on the same page and it takes 1070 DDR2CLK cycles to perform 1024 reads which is close to 1 DDR2CLK cycle per access.

Listing 4-8. Consecutive Locations Accessed Sequentially With Read Optimization Enabled

```
ustat1=dm(DDR2CTL0);
bit set ustat1 DDR2OPT|DDR2MODIFY1;
dm(DDR2CTL0)=ustat1;
nop;
I8 = intmem_addr;
M8 = 1;
I0 = sdram_addr;
M0 = 1;
```

```
lcntr = 1024, do(PC,1) until lce;
R0 = dm(I0,M0),pm(I8,M8) = R0;
```

In Listing 4-9 the access are made non-sequential by inserting a NOP instruction in between reads. All reads are on the same page and it takes approximately 11794 DDR2CLK cycles to perform 1024 reads. That is approximately 11 DDR2CLK cycles per read. Even though optimization is enabled it has no effect because of the non-sequential behavior of the read accesses.

Listing 4-9. Consecutive Locations Accessed Non-sequentially With Read Optimization Enabled

```
ustat1=dm(DDR2CTL0);
bit set ustat1 DDR2OPT|DDR2MODIFY1;
dm(DDR2CTL0)=ustat1;
nop;
I8 = intmem_addr;
M8 = 1;
I0 = sdram_addr;
M0 = 1;
lcntr = 1024, do(PC,2) until lce;
   R0 = dm(I0,M0),pm(I8,M8) = R0;
   Nop;
```

In Listing 4-10 the access are made from non-consecutive locations. All reads are on the same page and it takes approximately 11269 DDR2CLK cycles to perform 1024 reads which is approximately 11 DDR2 cycles per read. Even though optimization is enabled it has no effect because the locations accessed are non-consecutive. Set the DDR2MODIFY bit to match the DAG Modifier (2 in this case) to get better performance.

Listing 4-10. Non-Consecutive Locations Accessed Sequentially With Read Optimization Enabled

```
ustat1=dm(DDR2CTL0);
bit set ustat1 DDR2OPT|DDR2MODIFY1;
dm(DDR2CTL0)=ustat1;
nop;
I8 = intmem_addr;
M8 = 1;
I0 = sdram_addr;
M0 = 2;
lcntr = 1024, do(PC,1) until lce;
   R0 = dm(I0,M0),pm(I8,M8) = R0;
```

In Listing 4-11, all reads are on the same page and it takes 5655 DDR2CLK cycles to perform 1024 reads.

Listing 4-11. Consecutive Locations Accessed Sequentially With Read Optimization Disabled

```
ustat1=dm(DDR2CTL0);
bit clr ustat1 DDR2OPT|DDR2MODIFY1;
dm(DDR2CTL0)=ustat1;
nop;
I8 = intmem_addr;
M8 = 1;
I0 = sdram_addr;
M0 = 1;
lcntr = 1024, do(PC,1) until lce;
   R0 = dm(I0,M0),pm(I8,M8) = R0;
```

In Listing 4-12 the access are made non-sequential by inserting a NOP in between accesses. All reads are on the same page and it takes around 11272 DDR2CLK cycles to perform 1024 reads which is approximately 11 DDR2CLK cycles per read.

Listing 4-12. Consecutive Locations Accessed Non-sequentially With Read Optimization Disabled

```
ustat1=dm(DDR2CTL0);
bit clr ustat1 DDR2OPT|DDR2MODIFY1;
dm(DDR2CTL0)=ustat1;
nop;
I8 = intmem_addr;
M8 = 1;
I0 = sdram_addr;
M0 = 1;
lcntr = 1024, do(PC,2) until lce;
   R0 = dm(I0,M0),pm(I8,M8) = R0;
   Nop;
```

In Listing 4-13 the access are made from non-consecutive locations. All reads are on the same page and it takes around 10249 `DDR2CLK` cycles to perform 1024 reads. That is approximately 10 `DDR2CLK` cycles per read. Please note that compared to read optimization enabled case throughput is better by 1 `DDR2CLK` cycle per access here.

Listing 4-13. Non-Consecutive Locations Accessed Sequentially With Read Optimization Disabled

```
ustat1=dm(DDR2CTL0);
bit clr ustat1 DDR2OPT|DDR2MODIFY1;
dm(DDR2CTL0)=ustat1;
nop;
I8 = intmem_addr;
M8 = 1;
I0 = sdram_addr;
M0 = 2;
lcntr = 1024, do(PC,1) until lce;
   R0 = dm(I0,M0),pm(I8,M8) = R0;
```

In summary, for SISD mode the modifier of 1 allows programs to take advantage of sequential addressing. One burst reads two words until a new burst has started resulting in 1 cycle/word with optimization enabled. For SIMD mode the modifier of 2 fills one entire burst (explicit 2 words + implicit 2 words) also performing at 1 cycle/word if optimization is enabled.

Throughput Conditional Instructions

A conditional read/write may take 1 PCLK cycle (access made and access aborted, respectively). For more information, see "External Memory Access Restrictions" on page 4-170.

External Instruction Fetch Throughput

Read optimization logic does not apply to external instruction fetch.

SDRAM Throughput

Table 4-36 illustrates the performance of code execution depending on different access types for SDRAM.

Table 4-36. SDRAM 16-bit Instruction Fetch Throughput

Access	Page	Throughput per SDCLK
Sequential uninterrupted reads	Same	2 instructions per 6 cycles (CL=3)
Non sequential uninterrupted reads	Same	1 instruction per 9 cycles (CL=2) 1 instruction per 10 cycles (CL=3)
The SDC has to fetch 3 instruction data for each ISA instruction. First 48-bit instruction of a sequential read will take 8 cycles for CL = 2 and 9 cycles for CL = 3, thereafter it is two instructions per 6 cycles. The instruction available cycles will look like - 8, 10, 14, 16, 20, 22, 26, 28 … (CL = 2)		

DDR2 Throughput

Table 4-37 illustrates the performance of code execution depending on different access types for DDR2.

Table 4-37. DDR2 16-bit Instruction Fetch Throughput

Access	Page	Throughput per DDR2CLK (CL = 3)
Sequential uninterrupted reads	Same	2 instructions per 3 cycles
Non sequential uninterrupted reads	Same	1 instructions per 11 cycles
The DDR2C has to fetch 3 instruction data for each ISA instruction. **First 48-bit instruction of a sequential read will take 11 cycles for CL = 3, thereafter it is two instructions per 3 cycles.** **The instruction available cycles will look like - 11, 12, 14, 15, 17, 18, 20, 21 … (CL = 3)**		

AMI Throughput

When executing from external asynchronous memory, instruction throughput depends on the settings of asynchronous memory such as the number of wait states, the ratio of core to peripheral clock and other settings. For details, please refer to the external port global control register (EPCTL), the AMICTLx register, and the SDCTL0 register in "External Port Registers" on page A-20.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific). After the AMI/SDRAM/DDR2 registers are configured the write effect latency is 1.5 PCLK cycles minimum and 2 PCLK cycles maximum.

After the external port register is configured the effect latency is 4 PCLK cycles. This is the valid for the worst case of core to SDRAM/DDR2 clock ratio of 1:4

Programming Models

The following sections provide information on the various programming models that are used through the external port interface.

For all external port programming models two cases of latency are involved.

1. The latency for an external port clock ratio change is 8 PCLK cycles. After any external port clock ratio change (PMCTL register) no external port registers (external port, AMI, SDC, DDR2, external port DMA registers) should be changed during these 8 PCLK cycles. Also no external memory accesses (AMI, SDRAM or DDR2) are allowed during this period.
2. The access latency for external port registers is 4 PCLK cycles. After any external port register change (external port, AMI, SDC, DDR2, external port DMA registers) external memory accesses are not allowed during this period.

AMI Initialization

After reset, the SDCLK/DDR2CLK is running with the default PLL settings. However, the AMI must be configured and initialized. In order to set up the AMI, use the following procedure. Note that the registers must be programmed in order.

1. Chose a valid CCLK to SDCLK/DDR2CLK clock ratio in the PMCTL register.
2. Wait at least 15 CCLK cycles (effect latency).
3. Assign external banks to the AMI using the EPCTL register (default).

4. Enable the global AMIEN bit and program the AMI control (AMICTLx) registers. (Define control settings for AMI based on SDCLK speed and asynchronous memory specifications.

   The AMIMS and AMIS bits 1–0 of the AMI status register (AMISTAT) can be checked to determine the current state of the AMI.

5. Wait 8 core cycles before first data access (effect latency).

AMI Instruction Fetch

For ISA instruction fetch, these steps are required (besides power-up).

1. Assign external bank 0 to AMI in the EPCTL register.
2. Enable the global AMIEN bit and clear (=0) the PKDIS bit.
3. For ISA instruction the first fetch starts at logical address 0x20 0000.

SDRAM Controller

This section describes software programming steps required for the successful operation of the SDRAM controller.

Power-Up Sequence

After reset, the SDCLK is running with the default PLL settings. However, the controller must be configured and initialized. In order to set up the controller and start the SDRAM power-up sequence for the SDRAMs, use the following procedure. Note that the registers must be programmed in order.

1. Chose a valid CCLK to SDCLK clock ratio in the PMCTL register.
2. Wait at least 15 core clock cycles until the new SDCLK frequency has been settled up correctly.

3. Assign external banks to controller in the EPCTL register.
4. Wait at least 8 core clock cycles (effect latency).
5. Program the refresh counter in the SDRRC register.
6. Define global control for controller and SDRAM based on speed and SDRAM specifications in the SDCTL register.
7. Wait at least 8 core clock cycles (effect latency).
8. Once the SDPSS bit in the SDCTL register is set to 1, a dummy access is required to start the power-up sequence.

The SDRAM is ready for access.

The SDRS bit of the SDRAM control status register can be checked to determine the current state of the controller. If this bit is set, the SDRAM power-up sequence has not been initiated.

Changing the SDRAM Clock on the Fly

Self-refresh mode is an option, this mode can take infinite time. Use the following steps.

1. Ensure that the SDRAM controller is idle by checking the SDCI bit (bit 0) in the SDSTAT0 register.
2. Set the self refresh mode bit (SDSRF bit) and wait until the SREF status bit is set.
3. Shut off the clock to the external port using the EPOFF bit in the PMCTL1 register.
4. Change the clock ratio/frequency and wait 15 CCLK cycles.

5. Enable the clock to the external port.
6. Exit from by SDRAM dummy access, and wait until the SREF status bit is cleared.

The SDRAM controller is now ready for operation with the new clock frequency. If timing parameters require a change due to the frequency change that can be done after step 6.

If any mode register parameter requires a change a force mode register write must be performed with the new values.

SDRAM Instruction Fetch

Use the following steps to perform an ISA/VISA instruction fetch (exclusive of power up).

1. Assign external bank 0 to DDR2 using the EPCTL register.
2. Configure the power-up sequence.

For ISA instruction the first fetch starts at logical address 0x20 0000 and for VISA instruction fetch at address 0x60 0000.

Output Clock Generator Programming Model

The following non VCO programming sequence may be used to change the output generator clock and the core-to-peripheral clock ratio (for example the SDRAM clock). Note that if your program is only changing the PLL output divider, programs do not need to wait 4096 CLKIN cycles (required only if the PLL multiplier or the INDIV bit is modified).

1. Disable the peripheral (SDRAM). Note that the peripherals cannot be enabled when changing clock ratio.
2. Select the PLL divider by setting the PLLDx bits (bits 6–7 in the PMCTL register).

3. Select the clock divider (CCLK to SDRAM ratio) by setting the ratio bits (PMCTL register).
4. Wait 15 CCLK cycles. During this time, programs must not execute any valid instructions.
5. Enable the peripheral (SDRAM).

   The new divisor ratios are picked up on the fly and the clocks smoothly transition to their new values after a maximum of 15 core clock CCLK cycles.

Self-Refresh Mode

The following steps are required when entering and releasing self-refresh mode.

1. Set the SDSRF bit to enter self-refresh mode.
2. Poll the SDSRA bit in the SDRAM status register (SDSTAT) to determine if the SDRAM has already entered self-refresh mode.
3. Set the DSDCTL bit to freeze SDCLK (optional).
4. Self refresh mode-no activities on all SDRAM signals (clock optional).
5. Clear the DSDCTL bit to re-enable SDCLK (optional).
6. SDRAM access releases controller from self-refresh mode.

Changing the VCO Clock During Runtime

In previous SHARC models, only a hardware reset initiated another SDRAM power-up sequence. This is no longer the case since the PLL allows programs to change the output clocks during runtime.

All SDRAM timing specifications are normalized to the SDRAM clock. Since most of these are minimum specifications, (except t_{REF}, which is a maximum specification), a variation of the system clock violates a specific specification and causes a performance degradation for the other specifications.

The reduction of the system clock violates the minimum specifications, while increasing the system clock violates the maximum t_{REF} specification. Therefore, careful software control is required to adapt these changes. Therefore, the release from self-refresh mode should be a dummy read operation since it happens with the old frequency settings.

For most applications, the SDRAM power-up sequence and writing of the mode register needs to occur only once. Once the power-up sequence has completed, the SDPSS bit should not be set again unless a change to the mode register is desired.

The recommended procedure for changing the system frequency SDCLK is as follows.

1. Set the SDRAM to self-refresh mode by writing a 1 to the SDSRF bit of the SDCTL register.
2. Poll the SDSRA bit of SDSTAT register for self-refresh grant.
3. Execute the desired PLL programming sequence. (For more information, see “PLL Start-Up” on page 23-9.)
4. Wait 4096 CLKIN cycles ($\overline{\text{RESETOUT}}$ asserted) which indicates the PLL has settled to the new frequency.
5. Reprogram the SDRAM registers (SDRRC, SDCTL) with values appropriate to the new SDCLK frequency and assure that the SDPSS bit is set.

6. Bring the SDRAM out of self-refresh mode by performing a dummy read SDRAM access.
7. The controller now issues the commands PREA, 8xREF and MRS to initialize the controller and the SDRAM to the new frequency.

The SDRAM device is now ready to be accessed.

DDR2 Controller

The following sections are specific to DDR2 SDRAM memory on the ADSP-2146x processor. Note these general rules.

- If a program changes only the timing parameters (t_{RRD}, t_{RP} for example) without changing the clock ratios, then no initialization sequence is required. However, if any of the mode registers are changed (MR, EMR1, EMR2), then the program needs to perform a force load of the corresponding mode register using the force bits in the DDR2CTL0 register. Note the CAS latency (CL) is a timing parameter which is also transferred to the memory via the mode register command.
- However if a program changes the clock frequencies, then the program also needs to reset the DLL. Both the ADSP-2146x and DDR2 memory DLL will have to be reset. In such a case, an entire initialization sequence is required.
- With the worst case timing parameter, it takes 410 cycles for one external DDR2 bank to calibrate.

Power-Up Sequence

The following steps are used to power-up the DDR2 device.

1. Program the core to DDR2 clock ratio using the PMCTL register. For PLL changes wait at least 4096 core cycles, for output divider changes at least 15 core clock cycles for effect latency. Ensure the minimum DDR2 clock frequency is stable and at least 125 MHz (according to datasheet).
2. Wait at least 200 µs with a stable clock provided to the DDR2 memory (JEDEC standard).
3. If a new DDR2 frequency is desired, put the on-chip DLL into reset using DLL1-0CTL1 registers.
4. Wait at least 9 core cycles.
5. DLL in reset starts new locking event. Wait for the DLL to lock to the new frequency. Note that the DLL locking time depends on the CCLK to DDR2_CLK ratio and is:
   - 1:2 – 3000 CCLK cycles
   - 1:3 – 7500 CCLK cycles
   - 1:4 – 10000 CCLK cycles
6. Assign the required external DDR2 banks in the EPCTL register.
7. Wait 8 core cycles for effect latency.
8. Program the refresh rate control register (DDR2RRC).
9. Program the timing parameters in the DDR2CTL1 register.
10. Program all MR and EMR3-1 settings in the DDR2CTL5-3 registers.
11. Ensure that the DDR2_DLL_DIS bit (DDR2CTL3) and the SH_DLL_DIS (DDR2CTL0) bits are cleared.
12. Enable DDR size (row, column, bank) and other parameters in the DDR2CTL0 register.

13. Wait 8 core cycles for effect latency.
14. Start the power-up sequence with the DDR2PSS bit. Wait for DLL external bank calibration.

The device now ready for any access.

Changing the DDR2 Clock on the Fly

Two different modes allow programs to change the DDR2 clock during run time.

Precharge power-down mode requires careful software control since the DRAM is no longer refreshed and therefore a maximum window must be guaranteed. This interval is typically $t_{RAS}max = 8 \times t_{REFI}$ or $9 \times t_{REFI}$). On die termination must be turned off.

Self-refresh mode is the 2nd option, and this mode can take infinite time.

Changing the Clock Frequency During Precharge Power Down Mode

Use the following procedure to change the clock frequency during precharge power down mode.

1. Ensure that the DDR2 controller is idle by checking the DDR2CI bit (bit 0) in the DDR2STAT0 register.
2. Perform a Force precharge all banks command (FPC bit) and force 8 refresh commands (FARF bit)
3. Set the precharge power down mode (DIS_DDR2CKE bit) and wait until the power-down status bit is set.
4. Shut off the clock to the external port in the PMCTL1 register.
5. Change clock ratio/frequency and wait 15 CCLK cycles.
6. Enable the clock to the external port.

7. Wait for the DLL to lock to the new frequency.

8. Exit from power-down mode by clearing the DIS_DDR2CKE bit, and wait until the DDR2PD status bit is cleared.

9. Perform an on chip DLL calibration again by setting the Force DLL calibration bit.

The DDR2 controller is now ready for operation with the new clock frequency. If timing parameters require a change due to the frequency change that can be done after step 7.

If any mode register parameter requires a change a force mode register write must be performed with the new values.

Note that the maximum precharge power down time is $9 \times t_{REFI}$.

Changing the Clock Frequency During Self-Refresh Mode

Use the following procedure to change the clock frequency during self-refresh mode.

1. Ensure that the DDR2 controller is idle by checking the DDR2CI bit (bit 0) in the DDR2STAT0 register.

2. Set the self refresh mode bit (DDR2SRF) and wait until the SREF status bit is set.

3. Shut off the clock to the external port using the EPOFF bit in the PMCTL1 register.

4. Change the clock ratio/frequency and wait 15 CCLK cycles.

5. Enable the clock to the external port.

6. Wait for the DLLs to lock to the new frequency.

7. Exit from self-refresh mode by clearing the DDR2SRF bit, and wait until the SREF status bit is cleared.
8. Perform an on chip DLL calibration again by setting the Force DLL calibration bit.

External Port DMA

The following sections describe the programming steps for different types of DMA transfers. Before using the external port DMA it is assumed that the AMI/SDRAM or DDR2 controllers are programmed accordingly.

Standard DMA

Use the following procedure to set up and run a standard DMA on the external port.

1. Configure the AMICTLx registers to enable the AMI and to set the desired wait states, data bus width, and so on. Configure the SDCTL registers to enable SDRAM/DDR2, and to set the desired clock and timing settings, the data bus width, and other parameters.
2. Initialize the IIEP, IMEP, ICEP, EIEP, and EMEP registers.
3. If circular buffering is desired, use the corresponding TCB storage.
4. If scatter/gather DMA is desired, program additional writes to the TCEP and TPEP registers.
5. Enable DMA using the DMAEN bit, and set the transfer direction using the TRAN bit in the DMACx registers. If scatter/gather DMA is desired, set the TLEN bit. It is advised that the DMA FIFOs are flushed using the DFLSH bit when DMA is enabled.

Once the DMA control register is initialized, the DMA engine fetches the DMA descriptors from the address pointed to by CPEP. Once the DMA descriptors are fetched then the DMA (or the tap list DMA) process starts.

Once the DMA (or tap list DMA) is complete, the new DMA descriptors are loaded and the process is repeated until CPEP = 0x0. A DMA completion interrupt is generated at the end of each DMA block or at the end of entire chained DMA, depending on the PCI bit setting.

Chained DMA

Use the following procedure to set up and run a chained DMA on the external port.

1. Clear the chain pointer register.
2. Configure the AMICTLx registers to enable the AMI, set the desired wait states, the data bus width, and so on. Configure the SDCTL register to enable the SDRAM/DDR2, configure the desired clock and timing settings, data bus width, and other parameters.
3. Initialize the CPEP register and set the PCI bit if interrupts are required after the end of each DMA block. Set the CPDR bit if different DMA direction is required in conjunction with the OFCEN bit in the DMACx register.
4. If circular buffering is needed, use the corresponding TCB storage.
5. Enable DMA using the DMAEN, bit, set chaining using the CHEN bit. If circular buffering is required, set the CBEN bit in the DMACx registers. It is advised that programs flush the DMA FIFOs using the DFLSH bit when DMA is enabled.

Once the DMA control register is initialized, the DMA controller fetches the DMA descriptors from the address pointed to by the external port chain pointer register (CPEP).

Once the DMA descriptors are fetched, the normal DMA process starts. Upon completion, new DMA descriptors are loaded and the process is repeated until CPEP = 0x0. A DMA completion interrupt is generated at the end of each DMA block or at the end of an entire chained DMA, depending on the PCI bit setting.

Delay Line DMA

1. Configure the AMICTLx register with the desired wait states, enable AMI, data bus width and other parameters.
2. Initialize the CPEP register and set the PCI bit if interrupts are required after the end of each delay line DMA block.
3. Enable DMA (DMAEN), delay line DMA (DLEN), chaining (CHEN) if required in the DMACx register. Programs should flush the DMA FIFO (DFLSH) along with enabling the DMA. If circular buffering is required (which is normally the case) enable it by setting the CBEN bit.

Once the DMA control register is initialized the DMA engine fetches the DMA descriptors from the address pointed to by the CPEP register. Once the delay line DMA access is complete, the new DMA descriptors are loaded and the process is repeated until CPEP = 0x0. A DMA completion interrupt is generated at the end of each delay line DMA block or at the end of entire chained DMA, depending on the PCI bit setting.

When delay line DMA is enabled with chaining, all the chained DMA blocks follow the delay line DMA access procedure. It is not possible to mix normal DMA with delay line DMA in chained DMA.

Disabling and Re-enabling DMA

Use the following programming model to disable the external port DMA during transfers.

1. Clear the DMAEN bit on the DMACx register.
2. Wait until the EXTS bit is 0.
3. Write 0x0 to the ICEP and DMACx registers. In cases where DMA is used without chaining, writing to ICEP is not required.
4. Re-initialize the required DMA registers, and enable the DMACx register while flushing the data buffer. The external port DMA buffers are flushed by setting the respective DFLSH bits.

Additional Information

1. If DMA is disabled in the middle of a data transfer, then DMA interrupts cannot be relied on.
2. A standard DMA (no chaining) can be stopped midway by clearing the DMAEN bit in the DMACx register and then restarted from the point where it was stopped by re-enabling the DMAEN bit. This mode of inhibiting the DMA only works with standard DMA. If a chained/delay line DMA is disabled by clearing DMAEN bit then the DMA should be reprogrammed again following the above programming model.
3. For a chained DMA, new TCB loading can be inhibited by clearing the CHEN bit while keeping all other control bits the same. The new TCB is loaded once CHEN bit is re-enabled. The TCB load which was happening when CHEN was cleared will complete.
4. Before initializing a chained DMA (including delay line) make sure that the ICEP and ECEP registers are zero.

5. The DMA parameter registers (except DMACx) should not be written to while chaining is occurring (the CHS bit is set), but any register can be read during chaining.

6. A zero count for the ICEP, RCEP and TCEP registers is forbidden. If a chain pointer with such a descriptor is programmed then the DMA might hang. So a read count zero or a write count zero for a delay line DMA is also forbidden.

External Instruction Fetch

The section describes the software programming steps needed for the successful operation of external instruction fetch through the external port. Note only the additional steps for code execution are illustrated. For timing related settings refer to "Functional Description" on page 4-9.

AMI Configuration

For instruction fetch, the original (logical) address is multiplied by 3/2 and this address is translated depending on the bus width and PKDIS bit setting.

1. Assign external bank0 to AMI in the EPCTL register (default).

2. Wait at least 8 CCLK cycles (effect latency).

3. Enable the global AMIEN bit and clear (=0) the PKDIS bit.

SDRAM Configuration

For instruction fetch, the original (logical) address is multiplied by 3/2 and this address is translated depending on the bus width setting (X16DE bit).

1. Assign external bank 0 to SDRAM in the EPCTL register (default).
2. Wait at least 8 CCLK cycles (effect latency).
3. Configure the SDCTL and SDRRC registers accordingly.

DDR2 Instruction Fetch

Use the following steps to perform an ISA/VISA instruction fetch (exclusive of power up).

1. Assign external bank 0 to DDR2 using the EPCTL register.
2. Configure the power-up sequence.

For ISA instruction the first fetch starts at logical address 0x20 0000 and for VISA instruction fetch at address 0x60 0000.

External Memory Access Restrictions

The following external memory restrictions should be noted when writing programs.

1. The LW mnemonic is not applicable to external memory.
2. Conditional accesses to external memory should not be based on any of the FLAG pin status.
3. There is one cycle latency between a multiplier status change and an arithmetic loop abort. This extra cycle is a machine cycle and not the instruction cycle. Therefore, if there is a pipeline stall (due to external memory access etc.) then the latency does not apply.

4. A one cycle stall is generated whenever an instruction that contains a conditional external memory access is in the decode stage, where the evaluation of the condition is dependent on the outcome of the previous instruction in address stage. It applies to all kinds of conditions except for conditions based on FLAG status. The following is an example:

```
f12 = f11+f10;
if eq dm(ext) = r0;
```

5. The `FLUSH CACHE` instruction has an effect latency of one instruction when executing program instructions from internal memory, and two instructions when executing from external memory.

6. When a new external memory instruction fetch occurs on the processor due to a jump from internal to external memory, or after a cache hit while executing instructions from external memory, there is one stall cycle present in the fetch1 stage. This stall avoids resource conflicts at the cache interface.

7. Any sequence of external memory access (read or write) followed by an IOP access, causes the IOP access to fail. To workaround this restriction, separate the external memory access and IOP access by adding a NOP instruction or any other instruction which is not either an IOP read/write, or an external memory access. Example:

```
R12 = dm(Ext_mem);
NOP; /* fixes restriction */
R0 = dm(SPCTL2);
```

Debug Features

The following section describes the features available to aid in debugging the external port DMA module.

Core FIFO Write

The core may also write to the 6 deep data FIFO. When it does, the data word is pushed into the input side of the FIFO (as if it had come from the DMA on the channel). This can be useful for verifying the operation of the FIFO, the DMA channels, and the status portions of the external port DMA. The `DMACx` register returns the current state of the FIFO. Note that if both the DMA and core try to write to the FIFO, the core has higher priority.

5 LINK PORTS – ADSP-2146x

The ADSP-2146x processors have two 8-bit wide link ports, which can connect to another processor or peripheral link ports. The link ports allow a variety of interconnection schemes to I/O peripheral devices as well as co-processing and multiprocessing schemes. The port specifications are shown in Table 5-1.

Table 5-1. Link Port Specifications

Feature	**Link Port1–0**
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	No
SRU DAI Default Routing	N/A
SRU2 DPI Required	No
SRU2 DPI Default Routing	N/A
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	Yes
Transmission Simplex	Yes
Transmission Half Duplex	Yes
Transmission Full Duplex	No
Access Type	
Data Buffer	Yes

Table 5-1. Link Port Specifications (Cont'd)

Feature	Link Port1–0
Core Data Access	Yes
DMA Data Access	Yes
DMA Channels	2
DMA Chaining	Yes
Boot Capable	Yes (Link Port 0)
Local Memory	No
Max Clock Operation	LCLK

Features

These bidirectional ports have eight data lines, an acknowledge line, and a clock line. The maximum frequency of operation of the link ports is 166 MHz. The link port clock to core clock ratio programming is applicable only if the link port is configured as transmitter. The receiver link port can operate at any asynchronous clock frequency up to 166 MHz (or peripheral clock frequency (`PCLK = CCLK`/2) which ever is lower) independent of the programmed ratio.

The link ports contain the features shown in the following list.

- Operate independently and simultaneously.
- Pack data into 32-bit words; this data can be directly read by the processor or DMA-transferred to or from on-chip memory.
- Have double-buffered transmit and receive data registers.
- Include programmable clock and acknowledge controls for link port transfers. Each link port has its own dedicated DMA channel.

- Provide high-speed, point-to-point data transfers to other processors, allowing differing types of interconnections between multiple DSPs.

Pin Descriptions

The pins associated with each link port are described in the ADSP-2146x data sheet.

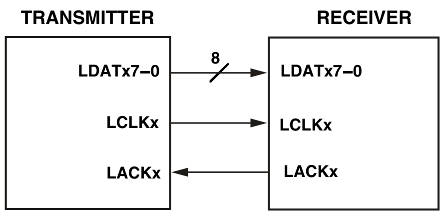

Figure 5-1. Link Port Pin Connections

Register Overview

Each link port has its own control and status register. These are described in the following sections and in "Link Port Registers" on page A-61. For information on the link port DMA registers, see "Standard DMA Parameter Registers" on page 3-4. For information on the link port buffer registers, see "Data Buffers" on page 3-10.

Control Registers (LCTLx). The control registers are used to enable the port, to set up DMA parameters, and to configure interrupts.

Status Registers (LSTATx). Programs can see several aspects of link port operation using the status registers. These include bus status, buffer status, receive and transmit status, and errors.

Clocking

The link port clock is derived from the clock out generator based on the link port to core clock ratio. For more information, see "Output Clock Generator" on page 23-5.

The link port to core clock ratios (1:2, 1:2.5, 1:3, 1:4) can be programmed in the PMCTL register. 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.

Functional Description

Each link port, shown in Figure 5-2, consists of eight data lines (LDATx7-0, x = 0, 1), a link port clock line (LCLKx), and a link port acknowledge line (LACKx). The LCLKx and LACKx pins of each link port allow handshaking for asynchronous data communication between DSPs. Other devices that follow the same protocol may also communicate with these link ports.

The link port operates in half-duplex mode, only receive or transmit operation can happen per link port by using core or DMA. If full-duplex operation is required both link ports must be used.

In receive operations, the data are received by the external receive buffer packed into 32-bit format and shifted to the internal receive buffer. The core or DMA read the data from the internal buffer. In transmit operations, the data are written to the internal transmit buffer and moved to the external transmit buffer to shift the data off-chip.The following sections provide details on this interface.

Architecture

Figure 5-2 shows the architecture of the link ports.

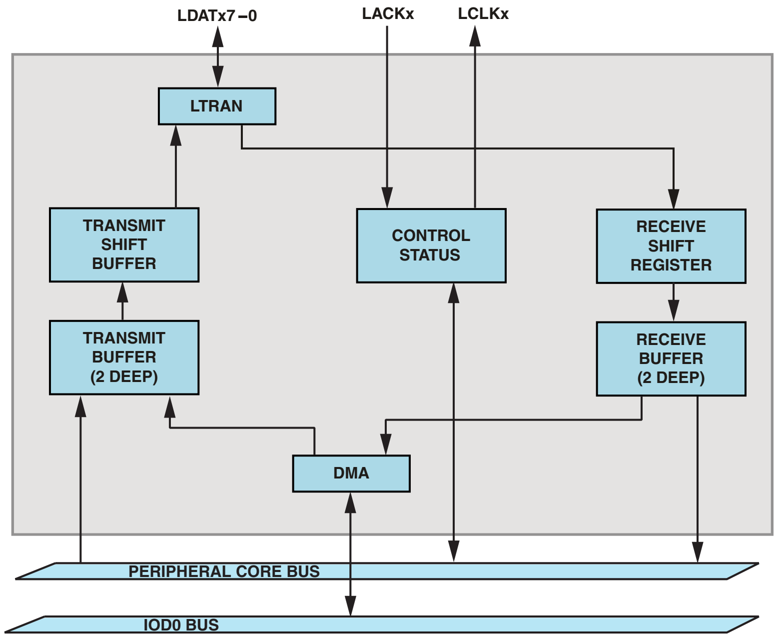

Figure 5-2. Link Port Block Diagram

Protocol

A link port transmitted word consists of 4 bytes (for a 32-bit word). The transmitter asserts the clock (LCLKx) high with each new byte of data. The falling edge of LCLKx is used by the receiver to latch the byte. The receiver asserts LACKx when it is ready to accept another word in the receive buffer, RXLBx. The transmitter samples LACKx driven by the receiver at the beginning of each word transmission (that is, after every 4 bytes with a positive level latch). If LACKx is deasserted at that time, the transmitter does not transmit the new word. The transmitter leaves LCLKx high and continues to drive the first byte if LACKx is deasserted. When LACKx is eventually asserted again, the transmitter drives LCLKx low and begins transmission of the next word. If the transmit buffer is empty, LCLKx remains low until the buffer is refilled, regardless of the state of LACKx.

The following list describes the stages during a link port handshake.

1. The LCLK signal stays high at byte 0 if LACK is sampled low on the previous LCLK falling edge. LCLK high indicates a stall.
2. The LxACK signal may deassert after byte 0.
3. The LACK signal reasserts as soon as the link buffer is not full.
4. The transmitter samples LACK to determine whether to transmit the next word.
5. The receiver accepts the remaining word even if LACK is deasserted. The transmitter does not send the following word.
6. Transmit data for next word is held until LACK is asserted.

The receive buffer may fill if a higher priority DMA, core I/O processor register access, or chain loading operation is occurring. The LACKx signal may deassert when it anticipates the buffer may fill. The LACKx signal is reasserted by the receiver as soon as the internal DMA grant signal has occurred, freeing a buffer location *or* the core reads the receive buffer RXLBx thereby freeing a buffer location. The LACKx signal inhibits transmission of the next word and not of the current byte.

Data is latched in the receive buffer on the falling edge of LCLKx. The receive operation is purely asynchronous and can occur at any frequency up to 166 MHz or peripheral clock frequency (whichever is less).

When a link port is not enabled, LDAT7-0, LCLKx and LACKx are three-stated. When a link port is enabled to transmit, the data pins are driven with whatever data is in the output buffer, LCLKx is driven high and LACKx is three-stated. When a port is enabled to receive, the data pins and LCLKx are three-stated and LACKx is driven high.

Intercommunication

The transmitter and the receiver may be enabled at different times. The LACKx and LCLKx signals should be held low with the external pull-down resistors. If the transmitter is enabled before the receiver, the LACKx signal (of the receiver) is held low and transmission is held off. If the receiver is enabled before the transmitter, the LCLKx signal (of the transmitter) is held low by the pull-down and the receiver is held off.

Unlike older SHARC processors that have link ports, the ADSP-2146x processors do not have an internal pull-down resistor. Because of this there is no PDRDE bit available to disable the internal pull-down and an external pull-down (20K Ohms) is required on the LACKx and LCLKx signals.

Figure 5-3, Figure 5-4 and Figure 5-5 show various timings for the link port.

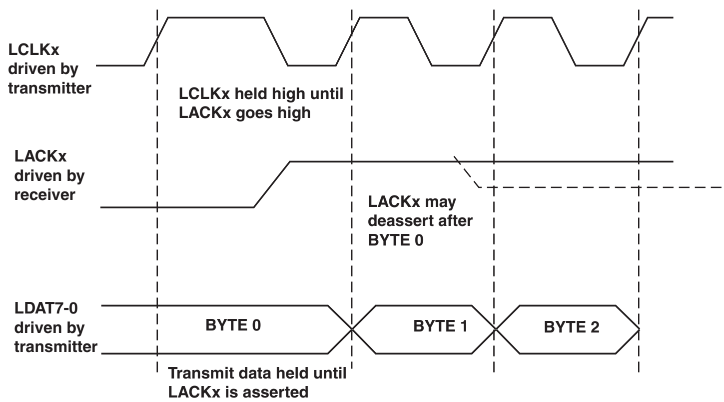

Figure 5-3. Enable Transmitter and Receiver at Different Times

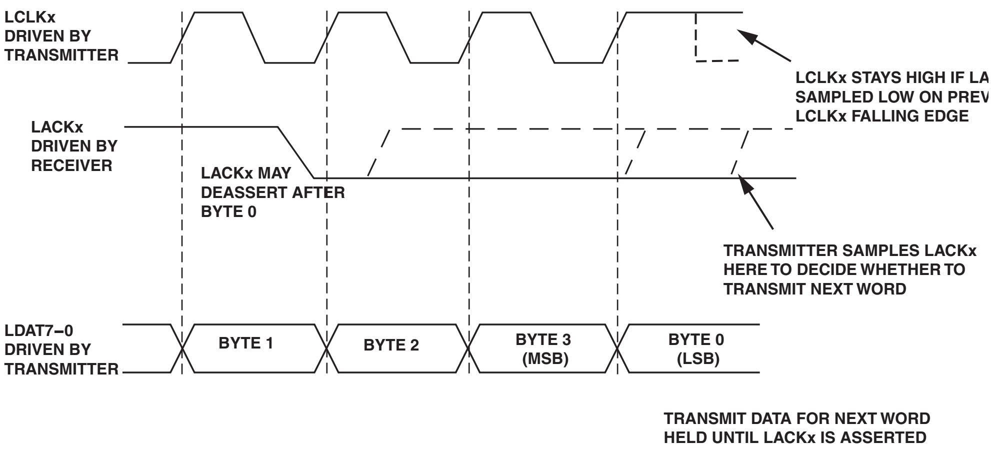

Figure 5-4. Relationship Between LACK and LCLK

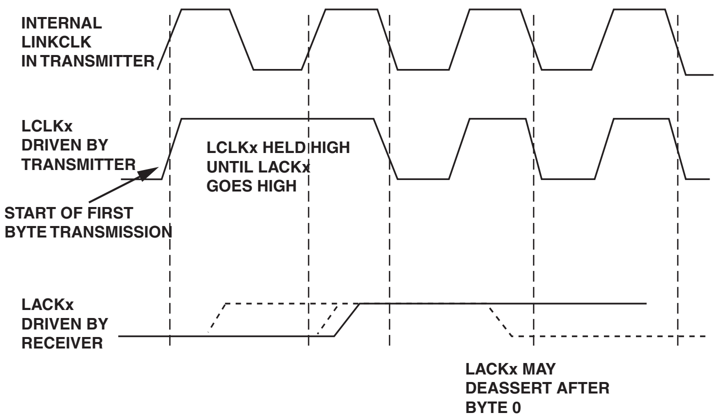

Figure 5-5. Relationship Between Internal LCLK and LCLK at Pads

Self-Synchronization

The link ports are designed to allow long distance connections to be made between the driver and the receiver. This is possible because the links are self-synchronizing—the clock and data are transmitted together. Only relative delay, not absolute delay between clock and data is relevant.

In addition, the `LACKx` signal inhibits transmission of the next word, not of the current nibble or byte. Since the processor operates on 4 bytes of data words and the link ports are 1 byte wide, each transaction has a length of 4. The receiver pulls `LACK` low after the first byte of the word being received causing the buffer to fill. This ensures that 3 `LCLK` cycles are available for deassertion propagation to the transmitter.

Multi-Master Conflicts

Multi-master conflicts can be resolved using token passing. In token passing, the token is a software flag that passes between processors. This is described in more detail in the following section "Example Token Passing".

The example shown in Figure 5-6 is a typical case where the link port is used as fast I/O link. A FPGA bridge is required to communicate between two different protocols. If using both link ports, full duplex operation is possible without core intervention.

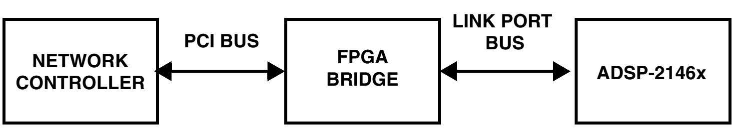

Figure 5-6. Fast I/O Link

Operating Modes

The following sections describe the operating modes of the link ports.

Receive Link Service Request Mode

A Link Service Request interrupt can be generated when an external source accesses the link port when the link port is disabled. For transmitter it requires that the LTRQ interrupt is unmasked by setting LTRQ_MSK bit in the LCTLx register.

When the transmitter is disabled, and the enabled receiver wants to initiate a transfer, it can drive LACKx high. When LACKx of the disabled link port is asserted, then a link service transmit request interrupt is generated and the receiver can initiate the transfer.

Transmit Link Service Request Mode

For the receiver, a Link Service Request requires that the LRRQ interrupt is unmasked by setting LRRQ_MSK bit in the LCTLx register.

When the receiver is disabled, and the enabled transmitter wants to initiate transfer, it can drive LCLKx high. When LCLKx of the disabled link port is asserted, then a Link service receive request interrupt is generated and the transmitter can initiate the transfer.

Example Token Passing

When two ADSP-2146x processors communicate using a link port, only one can be the transmitter or receiver. Token passing is a protocol that assists the DSPs alternate control. Figure 5-7 on page 5-11 shows a flow chart of the token passing process.

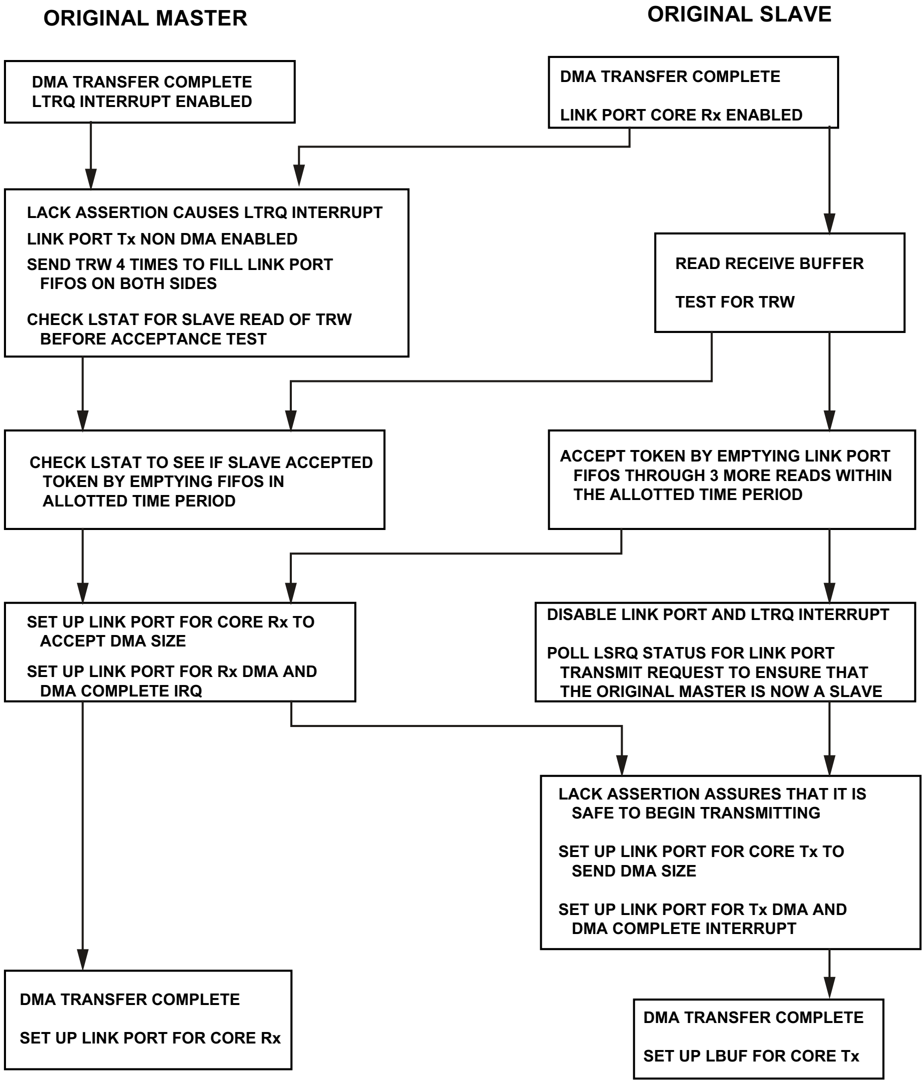

Figure 5-7. Token Passing Flow Chart

In token passing, the token is a software flag that passes between the processors. At reset, the token (flag) is set to reside in the link port of one device, making it the master and the transmitter. When a receiver link port (slave) wants to become the master, it may assert its LACKx line (request data) to get the master's attention. The master knows, through software protocol, whether it is supposed to respond with actual data or whether it is being asked for the token.

The token release word can be any user-defined value. Since both 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. If the received data is not the token release word, then the slave must assume the master was beginning a new transmission.

Through software protocol, the master can also request data by sending the *token release word* (TRW) without the LACKx (data request) going low first.

The following is a list of the areas of concern when a program implements a software protocol scheme for token passing.

- The program must make sure that both link ports are not enabled to transmit at the same time. In the event that this occurs, data may be transmitted and lost due to the fact that neither link port is driving LACKx. In the example, the TLRQ status bit is polled to ensure that the master becomes the slave before the slave becomes the master, avoiding the two transmitter conflict.

- The program must make sure that the link interrupt selection matches the application. If a status detection scheme using the status bits is to be used, it is important to note the following: If a link port that is configured to receive is disabled while LACKx is asserted, there is an RC delay before the external pulldown resistor on LACKx (if enabled) can pull the value below logic threshold. If the LTRQ status bit is unmasked (in this instance), then an LSR is latched and the LSRQ interrupt may be serviced, even though unintended, if enabled.

- The program must make sure that synchronization is not disrupted by unrelated influences at critical sections where timing control loops are used to synchronize parallel code execution. Disabling of nested interrupts is one technique to control this.

Data Transfer

The link ports are able to transfer data using DMA and core.

Packing Registers

The transmit shift and receive shift registers work with the FIFO buffers as described below.

Output Register

The transmit shift register receives byte wide FIFO data or register data (address) and serially shifts its data out externally off chip. The output can be controlled for generation of acknowledgements or can be manually over written. The transmit pack register is clocked with the rising edge.

Input Register

The receive shift register receives its data serially from off chip. Internally the receive shift register is byte wide and data received can either be transferred to the FIFO buffer or used in an address comparison. The receive pack register is clocked with the falling edge.

Note that the transmit/receive pack registers are not memory mapped.

Buffers

The transmit buffer registers (TXLBx) and receive buffer registers (RXLBx) buffer the data flow through the link port. The transmit and receive buffers consist of a 2 deep buffer and a shift register. The registers read from or write to internal memory under DMA or processor core control.

Transmit Buffer

In the transmit path, the TXLBx buffer is used to accept core data or DMA data from internal memory. Data is transferred to the shift register to send unpacked bytes to the ports. The least significant byte is transmitted first. As each word is unpacked and transmitted, the next location in the FIFO becomes available and a new DMA request is made if DMA is enabled. If the shift register becomes empty, the LCLKx signal is deasserted.

Receive Buffer

In the RXLBx receive buffer, data is transferred to the core or DMA from the buffer whereas the shift register performs the packing, least significant byte first (the least significant byte is placed in bits 7–0). The LACKx signal is deasserted by the receiver as soon as it receives the first byte from transmitter if the buffer already has a word (the receive buffer RXLBx is already half full). The packing is done as shown in Figure 5-8.

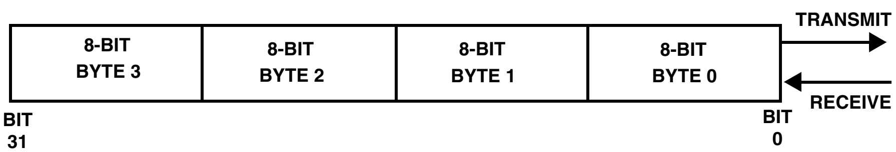

Figure 5-8. Link Port Output Packing Transmit/Receive Registers

For the ADSP-2146x processor, the least significant byte is transmitted first. This is different to legacy processors (ADSP-2116x) where the most significant byte is transmitted first.

Buffer Status

The entire receive and transmit path form a 3-stage FIFO. Two writes/reads can occur to the transmit/receive buffer by the core or DMA before it signals a full/empty condition. Full/empty status for the link buffer is shown by the FFST bits in the LSTATx register. If the link port is configured as a transmitter, then the FFST bits in the LSTATx register reflect the status of the TXLBx register. If the link port is configured as a receiver, then the FFST bits in LSTATx register reflect the status of RXLBx.

Buffer Reception Error

The LERR bit (LSTATx register) reports the byte packing complete status (complete/incomplete).

Flushing Buffers

Disabling the link port flushes the transmit and receive buffers.

Buffer Hand Disable

For more information, see "Buffer Hang Disable (BHD)" on page 5-23.

Core Transfers

In applications where the latency of link port DMA transfers to and from internal memory is too long, or where a process is continuous and has no block boundaries, the processor core may read or write link buffers directly using the full or empty status bit of the link buffer to automatically pace the operation. The full or empty status of a particular link buffer can be determined by reading the LSTATx bits in the LCTL register.

DMA should be disabled if reading or writing to the link port buffers.

If a read is attempted from an empty receive buffer, the core stalls (hangs) until the link port completes reception of a word. If a write is attempted to a full transmit buffer, the core stalls until the external device accepts the complete word. Up to four words (2 in the receiver and 2 in the transmitter) may be sent without a stall before the receiver core or DMA must read a link buffer register.

To support debugging buffer transfers, the processor has a buffer hang disable (LP_BHD) bit. When set (=1), this bit prevents the processor core from detecting a buffer-related stall condition, permitting debugging of this type of stall condition.

DMA Transfers

Each link port supports a DMA channel.

The link ports do not support internal to internal memory transfers like previous SHARCs (no link assignment register). If internal to internal memory transfers are required, refer to Chapter 6, "Memory-to-Memory Port DMA".

In standard DMA operations, the software needs to set up the DMA parameter registers before the link port control register is configured. After setting the DMA enable bit the transfer starts until the word count reaches zero, the DMA has finished.

Link Port DMA Group Priority

The link port 0 and 1 modules each have a DMA channel which are grouped together. When both channels have data ready, the channel arbitrates by rotating using a round robin method (which is the first arbitration stage). The winning channel requests the DMA bus arbiter to get control of the peripheral DMA bus (2nd stage of arbitration).

The I/O processor considers the two DMA channels as a single group and therefore one arbitration request. For more information, see "Peripheral DMA Arbitration" on page 3-36.

Interrupts

The following sections and Table 5-2 provide details on using link port interrupts.

Table 5-2. Link Port Interrupt Overview

Default Programmable Interrupt	Sources	Masking	Service
LP0I/LP1I not connected by default	- DMA complete - DMA chain TCB complete - core buffer access - internal transfer completion - access completion - link service request - invalid transmit attempt	N/A	ROC from LSTATx + RTI instruction

Sources

The link port interrupt is not connected by default to the IVT. Operating with status interrupts the corresponding `LP0I` or `LP1I` must be routed to a programmable IVT by using the `PICR` registers. The link ports generate interrupts under the conditions described in the following sections.

Core Buffer Service Request

When DMA is disabled the processor core may read from the RXLBx buffer or write to the TXLBx buffer. An interrupt is generated when the receive buffer is not empty or the transmit buffer is not full.

DMA Complete

A DMA channel interrupt is generated when a DMA block transfer through the link port with DMA enabled completes.

Internal Transfer Complete

The transmitter generates an internal transfer completion interrupt (DMACH_ IRPT_MSK = 1 and EXTTXFR_DONE_MSK = 0). Once the DMA count is zero, this interrupt is generated regardless of the state of the transmitter FIFO (traditional mode).

Access Complete

The transmitter generates an access completion interrupt (DMACH_IRPT_MSK = 0 and EXTTXFR_DONE_MSK = 1) once the external transfer is completed. When DMA is not enabled, this interrupt is generated when the transmitter FIFO is empty and the last byte has been transmitted. If using DMA, the transmitter checks if the DMA is complete.

Link Service Request

A link service request interrupt is generated when an external source accesses the link port when the link port is disabled. For example, if the enabled receiver wants to initiate a data transfer with the disabled transmitter, it can make LACKx high. When LACKx of the disabled link port goes high, then a link service request interrupt is generated. Now the receiver can initiate the transfer.

Chained DMA

For chained DMA, if the PCI bit is cleared (= 0), the DMA complete interrupt is generated only after the entire chained DMA access is complete. If the PCI bit is set (= 1), then a DMA interrupt is generated for each TCB.

Protocol Error

A link port invalid transmit (LPIT) is generated if the transmitter is driving LCLKx high because the receiver has not asserted LACKx and LCLKx goes low due to a processor reset (or some other reason, even though the receiver has not yet asserted LACKx). In this case, the receiving link port generates an interrupt.

Masking

The LP0I and LP1I signals are not routed by default to programmable interrupts. To service the LPxI, unmask (set = 1) any programmable interrupt bit in the IMASK/LIRPTL register.

Service

Status interrupts are latched and stored in the corresponding status register.

In the ISR, programs should read the corresponding status register (LSTATx) which clears the interrupt bits. Reading the status register when an interrupt occurs causes the core to hang until the interrupt bits are set in the status register. Otherwise, a simultaneous read and update of the status register results in a loss of information. This hang cannot be overridden with the BHD bit LPCTLx register.

Error interrupts are latched and stored in the corresponding status register.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

Link Port Effect Latency

After the link port registers are configured the effect latency is 2 PCLK cycles.

Programming Model

The following sections provide information on programming receive and transmit DMA and changing the link port clock.

Changing the Link Port Clock

The following programming sequence may be used to change the core-to-link port clock ratio only. Note that this procedure changes only the PLL output divider. Therefore programs do not need to wait 4096 CLKIN cycles (required only if the PLL multiplier or the INDIV bit is modified).

1. Disable the link ports. Note that the peripherals cannot be enabled when changing clock ratio.
2. Select the PLL divider by setting the PLLDx bits (bits 6–7 in the PMCTL register).

3. Select the link port clock divider (CCLK to LPCLK ratio) by setting the LPCKRx bits (bits 21 and 22 in the PMCTL register).

4. Enable the new divisors by setting the DIVEN bit (bit 9 in the PMCTL register).

5. Wait 15 CCLK cycles. During this time, programs must not execute any valid instructions. The LPCLK change does not happen on-the-fly. This means that when a clock ratio change is registered, the current clock cycle may get truncated before the change and the new clock cycle ratio start.

6. Enable link ports.

For more information on link port clocking and programming the PLL, see "Phase-Locked Loop (PLL)" on page 23-2.

Receive DMA

The following is the sequence that occurs when an external device transfers a block of data into the processor's internal memory using a link port.

Note that the link ports do not support internal to internal memory transfers like previous SHARCs. If internal to internal memory transfers are required, refer to "External Port DMA" on page 4-125.

1. The processor writes the DMA channel's parameter registers (index register IILBx, modify register IMLBx and count register CLBx) and initializes the link port for receive (LTRAN = 0).

2. The processor enables the link port by setting the LEN bit. DMA is enabled by setting the LDEN bit in the LCTLx register.

3. The external device begins writing data to the RXLBx buffer through the link port.

4. The RXLBx buffer detects that data is present and sends a internal DMA request.

5. After the request is granted, the DMA transfer is performed thereby emptying the RXLBx buffer FIFO.

Transmit DMA

The following is the sequence that occurs when the processor transfers a block of data from its internal memory to an external device using link port.

1. The processor writes the DMA channel's parameter registers (index register IILBx, modify register IMLBx and count register CLBx) and initializes the link port for transmit (LTRAN = 1).

2. The processor enables the link port by setting the LEN bit and enables the link port DMA by setting the LDEN bit in the LCTLx register. Because this is a transmit, setting LDEN automatically asserts an internal DMA request.

3. After the request is granted the internal DMA transfer is performed filling the TXLBx buffer's FIFO.

4. The external device begins reading data from the TXLBx buffer through the link port.

5. The TXLBx buffer detects that there is room in the buffer (partially empty) and asserts another DMA request continuing the process.

Debug Features

The following sections provide information on features that help in debugging link port software.

Shadow Register

For ease of debug all registers are available as shadow registers.

- For the transmit path, the TXLB1-0_IN_SHADOW register is the data buffer while the TXLB1-0_OUT_SHADOW register is the pack register which is connected to the link port data.
- For the receive path, the RXLB1-0_IN_SHADOW register is the data buffer while the RXLB1-0_OUT_SHADOW register is the pack register which is connected to the link port data.

Reading these registers does not change link port status. Moreover, the LSTAT1-0_SHADOW registers allows programs to read the status and clear the interrupt bits.

Buffer Hang Disable (BHD)

A buffer hang disable (BHD) bit has been provided in the control register (LPCTLx). Setting this bit to 1 prevents the core from hanging when a read from an empty receive buffer or a write to a full transmit buffer is attempted. If the BHD bit is set and a read is performed from an empty receive buffer, then the previous data is returned. Writing to a full transmit buffer with the BHD bit set overwrites the existing data.

6 MEMORY-TO-MEMORY PORT DMA

Table 6-1 shows the memory-to-memory DMA port specifications.

Table 6-1. MTM Port Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	No
SRU DAI Default Routing	N/A
SRU2 DPI Required	No
SRU2 DPI Default Routing	N/A
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	No
Transmission Simplex	Yes
Transmission Half Duplex	No
Transmission Full Duplex	No
Access Type	
Data Buffer	Yes
Core Data Access	No
DMA Data Access	Yes
DMA Channels	2
DMA Chaining	No

Table 6-1. MTM Port Specifications (Cont'd)

Feature	Availability
Boot Capable	No
Local Memory	No
Clock Operation	f_{PCLK}

Features

The memory-to-memory port incorporates:

- 2 DMA channels (read and write)
- Internal to internal transfers
- Data engine for DTCP applications (only for special part numbers)

Note that the SHARC supports another internal to internal DMA module (external port) which supports multiples DMA modes.

Register Overview

MTM Control Register (MTMCTL). Enables the read and write DMA channels across the internal memory and returns status about the read or write DMA channel.

Clocking

The fundamental timing clock of the MTM is peripheral clock (PCLK). The clock to this module may be shut off for power savings. For more information, see "Disabling Peripheral Clocks" on page 23-11.

Functional Description

The MTM module owns two DMA channels one for read and one for write including a data buffer which stores up to 2x32-bit data. After the DMA is configured, the read DMA channel fills the buffer with 64-bit data. After this transfer, the write DMA channel becomes active and empties the buffer according to its destination. This procedure is repeated until the DMA count is zero.

The memory-to-memory DMA controller is capable of transferring 64-bit bursts of data between internal memories.

The MTM controller supports data in normal word address space only (32-bit). External to external DMA transfers are not supported.

Data Transfer Types

The memory-to-memory DMA controller is capable of transferring 64-bit bursts of data between internal memories.

Buffer

For memory-to-memory transfers the two stage buffer is the interface between the write and read channels.

The write channel fills up the buffer first which triggers the read channel. After two reads the buffer becomes empty which re-triggers the write channel. MTM performance is therefore dependent on the buffer depth.

Buffer Status

The buffer status can't be directly read from the control register. However both DMA channels (read and write) return the status if the channels are pending or active MTMDMAxACT bits

Flushing the Buffer

The MTMFLUSH bit in the MTMCTL register can be set to flush the FIFO and reset the read/write pointers. Setting and resetting the MTMDEN bit only starts and stops the DMA transfer, so it is always better to flush the FIFO along with MTMDEN reset.

Note that the MTMFLUSH bit should not be set along with the MTMDEN bit set. Otherwise the FIFO is continuously flushed leading to DMA data corruption.

DMA Transfer

Two DMA channels are used for memory-to-memory DMA transfers. The write DMA channel has higher priority over the read channel. The transfer is started by a write DMA to fill up the MTM buffer with a 2 x 32-bit word. Next, the buffer is read back over the same IOD bus to the new destination. With a two position deep buffer and alternate write and read access over the same bus, throughput is limited. The memory-to-memory DMA control register (MTMCTL) allows programs to transfer blocks of 64-bit data from one internal memory location to another. This register also allows verification of current DMA status during writes and reads.

Interrupts

There are two DMA channels; one write channel and one read channel. When the transmission of a complete data block is performed, each channel generates an interrupt to signal that the entire block of data has been processed. Note that the write and read interrupts (`P15I`, if the `MTMI` bit in the `IMASK` register is enabled) are very close to each other, so only one interrupt is triggered.

Table 6-2 provides an overview of MTM interrupts.

Table 6-2. MTM Interrupt Overview

Default Programmable Interrupt	Sources	Masking	Service
MTM = P15I	DMA write complete DMA read complete	N/A	RTI instruction

Sources

There are two interrupt signals—one for the write and one for read channel. The MTM port can generate interrupts under the conditions described in the section below.

DMA Complete

When the transmission of a complete data block is performed, each DMA channel (write/read) generates an interrupt to signal that the entire block of data has been processed. If both channels are enabled interrupts occur very close to each other, and the read interrupt is aborted. This is because the read interrupt is dependent on write interrupt .

Masking

The `MTMI` signal is routed by default to programmable interrupt. To service the `MTMI`, unmask (set = 1) the `P15I` bit in the `IMASK` register.

For example:

```
bit set IMASK P15I; /* unmasks P15I interrupt */
```

Service

Interrupts are serviced with a RTI (return from interrupt) instruction.

MTM Throughput

Data throughput for internal to internal transfers is 12 PCLK cycles for 64-bit data.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

MTM Effect Latency

After the MTM register is configured the effect latency is 1.5 PCLK cycles minimum and 2 PCLK cycles maximum.

Programming Model

This data transfer can be set up using the following procedure.

1. Program the DMA registers for both channels.
2. Set (=1) the MTMFLUSH bit (bit 1) in the MTMCTL register to flush the FIFO and reset the read/write pointers.
3. Set (=1) the MTMEN bit in the MTMCTL register.

   A two-deep, 32-bit FIFO regulates the data transfer through the DMA channels.

7 FFT/FIR/IIR HARDWARE MODULES

Finite Impulse Response (FIR) filters are frequently used in DSP applications. With its high performance floating-point processing capabilities the SHARC processors are uniquely designed for FIR filtering. The SIMD SHARC core has two MAC units which provide 800 MIPS of processing speed when the processor is running at 400 MHz. However, for high performance applications, with their ever increasing complexity (such as room equalization or surround sound), even more processing power is needed.

Each of the accelerator modules (FFT/FIR/IIR) have access to the internal memory only.

To meet this need, the ADSP-214xx SHARC processors off load some of the most frequently used and intensive processing into hardware accelerators. An accelerator dedicated for filter processing can reduce the instruction processing load on the core, freeing it up for other tasks.

The FIR/IIR/FFT accelerator units are capable of performing the filters and FFT without core intervention. This gives software developers enormous freedom to use core processing cycles to implement complex algorithms, effectively adding more instructions per second to the processor.

The accelerator modules (FFT/FIR/IIR) each have local memory which is not accessible by the core during regular operation mode.

The interface specifications are shown in Table 7-1.

Table 7-1. Accelerator Specifications

Feature	FFT/FIR/IIR
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	No
SRU DAI Default Routing	N/A
SRU2 DPI Required	No
SRU2 DPI Default Routing	N/A
Interrupt Control	Yes
Protocol	
Master Capable	N/A
Slave Capable	N/A
Transmission Simplex	N/A
Transmission Half Duplex	N/A
Transmission Full Duplex	N/A
Access Type	
Data Buffer	Yes
Core Data Access	No
DMA Data Access	Yes
DMA Channels	2
DMA Chaining	Yes
Boot Capable	N/A
Local Memory	Yes (RAM)
Clock Operation	f_{PCLK}

FFT Accelerator

The FFT accelerator (shown in Figure 7-1) implements radix-2 complex floating-point FFT. The accelerator's data and twiddle coefficient interface is designed to connect to the processor's DMA engine (acting like a peripheral) and implements a synchronous pipeline read/write protocol with a pipeline depth of 1.

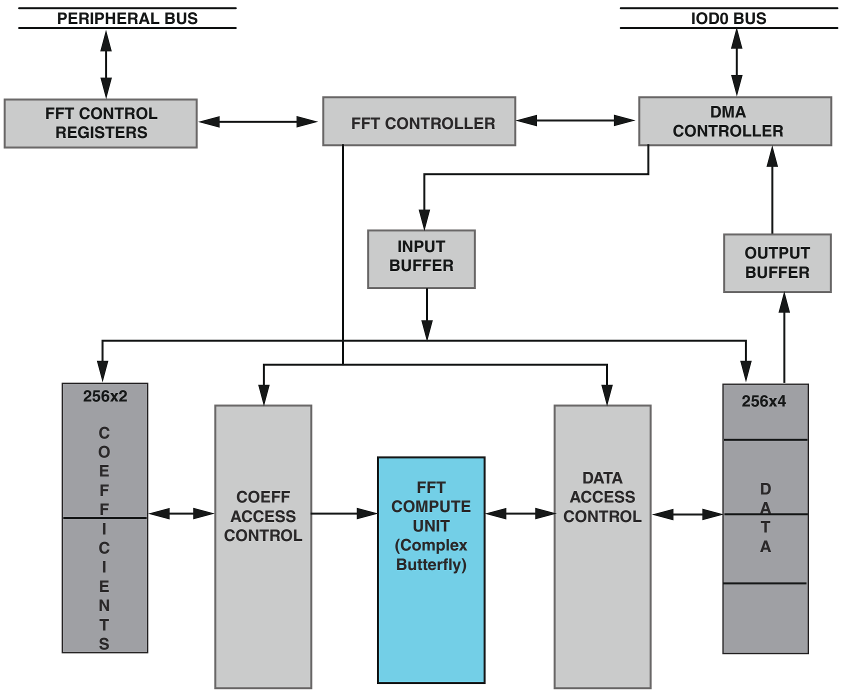

Figure 7-1. FFT Block Diagram

Features

The following list describes the features available through the FFT accelerator.

- Supports FFT sizes from 16 – $8k^2$ points all handled by DMA with no core intervention.
- Computes a radix 2 decimation in time algorithm with automated bit reversal.
- Contains a 1024 32-bit word data memory unit.
- Contains a 512 32-bit word twiddle coefficients memory unit.
- Contains a compute block unit with four floating-point multipliers and six floating-point adders.
- Has a control unit with configuration registers, responsible for all memory addresses and strobe generation.
- Contains a 8 × 32 deep input/output FIFO unit.

Register Descriptions

The accelerator has two control and two status registers that are used to program and check operation of the module. The module also contains DMA registers which are described in"I/O Processor" in Chapter 3, I/O Processor.

Power Management Control Register (PMCTL1). Used for FFT accelerator selection. Controls the clock power down to the module if not required.

Global Control Register (FFTCTL1). Used to enable, start, and reset the FFT module. It is also used to enable DMA and debug operation.

Control Register (FFTCTL2). Used to configure individual FFT parameters (such as length) and how the module process the FFT, such as data packing.

MAC Status Register (FFTMACSTAT). Reports errors and status on the multiply/accumulator.

DMA Status, Shadow DMA Status Registers (FFTDMASTAT, FFTSHDMASTAT). Provide information on DMA operations such as DMA progress and chain pointer loading.

Clocking

The FFT accelerator runs at the maximum speed of the peripheral clock (PCLK). The clock to this module may be shut off for power savings. For more information, see "Disabling Peripheral Clocks" on page 23-11.

Functional Description

The FFT accelerator is comprised of a compute block, data memory and coefficient memory. The design allows programs to off-load an FFT calculation by initializing few TCBs and control registers. In this way, the FFT accelerator can perform the FFT calculation in the background while the core is busy doing some other useful task. It can interrupt the core once the processing is complete. The following sections provide functional details of the FFT accelerator.

Compute Block

The compute block contains one complex butterfly stage (based on four IEEE floating-point multipliers and six IEEE floating-point adders) whose operation is pipelined and simultaneous.

Data Memory

The accelerator has a 1024 location deep, 32-bit wide data memory, organized into four independent blocks. Blocks are grouped in sets of two that are used to fetch or store real and imaginary parts of data simultaneously. Fetches and stores are accomplished by ping-ponging the read and write buffers.

Coefficient Memory

The accelerator has a 512 location deep, 32-bit wide twiddle memory, organized into two independent blocks (256x2). It allows fetching real and imaginary twiddles simultaneously.

Accelerator States

The FFT accelerator has five different states:

1. Reset
2. Idle
3. Reading
4. Processing
5. Writing

These states are described in detail in the following sections.

Reset State

Reset mode is activated either by setting the FFT_RST bit in the FFTCTL1 register or by applying logic low to the $\overline{\texttt{RESET}}$ input pin.

If reset is activated by setting the FFT_RST bit, this bit must be cleared to bring the accelerator out of reset.

Resetting via a logic low to the $\overline{\text{RESET}}$ pin resets all registers, thereby clearing the FFT_RST bit. Once the processor is brought out of reset by applying a logic high to the $\overline{\text{RESET}}$ pin, the FFT module goes into the idle state in the next clock cycle.

Idle State

This mode is used to program the accelerator's control registers. Setting the FFT_EN and FFT_START bits in the FFTCTL1 register moves the state from idle to reading.

Read State

In this state the module reads data and coefficients, but counts the number of read data only. This is because for successive FFT calculations the coefficient need not be read again—only the next set of data has to be read. When a specified number of data words are read, the state automatically moves to processing.

Processing State

In this mode the module computes FFT ping-pong stages in memory. Once this is done, the state automatically moves to the write state.

Write State

In this mode all the computed data is written out to internal memory. The state then automatically changes to either idle or read, depending on the way the block is configured using the repeat function (FFT_RPT bit in the FFTCTL2 register). If the FFT_RPT bit is set, the block moves to the read state, if cleared, the block moves to the idle state. The FFT_RPT bit is useful when programs need to continuously perform an FFT on input data without core intervention.

Internal Memory Storage

This section describes the required software buffers in internal memory and the required storage model for data and coefficients using the FFT accelerator.

Small FFT N<=256

To run a small FFT three buffers are required:

- Input Buffer [2 × N] packed or unpacked data
- Output Buffer [2 × N] packed or unpacked data
- Coefficient Buffer [2 × N]

Unpacked Data

If unpacked data is selected this is the required input format or output format. Programs can optionally select the input or output data streams to be unpacked. In this mode, the first samples are all real followed by the imaginary samples.

```
RE[0], RE[1], ...RE[N-1], IM[0], IM[2], ... IM[N-1]
```

This can be independent for the input or output data streams.

Packed Data

The default format for packed data is as follows.

```
RE[0], IM[0], RE[1], IM[1], .... RE[N-1], IM[N-1]
```

Twiddles

The default format for coefficient buffer (twiddles) is as follows.

```
Re(CF[0]), Im(CF[0]), -Im(CF[0]), Re(CF[0]), Re(CF[1]),
Im(CF[1]), -Im(CF[1]), Re(CF[1]),.......... Re(CF[N/2-1]),
Im(CF[N/2-1]), -Im(CF[N/2-1]), Re(CF[N/2-1]) (4xN/2 = 2N words)]
```

Large FFT N>256

To run a large FFT, 6 buffers are required:

- Input Buffer [2 × N] (packed data)
- Special Buffer [2 × N] (intermediate buffer used in step 1 for vertical FFT and in step 2 for special product = Product of vertical buffer with special twiddles)
- Output Buffer [2 × N] (packed data)
- Vertical complex Coeff Buffer [2 × V]
- Horizontal complex Coeff Buffer [2 × H]
- Special complex Coeff Buffer [4 × N]

Twiddles

For N>256, the FFT accelerator follows the 'Divide and Conquer' approach. Therefore, three types of coefficient buffers are required:

Complex Coefficient buffer for V point FFT

```
Re(CF[0]), Im(CF[0]), -Im(CF[0]), Re(CF[0]),
Re(CF[1]), Im(CF[1]), -Im(CF[1]), Re(CF[1]),
..........
Re(CF[V/2-1]), Im(CF[V/2-1]), -Im(CF[V/2-1]), Re(CF[V/2-1])
(4xV/2 = 2V words)
```

Complex Coefficient buffer for H point FFT

```
Re(CF[0]), Im(CF[0]), -Im(CF[0]), Re(CF[0]),
Re(CF[1]), Im(CF[1]), -Im(CF[1]), Re(CF[1]),
..........
Re(CF[H/2-1]), Im(CF[H/2-1]), -Im(CF[H/2-1]), Re(CF[H/2-1])
(4xH/2 = 2H words)
```

Special complex coefficient buffer

```
Re(SP_CF[0]), Im(SP_CF[0]), -Im(SP_CF[0]), Re(SP_CF[0]),
Re(SP_CF[1]), Im(SP_CF[1]), -Im(SP_CF[1]), Re(SP_CF[1]),
..........
Re(SP_CF[N-1]), Im(SP_CF[N-1]), -Im(SP_CF[N-1]), Re(SP_CF[N-1])
(4N words).
```

Where:

`Re(CF[x])` = Real part of the complex coefficient `CF[x]`,

`Im(CF[x])` = Imaginary part of the complex coefficient `CF[x]`,

`Re(SP_CF[x])` = Real part of special complex coefficient `SP_CF[x]`,

`Im(SP_CF[x])` = Imaginary part of special complex coefficient `SP_CF[x]`

SP_CF[n]=WN^vxh

where n = vxH + h, h = 0, 1,…. H–1, v = 0, 1,…. V–1.

For more information see EE-322, *Expert Code Generator for SHARC Processors*. This EE note can be fount on the Analog Devices web site.

Operating Modes

The following sections describe FFT processing types and methods.

Small FFT Computation (<= 256 Points)

A small FFT (`NOVER256` = zero) can be handled completely in one step since the twiddles and input data stream fit in the local memories for twiddles and data. In this way two input TCBs (twiddles and data) are fed into the accelerator. After performing the FFT the output TCB writes the results back into the internal memory and the next FFT can start.

Large FFT Computation (> 256 Points)

For large FFTs (`NOVER256` = non zero) the model looks different since the twiddle/data do not fit completely into the local memories. The FFT computation is matrix based on rows (horizontal) and columns (vertical) and performed in three steps:

```
x(0) x(1) x(2) . . . . x(H - 1) x(H) x(H + 1) x (H + 2) . . .
x(2H - 1) x(2H) x(2H + 1) x(2H + 2) . . . . x(3H - 1) | | | . . |
x((V - 1)H) x((V - 1)H + 1) x((V - 1)H + 2) . . . . x(VH - 1)
```

1. The vertical (column) V Point FFTs are performed on the matrix.
2. The output of step 1 is multiplied by special twiddles (special coefficients).
3. Horizontal (row) H Point FFTs are performed on the output matrix of Step 2. This produces the final FFT on vertical columns (column wise).

The final FFT result is obtained in internal memory, not local memory.

Example for FFT Size N=512

This example shows a large FFT matrix of V × H = 32 × 16.

Vertical FFT

1. Input coeff DMA from vertical coeff buffer[64]
2. Input data DMA from input buffer[1024] (modifier = 2H, circbuf = 2N)
3. FFT computation
4. Output DMA to special buffer[1024]

Special Product—Number of Iterations is N/128 = 4

1. First iteration:
   a. Input coeff DMA from special coeff buffer[512]
   b. Input DMA from special buffer[256]
   c. FFT computation
   d. Output DMA to Special buffer[256]
2. Second iteration:
   a. Input coeff DMA from special coeff buffer[512] (offset = 512)
   b. Input DMA from special buffer[256] (offset = 256)
   c. FFT computation
   d. Output DMA to special buffer[256] (offset = 256)
3. Third iteration:
   a. Input coeff DMA from special coeff buffer[512] (offset = 1024)
   b. Input DMA from special buffer[256] (offset = 512)
   c. FFT computation
   d. Output DMA to special buffer[256] (offset = 512)
4. Fourth iteration:
   a. Input coeff DMA from special coeff buffer[512] (offset = 1536)
   b. Input DMA from special buffer[256] (offset = 768)

c. FFT computation

d. Output DMA to special buffer[256] (offset = 768)

Horizontal FFT

1. Input coeff DMA from horizontal coeff buffer[32]
2. Input data DMA from special buffer[1024] (modifier = 2V, circbuf = 2N)
3. FFT computation
4. Output DMA to Output Buffer[1024] (modifier = 2V, circbuf = 2N)

This FFT generates a total of six partial FFT computations. Each computation has an input and output DMA which results in 12 interrupts.

No Repeat Mode

If the FFT_RPT bit = 0, after FFT_START = 1 the accelerator moves from the idle state into the read state (input DMA). After the read completes, the accelerator moves into the processing state then the write state to read the results back into internal memory. The accelerator ends in the idle state.

For large FFTs (based on the VDIM, HDIM and NOVER256 bits) the accelerator knows when the entire FFT processing has finished.

Repeat Mode

If the FFT_RPT bit = 1, after FFT_START = 1 the accelerator moves from the idle state into the read state (input DMA). After the read completes, the accelerator moves into the processing state then the write state to read the results back into internal memory. The accelerator then moves automatically back into the read state for the next FFT frame. In this state multiple linked TCBs which were executed during the first iteration are re-used.

For large FFTs (based on the configuration of the VDIM, HDIM and NOVER256 bits) the accelerator knows when the entire frame processing has finished in order to re-load the new FFT frame parameters at the right time.

Unpacked Data Mode

For small FFTs (FFT<=256), the unpacked data mode can be selected independently for the input or output streams through the use of the FFT_CPACKIN or FFT_CPACKOUT bits (FFTCTL2 register).

> The FFT_CPACKIN/FFT_CPACKOUT settings are not applicable for < 512 points. The input is always expected to be in alternate real and imaginary format and the output is always generated in the same format.

Inverse FFT

The inverse FFT uses the same algorithm as the forward FFT. The accelerator takes advantage of this fact when processing IFFTs by setting up a coefficient TCB with change of sign for the sine twiddles (FFT uses twiddles cosine, sine, -sine, cosine, the IFFT uses cosine, -sine, sine, cosine). When TCB loading completes, the accelerator processes the inverse FFT and returns the data into the local data memory. Finally, in write mode, data is returned to internal memory.

In order to get the correct amplitude for the inverse FFT, the output buffer needs to be scaled by 1/N.

Data Transfer

The FFT accelerator works exclusively through DMA and therefore does not require core intervention. This allows the core to perform other system tasks. The core is used to configure the DMA parameter registers and the accelerator control registers and to start accelerator operation.

FFT Buffers

As shown in Figure 7-1 on page 7-3, the input and output DMA stream each pass an 8 deep buffer. These I/O buffers ensure that the FFT stream of the accelerator is not stalled during high DMA bus loads. Note that the buffer status cannot be read.

Buffer Status

Buffer status cannot be read.

Flushing the Buffer

The FFT does not have any control bit for flushing the buffers. The buffers are flushed by entering into reset mode.

DMA Transfers

The FFT accelerator supports circular buffer chained DMA. Two TCB structures are associated with input and output DMA. The input TCB structure is used for transferring either data or coefficients to the accelerator block and the output TCB is used for receiving data from the FFT block to the internal memory of the SHARC processor. For TCB structure details see “FFT Accelerator TCB” on page 3-19.

DMA Channels and TCB Structure

The accelerator has two DMA channels that connect to internal memory. The channels fetch the data and coefficients from internal memory and store the results to internal memory. The DMA controller supports circular buffer chain pointer DMA. Separate TCBs must be created for both input and output DMA.

Note that bit 20 of the input chain pointer register (`FFTICP`) indicates whether the TCB is for loading data or coefficients. If the TCB is a coefficient TCB, then circular buffering is not supported and the input length and base registers are ignored.

Table 3-21 on page 3-19 and Table 3-22 on page 3-19 show the input and output TCB structures.

Chained DMA

The DMA controller supports circular buffer chain pointer DMA. The input TCB structure consists of index, modify, count and chain-pointer register values for input data. The input TCB also consists of length and base pointer register values to support circular buffering. Similar to the input TCB structure, the output TCB also consists of index, modify, count, chain pointer, length and base pointer register values to support circular buffered chained DMA for output data.

Once the accelerator is enabled, it loads the TCB values pointed to by the chain pointer register value into its internal registers. The FFT accelerator uses the input TCB values to fetch coefficients and data. It then computes the FFT on the fetched data without any core intervention. Once the computing is complete, the results are stored into the internal memory of the processor using the TCB values of the output TCB registers. If the repeat bit (FFT_RPT) is set, the accelerator goes continues on a new FFT frame once the current FFT frame is processed.

One or more Transfer Control Blocks (TCB) chained to each other may need to be configured for both input and output DMA channels. Each of these TCBs may contain any of the following.

- DMA parameter register values for input data.
- DMA parameter register values for twiddles load.
- DMA parameter register values for output data.

- Intermediate results for large FFT are stored in the internal memory.
  - DMA parameter register values for intermediate input/output data (required only for large FFT).
  - DMA parameter register values for special twiddles (required only for small FFT)

The circular access type is used for large FFTs to process the entire FFT (VxH) matrix.

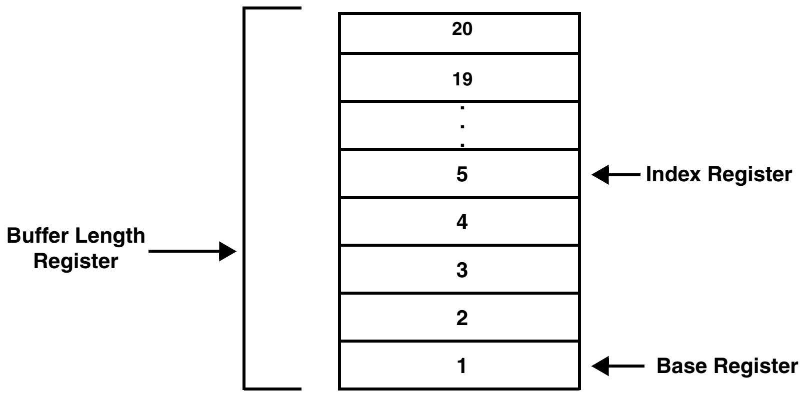

Figure 7-2. Circular Buffer Addressing

Interrupts

Table 7-2 provides an overview of FFT interrupts.

Table 7-2. FFT Interrupt Overview

Default Programmable Interrupt	Sources	Masking	Service
ACC0I/ACC1I not connected by default	Input DMA complete Output DMA complete	N/A	ROC from FFTDMASTAT + RTI instruction
	MAC IEEE floating point exceptions	N/A	ROC from FFTMACSTAT + RTI instruction

Sources

The FFT module drives two interrupt signals, ACC0I for the DMA status and ACC1I for the MAC status. The FFT module generates interrupts as described in the following sections.

DMA Complete

The DMA interrupt is shared by the input and output DMA. They are generated at the end of every chain or at the end of an entire DMA sequence, depending on the PCI value in the respective chain pointer registers. The interrupt follows the access completion rule, where the interrupt is generated when all data are written back to internal memory.

MAC Status

A MAC status interrupt is generated under these conditions

- MAC underflow – Set if MAC result to small
- MAC overflow – Set if MAC result overflows
- MAC not a number – Set if number is not IEEE compliant
- MAC denormal – Set if number is not IEEE compliant

Chained DMA

For chained DMA, if the PCI bit is cleared (= 0), the DMA complete interrupt is generated only after the entire chained DMA access is complete. If the PCI bit is set (= 1), then a DMA interrupt is generated for each TCB.

Masking

The ACC0I and ACC1I signals are not routed by default to programmable interrupts. To service the ACCxI, unmask (set = 1) any programmable interrupt bit in the IMASK/LIRPTL register.

Service

When a DMA interrupt occurs, programs can find whether the input DMA interrupt occurred or the output DMA interrupt occurred by reading the DMA status register (FFT_DMASTAT). The DMA interrupt status bits are sticky and are cleared by a read.

When a MAC status interrupt occurs, programs can find whether the MAC interrupt occurred by reading the MAC status register (FFT_MACSTAT). The MAC interrupt status bits are sticky and are cleared by a read.

FFT Performance

In this section:

V = Number of rows
H = Number of columns
N = V × H

- Reads from internal memory take 2 cycles/word.
- Writes to internal memory take 1 cycle/word.
- It takes 2 PCLK cycles to compute a single complex butterfly by the FFT computation.

For performance consideration each FFT computation is accompanied with a preceding Read DMA and a post write DMA.

Small FFT (N is <= 256)

Data reads: 2N × 2
Butterfly computes: N log2N cycles (A radix2 takes N/2 log2N × 2 PCLK cycles)
Data write: 2N × 1

Large FFT (N >= 256)

Total number of performance cycles = (Vertical FFT + Special Prod + Horizontal FFT) cycles.

Vertical FFT Cycles

Data and coefficient reads: 2N × 2 + 2V × 2
Butterfly computes: (Vlog2V) × H
Data writes: 2N × 1

Special Prod Cycles

Data and coefficient reads: 2N × 2 + 4N × 2
Product compute: 2 × 4N/4
Data writes: 2N × 1

Horizontal FFT Cycles

Data and coefficient reads: 2N × 2 + 2H × 2
Butterfly compute: (Hlog2H) × V
Data writes: 2N × 1

For the large FFT mode the entire expression is 28N + Nlog2N + 4 × (V+H) PCLK cycles. Only the term dependent on H and V is 4. In order to improve performance V and H should be selected in such a way that H + V is minimum.

For N=512, (H,V) should be (16,32) or (32,16).
For N=1024, (H,V) should be (32,32) and so on.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

FFT Accelerator Effect Latency

After the FFT registers are configured the effect latency is 1.5 PCLK cycles minimum and 2 PCLK cycles maximum. Writes to the PMCTL1 register have an effect latency of two PCLK cycles. Wait for at least four CCLK cycles after selecting an accelerator before accessing any of its registers.

Programming Model

There are two separate programming models, one for a FFT that fits in the accelerator's internal memory (N = 256 points or less) and one for a FFT that is larger than the accelerator's internal memory (N = 512 points or more). In both models, is assumed that the accelerator starts in idle mode.

N <= 256, No Repeat

For details on the storage format of the coefficients see "Internal Memory Storage" on page 7-8.

1. Configure the ACCSEL bits in the PMCTL1 register to select the FFT accelerator.

2. Program the FFTCTL2 register with:
   VDIM = N/16
   LOG2VDIM = Log2(N)
   HDIM = 0
   LOG2HDIM = 0
   FFT_RPT = 0
   FFT_CPACKIN/FFT_CPACKOUT = 0 or 1 depending on whether input/output data is packed into complex words or sent/received data is real or imaginary.

3. Set (=1) the FFT_RST bit in the FFTCTL1 register and wait for a minimum of 4 CCLK cycles.

4. Program control register FFTCTL1 with:
   FFT_RST = 0
   FFT_EN = 1
   FFT_START = 1
   FFT_DMAEN = 1
   FFT_DEBUG = 0
5. Configure a coefficient DMA to read N complex twiddle factors from the coefficient buffer into the accelerator (total of 2N 32- bit words) and wait until the DMA is complete (or chain DMA in Step 4). This step is not needed if twiddles are already in the coefficient memory of the accelerator.
6. Configure a data DMA to read N complex data points from the input buffer into the accelerator (total of 2N 32-bit words).
7. Configure a data DMA to write N complex data points from the accelerator into the output buffer (total of 2N 32-bit words). There is no need to wait until the DMA in Step 6 completes.
8. Wait until the DMA in Step 7 completes (by interrupt or polling). The computed FFT is now in the core's internal memory and the accelerator is in idle mode.

N <= 256, Repeat

For details on the storage format of the coefficients see "Internal Memory Storage" on page 7-8.

1. Configure the ACCSEL bits in the PMCTL1 register to select the FFT accelerator.
2. Program the FFTCTL2 register with:
   VDIM = N/16
   LOG2VDIM = Log2(N)
   HDIM = 0
   LOG2HDIM = 0

FFT_RPT = 1
FFT_CPACKIN/FFT_CPACKOUT = 0 or 1 depending on whether input/output data is packed into complex words or sent/received data is real or imaginary.

3. Set (=1) the FFT_RST bit in the FFTCTL1 register and wait for a minimum of 4 CCLK cycles.

4. Program the FFTCTL1 register with:
   FFT_RST = 0
   FFT_EN = 1
   FFT_START = 1
   FFT_DMAEN = 1
   FFT_DEBUG = 0

5. Configure a coefficient DMA to read N complex twiddle factors from the coefficient buffer into the accelerator (total of 2N 32- bit words) and wait until the DMA is complete (or chain DMA in Step 4). This step is not needed if twiddles are already in the coefficient memory of the accelerator.

6. Configure a data DMA to read N complex data points from the input buffer into the accelerator (total of 2N 32-bit words).

7. Configure a data DMA to write N complex data points from the accelerator into the output buffer (total of 2N 32-bit words). There is no need to wait until the DMA in Step 6 completes.

8. Wait until the DMA in Step 7 completes (by interrupt or polling). The computed FFTs is now in the core's internal memory and the accelerator is in reading mode, waiting for next batch of FFTs.

N >= 512, No Repeat

For details on the storage format of the coefficients see "Internal Memory Storage" on page 7-8.

Configure the FFT Control Register

1. Configure the ACCSEL bits in the PMCTL1 register to select the FFT accelerator.
2. Factor N = VH, where 16 ≤ V and 16 ≤ H.
3. Set (=1) the FFT_RST bit in the FFTCTL1 register and wait for a minimum of 4 CCLK cycles.
4. Program the FFTCTL2 register with
   VDIM = V/16
   LOG2VDIM = Log2(V)
   HDIM = H/16
   LOG2HDIM = Log2(H)
   NOVER256 = VH/256
   FFT_RPT = 0
5. Program the FFTCTL1 register with
   FFT_RST = 0
   FFT_EN = 1
   FFT_START = 1
   FFT_DMAEN = 1
   FFT_DEBUG = 0

Vertical FFT Configuration

6. Configure a coefficient DMA to read 2V twiddle factors from the vertical coeff buffer into the accelerator (total of 2V 32-bit words) and wait until the DMA is complete (or chain DMA in Step 7). This step is not needed if twiddles are already in the coefficient memory of the accelerator.

7. Configure a data transmit DMA to load 2N – 1 data points from the input buffer into the accelerator with a modify value of 2H, and a circular buffer length of 2N – 1. Chain a data transmit DMA of count = 1 that loads the last imaginary point.

The FFT_CPACKIN/FFT_CPACKOUT settings are not applicable for N ≥ 512 points. The input is always expected to be in alternate real and imaginary format and the output is always generated in the same format.

8. Configure a data receive DMA to read 2N data points from the accelerator into the special buffer with a modify of 1. There is no need to wait until the DMA in Step 6 completes.

Special Buffer Configuration

9. Configure a DMA to load special coefficients from the special coefficients buffer into the accelerator, with a count = 512.

10. Once the DMA in Step 9 completes, configure a data DMA (chained or via interrupt) to read 256 data points (count = 256) from the special buffer into the accelerator with a modify value = 1.

11. Configure a data DMA to write 256 data points (count = 256) from the accelerator into the special buffer with modify value = 1. There is no need to wait until the DMA in Step 9 completes.

12. Repeat step 9 N/128 times (offset processing the entire 2N buffer of data).

Horizontal FFT Configuration

13. Once the last DMA in Step 10 completes, configure a coefficient DMA to read 2H twiddle factors from the horizontal coeff buffer into the accelerator.

14. Once the DMA in Step 12 completes, configure a data DMA (chained or via interrupt) to read 2N – 1 data points from special buffer into the accelerator with a modify value = 2V and a circular buffer length of 2N – 1. Chain a data DMA of count = 1 that reads the last imaginary point.

15. Configure a data DMA to write 2N–1 data points from the accelerator into the output buffer with a modify value = 2V and a circular buffer length of 2N – 1. There is no need to wait until the DMA in Step 9 completes. Chain a data DMA of count = 1 that reads the last imaginary point.

16. Wait until the DMA in step 14 completes (by interrupt or polling). The computed FFT is now in the output buffer and the accelerator is in idle mode.

N >= 512, Repeat

For details on the storage format of the coefficients see "Internal Memory Storage" on page 7-8.

Transmit DMAs take place using input TCBs; receive DMAs take place using output TCBs.

1. Configure the ACCSEL bits in the PMCTL1 register to select the FFT accelerator.

2. Factor N = VH, where $16 \leq V$ and $16 \leq H$.

3. Set (=1) the FFT_RST bit in the FFTCTL1 register and wait for a minimum of 4 CCLK cycles.

4. Program the FFTCTL2 register with:
   VDIM = V/16
   LOG2VDIM = Log2(V)
   HDIM = H/16

LOG2HDIM = Log2(H)
NOVER256 = VH/256
FFT_RPT = 1.

5. Program the FFTCTL1 register with: FFT_RST = 0
   FFT_EN = 1
   FFT_START = 1
   FFT_DMAEN = 1
   FFT_DEBUG = 0

For steps 6–15, see "N >= 512, No Repeat" above.

Using Debug Mode

The next sections show the steps required for reading and writing local memory in debug mode.

Write to Local Memory

1. Enable the FFT module using the PMCTL1 register.
2. Wait at least 4 CCLK cycles.
3. Clear the FFT_DMAEN bit in the FFTCTL1 register.
4. Set the FFT_DBG bit in the FFTCTL1 register.
5. Write first data to the FFTDDATA register.
6. Write address to the FFTDADDR register. Note the MSB Address bits determines which memory to write.
7. Wait at least 12 CCLK cycles before writing again FFTDDATA register.

Read from Local Memory

1. Enable FFT module using the PMCTL1 register.
2. Wait at least 4 CCLK cycles.

3. Clear the FFT_DMAEN bit in the FFTCTL1 register.
4. Set the FFT_DBG bit in the FFTCTL1 register.
5. Write address to the FFTDADDR register. The MSB address bits determine which memory to read.
6. Wait at least 20 CCLK cycles before writing data to FFTDDATA register.

Debug Features

The following sections describe the debugging features available on the accelerator.

Local Memory Access

Setting the FFT_DBG bit in the FFTCTL1 register puts the accelerator into debug mode and allows all memory locations (coefficient and data memory) to be read and written indirectly, using FFTDADDR and FFTDDATA registers. The MSB bits of the FFTDADDR register determines if the access is for the data or the coefficient memory.

Shadow Register

A shadow DMA status register, FFTSHDMASTAT, can read the DMA status register without modifying the status values.

FIR Accelerator

Finite Impulse Response (FIR) filters are used in a wide array of applications, and can be used in multi-rate processing in conjunction with an interpolator or decimator.

Features

This hardware module is capable of performing FIR filters without core intervention. This gives programs freedom to use the core to implement complex algorithms, effectively adding more bandwidth to the processor.

- FIR supports fixed point and IEEE floating point format
- Has four MAC units which operate in parallel
- Various rounding modes supported
- Single rate or multi-rate window processing
- Change the rates with decimation or interpolation mode
- Up to 32 filter channels available in TDM

Register Overview

The FIR accelerator registers are described below.

Power Management Control Register (PMCTL1). Used for FIR accelerator selection. Controls the clock power down to the module if not required.

Global Control Register (FIRCTL1). Configures the global parameters for the accelerator. These include number of channels, channel auto iterate, DMA enable, and accelerator enable.

Channel Control Register (FIRCTL2). The `FIRCTL2` register is used to configure the channel specific parameters such as filter TAP length, window size, sample rate conversion, up/down sampling and ratio.

DMA Status Register (FIRDMASTAT). Provides the status of the FIR accelerator operation. This information includes chain pointer loading, coefficient DMA, data preload DMA, processing in progress, window processing complete, and all channels processing complete.

MAC Status Register (FIRMACSTAT). Provides the status of MAC operation for all four multiply accumulators. In fixed-point mode, only the ARIx (adder result infinity) is used, all other bits are reserved.

Debug Control Register (FIRDEBUGCTL). Controls the debug mode operation of the accelerator.

Clocking

The FIR accelerator runs at the maximum speed of the peripheral clock frequency (f_{PCLK}).

Functional Description

Figure 7-3 shows the block diagram of the 1024-TAP FIR hardware accelerator. The accelerator consists of a 1024 word coefficient memory, a 1024 deep delay line for data, and four MAC units. The accelerator runs at the peripheral clock frequency (PCLK).

The FIR accelerator has following logical sub blocks.

1. A data path unit that consists of:
    a. A 1024 deep coefficient memory
    b. A 1024 deep delay line for the data
    c. Four 32-bit floating-point and fixed-point multiplier and adder units
    d. One 32-bit prefetch buffer to operate in a pipelined fashion
    e. One 32-bit buffer to hold previous partial sum
    f. One 32-bit buffer to hold the output

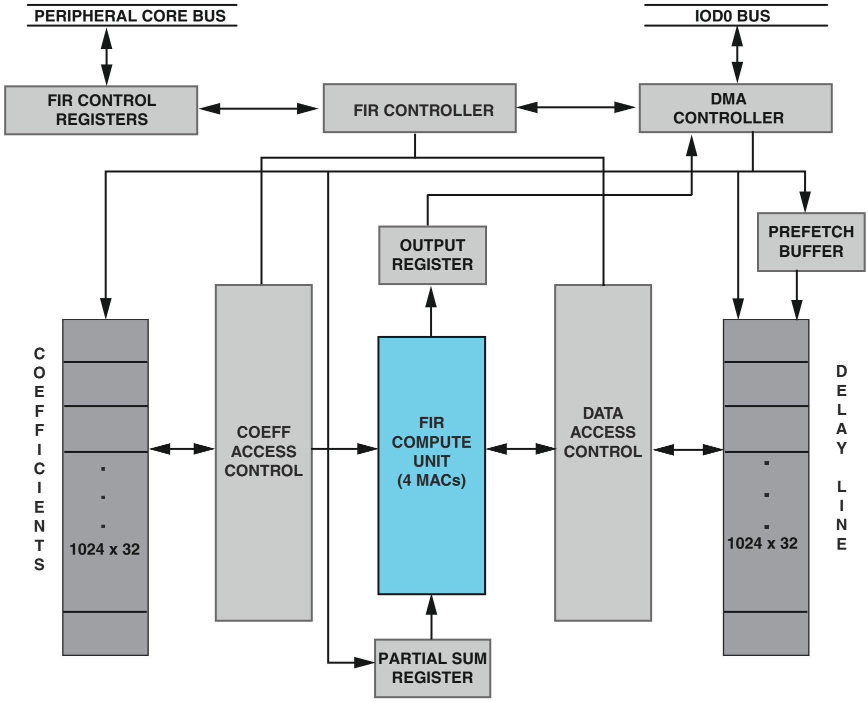

Figure 7-3. FIR Block Diagram

2. Configuration registers for the number of TAPs, number of channels, filter enable, interrupt control, DMA enable, up sample/down sample control, and ratios.
3. Core access interface for writing the DMA/filter configuration registers and reading the status register.
4. DMA bus interface for transferring data and/or coefficients to and from the accelerator.
5. DMA configuration registers including chain pointer, input, output, and coefficient registers.

Compute Block

The MAC unit, shown in Figure 7-4, has four multiply accumulators. They operate simultaneously on a single filter as described below.

- The MAC unit operates on the data and coefficient fetched from the data and coefficient RAMs.
- Each MAC can perform 32-bit floating-point or 32-bit fixed-point MAC operations.
- Floating-point format is IEEE compliant.
- Multiply and accumulation operation (addition) are pipelined.
- 32-bit floating-point MAC operation generates 32-bit multiply results.
- 32-bit fixed-point operation generates 80-bit results (64-bit result + 16 guard bits).

Partial Sum Register

The partial sum register is useful for floating-point multi-iteration mode. For a particular channel, the intermediate MAC result is written to the internal memory's output buffer. If the same channel is requested again, the partial result register is updated with the intermediate MAC result via DMA from the internal memory's output buffer and added to the current MAC result after each iteration. This process is repeated until all iterations are done (the entire soft filter length is processed).

Delay Line Memory

The accelerator has a 1024 TAP delay line to hold the data locally. The DMA controller fetches the data from internal memory and loads it into the delay line. Four read accesses can be made to the delay line simultaneously.

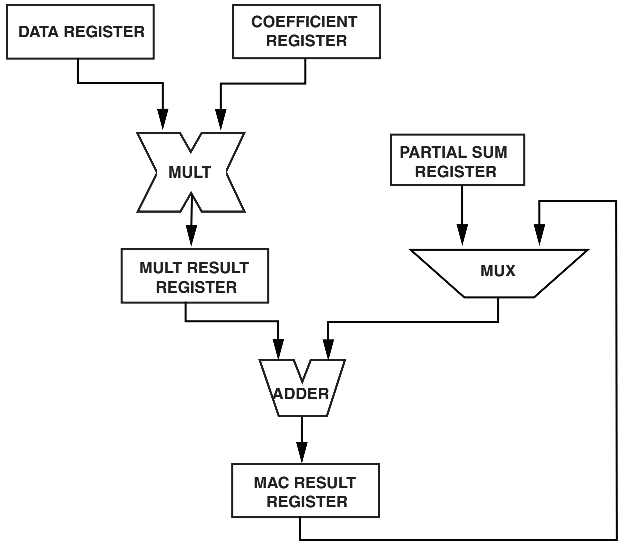

Figure 7-4. FIR MAC Unit

Coefficient Memory

The accelerator has a 1024 deep coefficient memory to store the coefficients. The DMA controller loads the coefficients from internal memory into coefficient memory. Four coefficients can be fetched from the coefficient memory simultaneously. If the soft filter length is more than 1024, processing is done in multi-iteration mode.

Pre Fetch Data Buffer

This buffer enables pipeline operation. 1 data sample is pre-fetch when the compute unit is operating on the delay-line corresponding to the current sample. The data pre-fetched in this buffer is later used to update the delay line for the next sample. This happens in parallel again, when the compute unit is not accessing the delay line in other words when it is adding the output from the four MACs and the partial sum register.

Table 7-3. Pipeline Operation for Window Size = 1

Cycles	1	2	3	4	5	6
Output DMA			N	N1	N2	N3
Compute		N	N1	N2	N3	
Input DMA	N	prefetch N1	prefetch N2	prefetch N3		

Processing Output

The accelerator uses all four MACs simultaneously to calculate one output sample as shown in Figure 7-5 and the following procedure.

1. The accelerator fetches four input data from the delay line and four corresponding coefficients from the coefficient memory and feeds them to the MAC units for multiply/accumulation.

2. The accelerator repeats the procedure with the next four input data and coefficients until all the TAPs complete. For an N TAP filter for example, this procedure is done N/4 times.

3. When all the TAPs complete, the accelerator adds the four MAC outputs together to the previous partial sum (if any) to calculate the final result.

4. Finally, that output sample is stored back in internal memory.

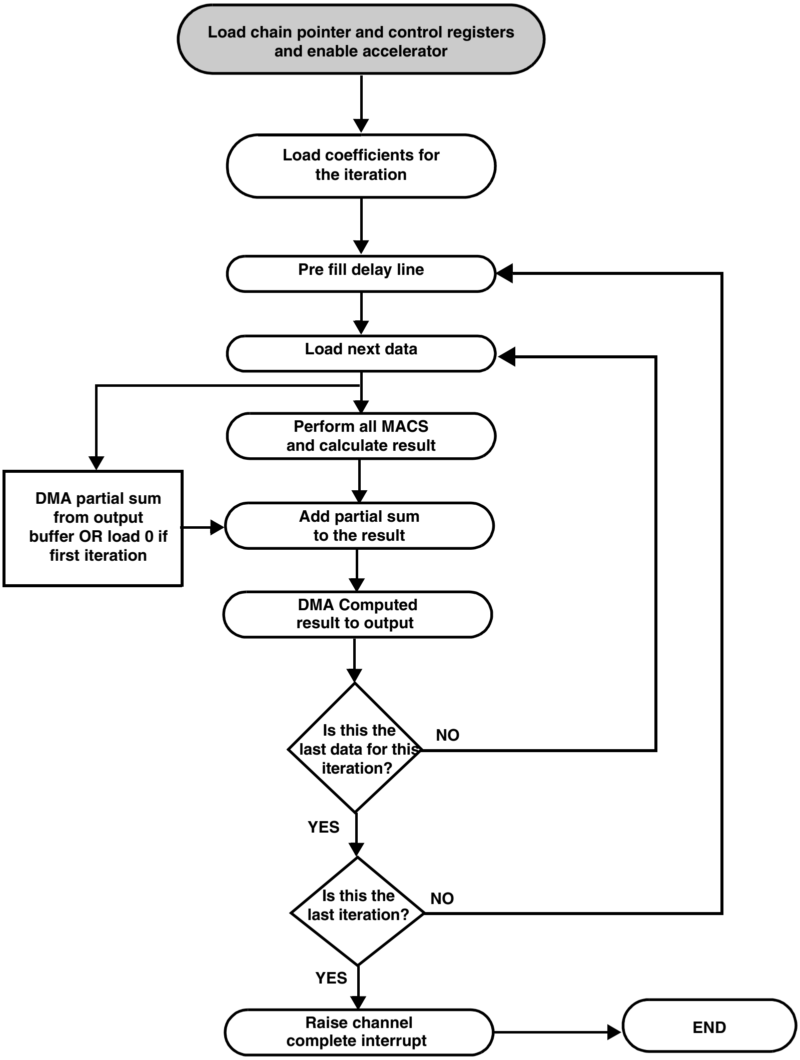

Figure 7-5. Multi-Iteration Filtering Flow

Internal Memory Storage

The following sections describe the storage format for the accelerator.

Coefficients and Input Buffer Storage

For any N TAP filter with coefficients:

```
C[i] i = 0,1,
…
N - 1
```

the coefficients should be stored in internal memory buffer in the order:

```
C[N - 1], C[N - 2]
…
C[1],C[0]
```

and the CI should point to `C(N - 1)`

Single Rate Input Filtering

The total size of the input buffer should at least be equal to N – 1 + W. If the input buffer that needs to be processed is:

```
x[n],x[n+1],x[n+2]
....
x[n+W-1]
```

it should be stored in the memory as

```
x[n-(N-1)], x[n-(N-2)]
....
x[n-1], x[n], x[n+1]
....
x[n+W-1]
```

and IIFIR should point to `x[n - (N - 1)]`

Decimation

Assuming M = decimation ratio, the total size of the input buffer should at least be equal to `N-1+WxM`. If the input buffer that needs to be processed is
`x[n],x[n+1],x[n+2]....x[n+WxM-1]`,

it should be stored in the memory as
`x[n-(N-1)], x[n-(N-2)]....x[n-1], x[n], x[n+1]....x[n+WxM-1]`

and IIFIR should point to `x[n-(N-1)]`.

Interpolation

Assuming L= interpolation ratio, the total size of the input buffer should at least be equal to Ceil `((N-1)/L)+W/L`.

If the input buffer that needs to be processed is
`x[n], x[n+1], x[n+2]....x[n+W/L-1], and K= Ceil((N-1)/L)`

it should be stored in the memory as:
`x[n-k], x[n-(K-1)], x[n-(K-2)]....x[n-1], x[n], x[n+1]....`
`x[n+W/L-1]`

and the IIFIR should point to `x[n-K]`.

Operating Modes

The FIR core performs a sum-of-products operation to compute the convolution sum. It supports single-rate, decimation, and interpolation functions.

Single Rate Processing

In a single-rate filter, the output result rate is equal to the input sample rate. The filter output Y(n) is computed according to following equation where N is the number of filter coefficients: c(i) i = 0,..., N – 1 are the filter coefficients and x(n) represents the input time-series.

$$Y(n) = \sum_{k=0}^{N-1} c(k) \times x(n-k)$$

Single Iteration

Results are computed in single iteration when the soft filter length is less than or equal to 1024.

Multi Iteration

Results are computed in multiple iterations when the soft filter length is greater than 1024 (for example, 2048 TAPs on a 1024 hard filter length). In this mode, the controller implements two iterations of 1024 TAPs. Note that if the soft filter length is not a multiple of the hard filter length the controller iterates until the soft filter length is satisfied.

Example: 550 taps on a 256 tap filter.

In this example, the FIR controller implements two iterations of 256 taps and one iteration of 38 taps.

Multi-iteration mode is not supported in fixed-point format.

Window Processing

Sample based processing mode is selected by configuring window size to 1. In this mode, one sample from a particular channel is processed through all the biquads of that channel and the final output sample is calculated.

In window based mode, multiple output samples (up to 1024) equal to the window size of that channel are calculated. After these calculations are complete, the accelerator begins processing the next channel. A

configurable window size parameter is provided to specify the length of the window.

Multi Rate Processing

Multi rate filters change the sampling rate of a signal—they convert the input samples of a signal to a different set of data that represents the same signal sampled at a different rate.

Decimation

A decimation filter provides a single output result for every M input samples, where M is the decimation ratio. Note that the output rate is 1/M'th of the input rate. The filter implementation exploits the low output sample rate by not starting a computation until a new set of M input samples is available.

In this mode, after low pass filtering (for anti aliasing), FIR logic discards the ratio – 1 samples of output data. For performance optimization, FIR logic skips the computation of output samples, which are discarded.

The input buffer size for decimation filters is $N - 1 + (W \times M)$ where:

- N is the number of taps
- W is the window size
- M is the decimation ratio

The window size (`WINDOW` bits) in the `FIRCTL2` register must be programmed with the number of output samples.

To start this mode, programs set the `FIR_RATIO` and `FIR_UPSAMP` bits in the `FIRCTL2` register (along with normal filter setting). Also the `TAPLEN` bits setting should be greater than or equal to `FIR_RATIO` bits setting for decimation filter.

Interpolation

An interpolator filter provides L output results for each new input sample, where L is the interpolation ratio. Note that the output rate is L times the input rate.

In this mode, according to the ratio specified in configuration register, FIR logic inserts L – 1 zeros between any two input samples (up-sampling) and then performs the interpolation (through the FIR filter).

Both up-sampling and down-sampling do not support multi iteration mode. Therefore, the filtering operation can only be done on up to 1024 TAPs and the ratio of up/down sampling can only be an integer value.

In an interpolation filter FIR logic inserts L – 1 zeros between each sample and the program has to make sure that these zeroes are fully shifted out of the delay line before moving on to the next channel. This puts a restriction on window size in terms of L – *the sample ratio* as shown below.

WINDOWSIZE = *n* × *SAMPLERATIO*
where n is the number of input samples.

The input buffer size is smallest integer greater than or equal to (N – 1 + W)/L for interpolation filters where:

- N is the number of taps
- W is the window size
- L is the interpolation ratio

To start the mode, programs configure the `FIR_RATIO` and `FIR_UPSAMP` bits (along with filter settings) in the `FIRCTL2` register.

Channel Processing

Figure 7-6 on page 7-43 shows the flow diagram for processing a single channel. Channels are processed in TDM format by setting the FIR_CH bits greater one. In the time slot corresponding to a particular channel, the corresponding TCB is loaded from internal memory.

1. The FIRCTL2 value is fetched from internal memory and is used to configure the filter parameters for that channel.
2. The accelerator fetches the coefficients using the CIFIR register as the pointer and loads them into coefficient memory.
3. The delay line is pre-filled using the IIFIR register as the pointer.
4. The accelerator calculates the first output and stores the result back into the output buffer using the OIFIR register as the pointer.
5. While calculating the output the accelerator fetches the next data in parallel. After one window of data is processed, the index registers in the internal memory TCB ares updated so that in the next time slot of the same channel, processing can be continued from where it stopped.
6. Processing moves to the next channel and repeats the procedure. If the soft filter length is more than the hard filter length, multiple iterations are done to process the window.

Floating-Point Data Format

The FIR accelerator treats data and coefficients in 32-bit floating-point format as the default functional mode.

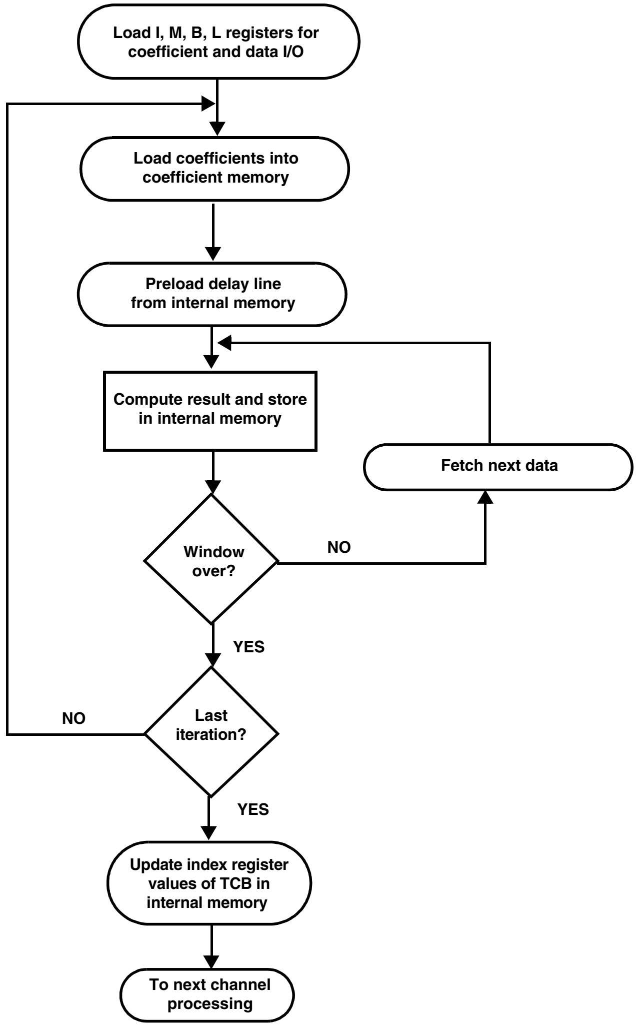

Figure 7-6. Single Channel Filtering Flow

Fixed-Point Data Format

In fixed-point mode, the 32-bit input data/coefficient is treated as fixed-point. A 32-bit fixed-point MAC operation generates an 80-bit result. Fixed-point data/coefficients can be unsigned integer, unsigned fractional and signed integer.

> In fixed point mode, the entire 80-bit result register is always written back in bursts of 3 × 32 bits. The first word is the LSW, the 2nd the MSW and the third word is a 16-bit overflow, the remaining 16-bits are padded with zeros. Therefore for fixed-point WINDOWSIZE = WINDOWSIZE × 3.
>
> If signed fractional format is used, the output needs to be scaled by 2 since the MAC does not the right shift to remove the redundant sign bit. A final routine needs to decimate the output buffer to the desired samples.

Multi iteration mode is not supported in this format. Therefore, the maximum TAP length is 1024.

Data Transfer

The FIR filter works exclusively through DMA.

DMA Access

The FIR accelerator has two DMA channels (accelerator input and output) to connect to the internal memory. The DMA controller fetches the data and coefficients from memory and stores the result.

Chain Pointer DMA

The DMA controller supports circular buffer chain pointer DMA. One transfer control block (TCB) needs to be configured for each channel. The TCB contains:

- A control register value to configure the filter parameters (such as filter tap length, window size, sample rate conversion settings) for each channel
- DMA parameter register values for the input data (delay line)
- DMA parameter register values for coefficient load
- DMA parameter register values for output data

Intermediate results in multi-iteration mode are saved in the output buffer

As shown in “FIR Accelerator TCB” on page 3-17 and Figure 7-7, the accelerator loads the TCB into its internal registers and uses these values to fetch coefficients and data and to store results. After processing a window of data for any channel, the accelerator writes back the appropriate values to the IIFIR and OIFIR fields of the TCB in memory, so that data processing can begin from where it left off during the next time slot of that channel.

The write back value for input buffer is:

- IIFIR + W for single rate filtering.
- IIFIR + W × M for decimation (M = decimation ratio).
- IIFIR + W/L for interpolation (L = interpolation ratio).
- The write back values for output buffer in floating point mode is: OIFIR + W.
- The write back values for output buffer in fixed point mode is: OIFIR + 3 × W.

The FIRCTL2 register is part of the FIR TCB. This allows programming individual FIR channels with different control attributes.

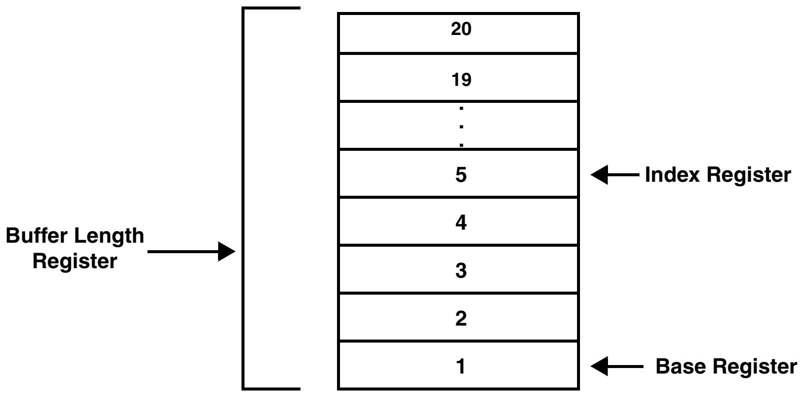

Figure 7-7. Circular Buffer Addressing

Interrupts

Table 7-4 provides an overview of FIR interrupts.

Table 7-4. FIR Interrupt Overview

Default Programmable Interrupt	Sources	Masking	Service
ACC0I/ACC1I not connected by default	Input DMA complete Output DMA complete Window complete All channels complete	N/A	ROC from FIRDMASTAT + RTI instruction
	MAC IEEE floating-point exceptions MAC fixed-point Overflow		ROC from FIRMACSTAT + RTI instruction

Sources

The FIR module drives two interrupt signals, `ACC0I` for the DMA status and `ACC1I` for the MAC status. The FIR module generates interrupts as described in the following sections.

Window Complete

This interrupt is generated at the end of each channel when all the output samples are calculated corresponding to a window and updated index values are written back.

All Channels Complete

This interrupt is generated when all the channels are complete or when one iteration of time slots completes. Note that the interrupt follows the access completion rule, where the interrupt is generated when all data are written back to internal memory.

Chained DMA

For chained DMA, if the `PCI` bit is cleared (= 0), the DMA complete interrupt is generated only after the entire chained DMA access is complete. If the `PCI` bit is set (= 1), then a DMA interrupt is generated for each TCB.

MAC Status

A MAC status interrupt is generated under these conditions

- Multiplier result zero – Set if Multiplier result is zero
- Multiplier Result Infinity – Set if Multiplier result is Infinity
- Multiply Invalid – Set if Multiply operation is Invalid
- Adder result zero – Set if Adder result is zero
- Adder result infinity – Set if Adder result is infinity
- Adder invalid – Set if Addition is invalid
- Adder overflow – for fixed-point operation

Masking

The ACC0I and ACC1I signals are not routed by default to programmable interrupts. To service the ACCxI, unmask (set = 1) any programmable interrupt bit in the IMASK/LIRPTL register

Service

When a DMA interrupt occurs, programs can find whether the input or output DMA interrupt occurred by reading the DMA status register (FIRDMASTAT). The DMA interrupt status bits are sticky and are cleared when the DMA status register is read. When a MAC status interrupt occurs, programs can find this by reading the MAC status register (FIRMACSTAT). The MAC interrupt status bits are sticky and are cleared by a read.

The status interrupt sources are derived from the FIRMACSTAT register. If the status interrupt occurs as a result of the last set of MAC operations of a processing iteration corresponding to a particular channel, the interrupt is generated continuously and cannot be stopped, even after disabling the accelerator. The interrupt can only be stopped by another processing iteration that results in a non-zero or valid multiply/add result. However, in this situation it is difficult to isolate whether the interrupt corresponds to the previous processing iteration or that of the current one. This makes the use of status interrupts impractical.

An alternate way is to poll status bits of the FIRMACSTAT register inside the DMA interrupt service routine. However, the behavior of the status bits, as described below, should be kept in mind. The status bits in the FIRMACSTAT registers are sticky. Once a status bit is set, it gets cleared only when the FIRMACSTAT register is read and the previous set of MAC operations resulted in a non-zero/valid output. Therefore, if the last set of MAC operations of a particular processing iteration results in a zero/non-valid output, the corresponding status bit won't be cleared, even after reading the FIRMACSTAT register. To avoid a false indication in the next processing iteration, it is necessary to ensure that all the status bits are cleared after the current iteration finishes.

The solution is to read the FIRMACSTAT register twice inside the DMA interrupt service routine. The first read is used to identify which status bits are set. The second read is used to discover if the status bit was set because of the last set of MAC operations. If the status bit was not set because of the last set of MAC operations, it provides a zero result.

Otherwise, the bit was set because of the last set of MAC operations. In that case, the status bit must be cleared by performing a simple dummy FIR processing iteration (tap length = 4 and window size = 1) by choosing the appropriate coefficients and input buffer and reading the FIRMACSTAT register after the processing is complete. For more information, see "FIR MAC Status Register (FIRMACSTAT)" on page A-82.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

FIR Accelerator Effect Latency

After the FIR registers are configured the effect latency is 1.5 PCLK cycles minimum and 2 PCLK cycles maximum. Writes to the PMCTL1 register have an effect latency of two PCLK cycles. Wait for at least four CCLK cycles after selecting an accelerator before accessing any of its registers.

FIR Throughput

Accelerator input and output channels are used to connect to internal memory. Data throughput is one 32-bit data word per peripheral clock cycle for writes to memory, provided there are no conflicts. Read throughput from memory, throughput is one 32-bit data word per two peripheral clock cycles.

The following information describes the performance of the FIR accelerator in processor cycles.

Total number of PCLK cycles for single rate filtering N<=1024 is:

$$(\text{TCB load} + 4 \times N + W(N/4 + 2)) \times C$$

and the total number of PCLK cycles for decimation is:

$$(\text{TCB load} + 4 \times N + W(N/4 + 2) + (W - 1) \times (M - 1) \times 7) \times C$$ where:

- N – Number of taps
- W – Window size
- C – Number of channels
- TCB load = 49 PCLK cycles
- 4 × N – Number of cycles for loading coefficients an data considering two cycles for read
- N/4 + 2 – FIR compute cycles considering four pipelined MACs
- M – Decimation ratio

Programming Model

The following steps should be used when programming the accelerator. Enable the FIR accelerator by setting accelerator select bits (ACCSEL in the PMCTL1 register) to 00.

Single Channel Processing

1. Create input, coefficient, and output buffers in internal memory.

   For input and coefficient buffer storage format see "Coefficients and Input Buffer Storage" on page 7-37.

2. Create the TCBs in internal memory. Each TCB corresponds to a particular channel.

   TCBs hold the FIRCTL2 register which allows programming the window size and tap size along with up or down sample enable, sample rate conversion enable, and the conversion ratio for decimation and interpolation filters.

3. Configure the index, modifier, length entries in the TCBs to point to the corresponding channels' data buffer, coefficient buffer, and output data buffer.

   The output index register should always point to the start of the output buffer. However, the input index register's value should be initialized based on the explanation provided in "Coefficients and Input Buffer Storage" on page 7-37.

4. The core configures the FIRCTL1 register with the number of channels (one channel), fixed- or floating-point format.

5. Set the enable bit to start accelerator operation in the modes configured (in FIRCTL1 and FIRCTL2 registers) by loading the first channels' TCB. Once the first channel window is calculated, the input and output index registers are written back to internal memory corresponding to the first channel. Once the write back is complete the accelerator moves into idle.

Multichannel Processing

Figure 7-8 on page 7-53 shows the diagram for multichannel filtering. Multiple channels are processed in a time division multiplexed (TDM) format. After completing all the channels, the accelerator can either repeat the slots or wait for core intervention.

For multichannel filtering, use the following steps.

1. Program the number of channels in the FIRCTL1 register using the FIR_NCH bits (5–1).

2. Configure the TCBs in internal memory with one channel's TCB pointing to the next channel's TCB.

3. Write the first TCB value into the CPFIR register and enable the accelerator.

   The accelerator fetches the first channel's TCB and, using it as pointer, pre-fills the delay line and coefficient memory and loads the FIRCTL2 register to configure the filter parameters corresponding to that channel.

   The accelerator then calculates output samples corresponding to one Window and stores the data back in internal memory.

   At the end of the Window the accelerator updates the IIFIR and OIFIR registers in the TCB of internal memory and moves to the next channel.

   When all the channels are finished and the auto channel iterate (CAI, bit 9) is set, the accelerator processes the first channel again and iterates through the channels. If the CAI bit is cleared, the accelerator waits for core intervention.

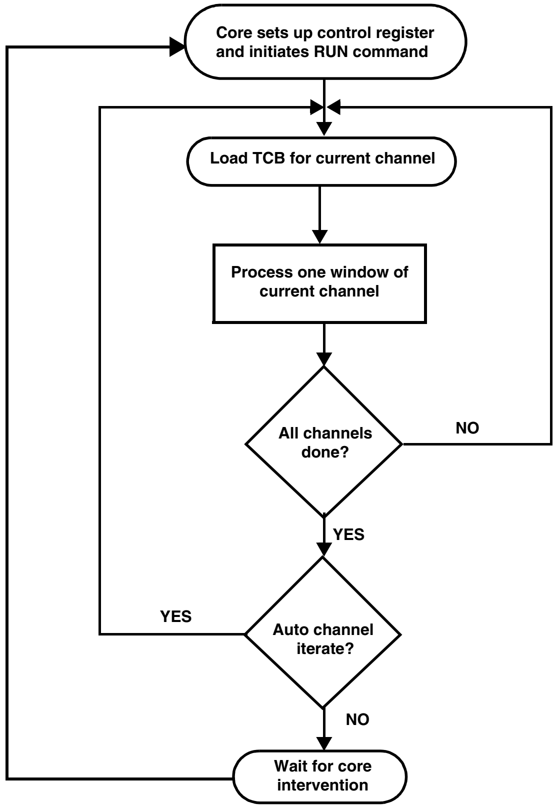

Figure 7-8. Wait for Core Intervention => Idle (if CAI bit = 0)

Dynamic Coefficient Processing Notes

1. The dynamic update of the coefficients may be useful for the FIR accelerator. The reason is that the FIR accelerator re-loads the coefficients for each iteration (if the CAI bit is set) before the start of processing of each channel.

2. The dynamic coefficient update should be possible for single iteration mode (tap length <=1024) by making sure that the new coefficients are updated after the accelerator loads the coefficients for current processing and before the next processing starts. The expression for the maximum time available for the coefficient memory update should be equal to $2 \times N + W(\text{ceil}(N/4) + 2)$ PCLK cycles.

3. For multi-iteration mode dynamic updates are not supported. Programs must finish current processing, disable the accelerator, update the coefficients, and re-enable the accelerator.

Debug Mode

The next sections show the steps required for reading and writing local memory in debug mode.

Write to Local Memory

1. Enable the FIR module using the PMCTL1 register.

2. Wait at least 4 CCLK cycles.

3. Clear the FIR_DMAEN bit in the FIRCTL1 register.

4. Set FIR_DBGMODE, FIR_DBGMEM and FIR_HLD bits in FIRDEBUGCTL register.

5. Set the FIR_ADRINC bit in FIRDEBUGCTL register for address auto increment.

6. Write start address to the FIRDBGADDR register. Note if bit 11 is set, coefficient memory is selected.
7. Wait at least 4 CCLK cycles.
8. Write data to the FIRDBGWRDATA register.

Read from Local Memory

1. Enable FIR module using the PMCTL1 register.
2. Wait at least 4 CCLK cycles.
3. Clear the FIR_DMAEN bit in the FIRCTL1 register.
4. Set FIR_DBGMODE, FIR_DBGMEM and FIR_HLD bits in FIRDEBUGCTL register.
5. Set the FIR_ADRINC bit in FIRDEBUGCTL register for address auto increment.
6. Write start address to the FIRDBGADDR register. Note if bit 11 is set, coefficient memory is selected.
7. Wait at least 4 CCLK cycles.
8. Read data from the FIRDBGRDDATA register.

Single Step Mode

Single step mode can be used for debug purposes. An additional debug register is used in this mode.

1. Enable stop DMA during breakpoint hit in the emulator settings.
2. Clear the FIR_HLD bit and enable FIR_DBGMODE and FIR_RUN bits in FIRDEBUGCTL register.

3. Program FIR module according to the application.
4. In single step each iteration is updated in the emulator session.

FIR Programming Example

An application needs FIR filtering of six channels of data. The first four channels require 256 TAP filtering and the last two channels require 1024 TAP filtering. The window size for all the channels is 128.

1. Create a circular data buffer in internal memory for each channel.

   The buffer should be large enough to avoid overwriting data before being processed by the accelerator. Ideally, the input buffer size for a channel is *tap length* + *window size* – 1 for that channel. The 256 coefficients of each of the first four channels and the 1024 coefficients each of the last two channels are also configured in internal memory buffers. The output buffer size is equal to the window size.

2. Create six TCBs in internal memory with each channel's chain pointer (CP) entry pointing to the next channel's and the sixth channel's CP entry pointing back to the first's in a circular fashion.
3. Configure the `FIRCTL2` register for the first four channels' TCBs to 256 TAPs and a window size of 128, and the next two channels for 1024 TAPs and a window size of 128, respectively.
4. Configure the index, modifier, length entries in the TCBs to point to the corresponding channel's data buffer, coefficient buffer, and output data buffer. The location of the first channel's TCB is written to the `CPFIR` register. The `FIRCTL1` register is then programmed with an `FIR_CH` value that corresponds to six channels.

a. The accelerator iterates through six channels once and then waits for core intervention, (the FIR_CAI bit is not set, the DMA is enabled, and the FIR_EN bit is set).

b. The accelerator then loads the first channel's TCB then loads the coefficient and data and processes one window.

c. After saving the index values to memory the accelerator moves to the next channel.

d. After all six channels are complete the accelerator halts and waits for core intervention.

Computing FIR Output, Tap Length > Than 4096

With little core intervention, the FIR accelerator can as well be used to calculate output for tap length greater than 4096 taps. The section shows how it can be done with an example of 8192 taps. Transfer function of an 8192 FIR filter can be divided into two 4096 FIR filters as shown below.

$$H(Z) = b_0 + b_1 Z^{-1} + b_2 Z^{-2} + \ldots\ldots\ldots\ldots b_{4095} Z^{-4095} + b_{4096} Z^{-4096} b_{4097} Z^{-4097} + \ldots\ldots\ldots b_{8191} Z^{-8191}$$

$$= b_0 + b_1 Z^{-1} + b_2 Z^{-2} + \ldots\ldots\ldots\ldots b_{4095} Z^{-4095} + Z^{-4096}(b_{4096} + b_{4097} + \ldots\ldots\ldots b_{8191} Z^{-4096})$$

Filter coefficients of an 8192 tap filter therefore need to be divided among two 4096 tap FIR filters.

Filter 1

Coefficients = b0, b1, b2,……, b4095

Input data = x[n], x [n-1],………x[n-4095]

Filter 2

Coefficients=b4096, b4097,………….,b8191

Input data = x[n-4096], x[n-4097],……x[n-8191]

The accelerator can be used in two channel mode where channel 1 operates on x[n]…x[n-4095] input data with the filter coefficients of filter 1 and channel 2 operates on x[n-4096]…x[n-8191] with the filter coefficients of filter 2.

Once both the channels are processed, the partial sum output of both the channels can to be added together to get the final output. The following programming steps are needed to implement this approach (tap length = TAPS = 8192, window size = WINDOW).

1. Create a circular input data buffer in internal memory (IBUF). The buffer should be large enough to avoid overwriting data before being processed by the accelerator. Ideally, the input buffer size for a channel is TAPS + WINDOW–1.
2. Create a coefficient buffer of size TAPS (8192) (CBUF).
3. Create one output buffer of size WINDOW (OBUF) and another temporary output buffer (OBUF1) to store the partial sum.
4. Create two TCBs in internal memory with first TCB chained to the second and second one chained to the first in circular manner.
    a. The CIFIR field of the first TCB should point to the start address of the coefficient buffer (CBUF) and that of the second TCB should point to 4096 offset from the start of the coefficient buffer (CBUF + 4096).
    b. The OBFIR and OIFIR field of the first TCB should point to the start address of OBUF and that of the second TCB should point to the start address of OBUF1.

   c. The IIFIR field of the first TCB should point to the start address of IBUF and that of the second TCB should point to 4096 offset from the start address of IBUF.

   d. The FIRCTL2 field of both the TCB should be configured for tap length = TAP/2 = 4096 and window size = WINDOW.

5. Initialize the CPFIR register pointing to the first TCB.

6. Program the FIRCTL1 register to initiate the accelerator processing now by setting the FIR_EN, FIR_DMAEN bits and no of channels configured as 2

7. Wait for the FIR all channel done interrupt to occur and inside the ISR, add the partial sum results using core from both the output buffers (OBUF and OBUF1) to get the final output. To save memory, the contents of the buffer OBUF can as well be replaced by the final output result.

Debug Features

The following sections provide information of debugging the FIR accelerator.

Local Memory Access

The contents of FIR delay line and coefficient memories are made observable for debug by setting the `FIR_DBGMODE`/`FIR_DBGMEM` and `FIR_HLD` bits in the `FIRDEBUGCTL` control register. The debug address register (`FIR_DBGADDR`) and two data registers are provided for debug operations. Bit 11 of the `DBGADDR` register selects coefficient memory if set (=1) and selects delay line memory in cleared (=0).

In the debug mode, the read data register (DBGMEMRDDAT) returns the contents of the memory location pointed to by the address register. Data can be written into any memory location using DBGMEMWRDAT register writes. If the address auto increment bit (FIR_ADRINC) is set, the address register auto increments on DBGMEMWRDAT writes and DBGMEMRDDAT reads. During auto increment, the FIR_DBGADDR register cannot cross the data memory/coefficient memory boundary.

Single Step Mode

Programs can single step through the MAC operations and observe the memory contents after each step. The FIR_DBGMODE/FIR_HLD and FIR_RUN bits control the FIR MAC units.

Emulation Considerations

In FIR debug mode, the DMA operations are not observable.

IIR Accelerator

The ADSP-214xx processors have an IIR filter accelerator implemented in hardware, that reduces the processing load on the core, freeing it up for other tasks.

Features

The accelerator supports a maximum of 24 channels. There is support for up to 12 cascaded bi-quads per channel. This means that the accelerator locally stores all the biquad coefficients of 24 channels. Window size can be configured from 1 (sample based) to 1024.

- IIR supports IEEE floating point format 32/40-bit
- Various rounding modes supported

- Sample based or window based processing
- Up to 12 cascaded biquads per channel
- Up to 24 filter channels available in TDM
- Allows Biquad save state storage

Register Overview

The following sections provide information on the IIR accelerator control and status registers.

Power Management Control Register (PMCTL1). Used for IIR accelerator selection. Controls the clock power down to the module if not required.

Global Control (IIRCTL1). Configures the global parameters for the accelerator. These include number of channels, channel auto iterate, DMA enable, and accelerator enable.

Channel Control (IIRCTL2). The `IIRCTL2` register is used to configure the channel specific parameters. These include number of biquads and window size.

DMA Status (IIRDMASTAT). Provides the status of accelerator operation including chain pointer loading, coefficient DMA, processing progress, window complete and all channels complete.

MAC Status (IIRMACSTAT). Provides the status of the MAC operations.

Debug Mode Control (IIRDEBUGCTL). Controls the debug mode operation of the accelerator.

Clocking

The IIR accelerator runs at the maximum speed of the peripheral clock (f_{PCLK}).

Functional Description

Figure 7-9 shows the block diagram of the IIR hardware accelerator. The accelerator has a coefficient memory size of 1440 × 40 bits (12 biquads × 12 channels × 5 coeffs), a data memory size of 576 × 40 bits (12 biquads × 12 channels × 2 states) and one MAC unit with an input data buffer to supply data to the MAC.

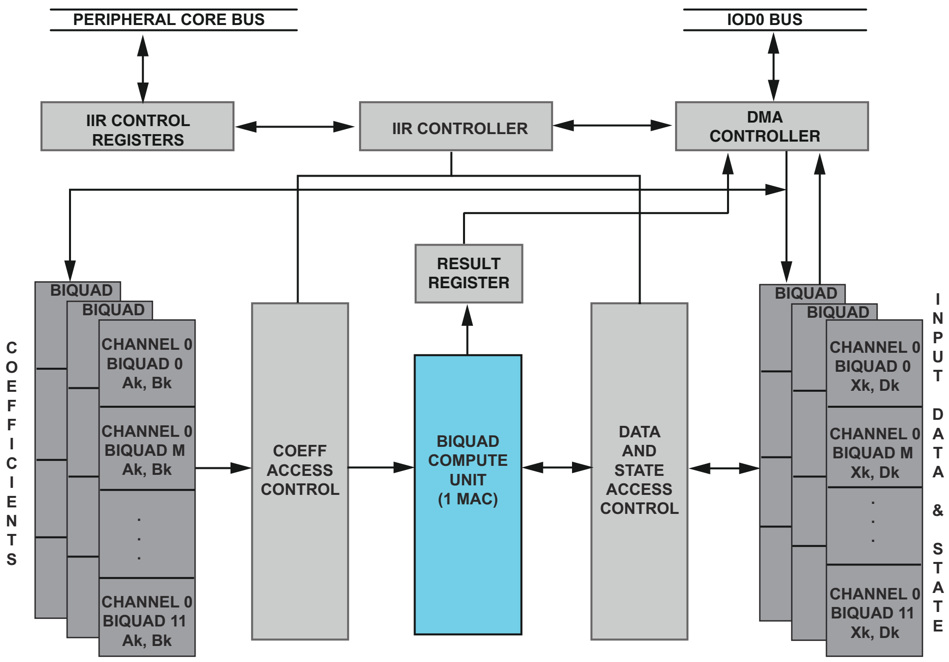

Figure 7-9. IIR Accelerator Block Diagram

The IIR accelerator is implemented using Transposed Direct Form II biquad which has less coefficient sensitivity. Figure 7-10 shows the signal flow graph for the biquad structure.

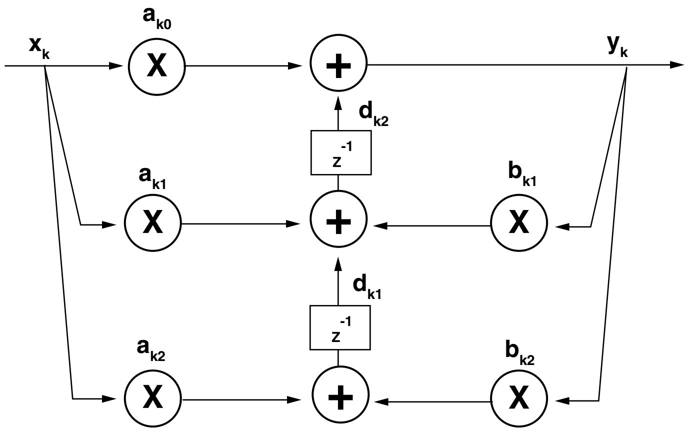

Figure 7-10. Transposed Direct Form II Biquad

The accelerator has the following logical sub blocks.

- A data path unit with the following elements:
  - 32/40-bit coefficient memory (Ak, Bk) for storing biquad coefficients
  - 32/40-bit input data (Xk) and state (Dk)
  - One 40/32-bit floating-point multiplier and adder (MAC) unit
  - An input data buffer to efficiently supply data to MAC
  - One 40-bit result register to hold result of biquad

- Configuration registers for controlling various parameters such as the number of biquads, the number of channels, interrupt control, and DMA control
- A core access interface for writing the DMA/filter configuration registers and for reading the status registers
- A DMA bus interface for transferring data to and from the accelerator. This interface is also used to preload the coefficients (Ak, Bk) and state (Dk) at start up.
- DMA configuration registers for the transfer of input data, output data and coefficients

Multiply and Accumulate (MAC) Unit

The MAC unit shown in Figure 7-11 has a pipelined multiplier and accumulator unit that operates on the data and coefficient fetched from the data and coefficient memory. The MAC can perform either 32-bit floating-point or 40-bit floating-point MAC operations. 32-bit floating-point operations generate 32-bit results and 40-bit floating-point operations generate 40-bit results.

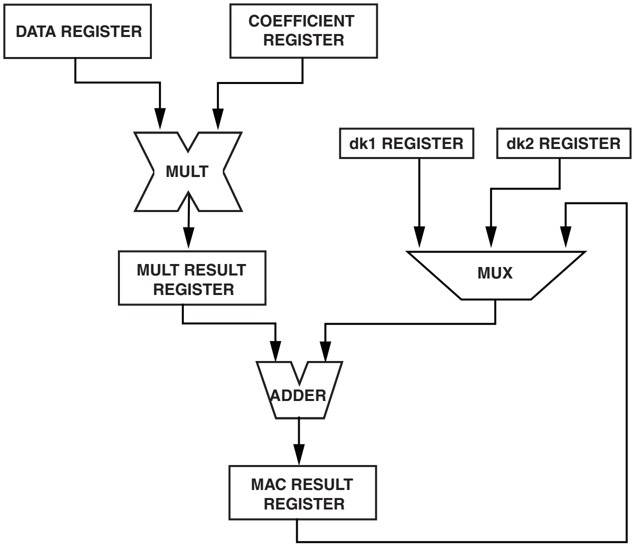

Figure 7-11. IIR MAC Unit

Input Data and Biquad State

The size of data memory is 576 × 40 bits and is used to hold the dk1 and dk2 state of all the biquads locally. The DMA controller fetches the sample data from internal memory and calculates the output as well as the dk1 and dk2 values for each biquad and stores them in local data memory.

Coefficient Memory

The size of coefficient memory is 1440 × 40 bits and is used to store all the coefficients of all the biquads. At start-up, DMA loads the coefficients from internal memory into local coefficient memory.

Internal Memory Storage

This section describes the required storage model for the IIR accelerator.

Coefficient Memory Storage

Coefficients and Dk values for a particular biquad BQD[k] should be stored in internal memory in the order Ak0, Ak1, Bk1, Ak2, Bk2, Dk2, Dk1.

The naming convention for the filter coefficients used here is different from the one used in MATLAB. The following conversion should be used when using MATLAB generated coefficients: (Akx = bx and Bkx = –ax).

In other words, the coefficients for each biquad should be stored in the order:

```
b0, b1, -a1, b2, -a2, dk2, dk1
```

For N biquad stages, the order of coefficients should be as follows:

```
b01, b11, -a11, b21, -a21, dk21, dk11,
b02, b12, -a12, b22, -a22, dk22, dk12
,......
b0N, b1N, -a1N, b2N, -a2N, dk2N, dk1N.
```

where bxN and axN are the coefficients ([b, a]) for the Nth biquad stage.

Operating Modes

The accelerator can be operated in the following modes.

Window Processing

Sample based processing mode is selected by configuring window size to 1. In this mode, one sample from a particular channel is processed through all the biquads of that channel and the final output sample is calculated.

In window-based mode, multiple output samples (up to 1024) equal to the window size of that channel are calculated. After these calculations are complete, the accelerator begins processing the next channel. A configurable window size parameter is provided to specify the length of the window.

40-Bit Floating-Point Mode

In 40-bit floating-point mode, the input data/coefficient is treated as a 40-bit floating-point number. 40-bit floating-point MAC operations generate 40-bit results. This mode can be selected by setting bit 12 of the `IIRCTL1` register.

Since the DMA bus width is 32 bits, in 40-bit mode the IIR accelerator performs two packed 32-bit accesses to the memory to fetch one 40-bit input or coefficient data, or to store one 40-bit output word. The first 32-bit word provides the lower 32 bits and the 8 LSBs of the second 32-bit word provides rest of the upper 8 bits of the a complete 40-bit word. Figure 7-12 shows the 32-40 bit packing used by accelerator.

Overheads might be required to pack the input 40-bit data into the format acceptable by the IIR accelerator and for unpacking the output of accelerator to the format acceptable by the rest of the application.

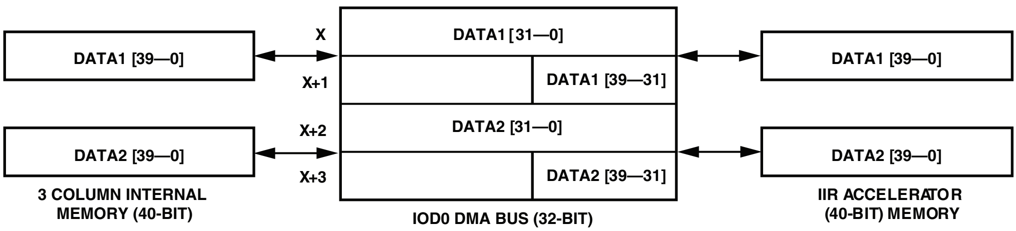

Figure 7-12. 32-Bit To 40-Bit Packing

Save Biquad State Mode

The IIR_SS bit (IIRCTL1 register) completely stores the current biquad states in local memory (writes all the DK1 and DK2 states back into the internal memory states). This is useful in applications that require fast switching to another high priority accelerator task—a required IIR to FIR processing transition for example. After resuming these states can be reloaded and IIR processing can be continued. Note that the DMA status is automatically stored after each iteration.

The save state operation cannot be stopped after it starts, even by clearing the IIR_EN or IIR_DMAEN bits. Although the bits would clear on the core side, settings take effect only after the save state operation completes. Therefore, before trying to disable the IIR accelerator, you must poll the corresponding status bits in the IIRDMASTAT register (see Table A-53) to ensure the save state operation completed successfully. The following expressions provide the latency due to the save state operation, assuming no higher priority DMA is ON:

For 32-bit mode: $14*N + ((8*M)+2)*N)$

For 40-bit mode: $14*N + ((15*M)+2)*N)$

where N is the number of channels and M is the number of biquads per channel.

Write access to any of the IIR accelerator registers loaded by chaining (see Table 3-20) is not allowed while the save state operation is in progress. Attempted writes to these registers might result in the blocking of IOP core reads until the save state operation completes.

Data Transfers

The IIR filter works exclusively through DMA.

DMA Access

The IIR accelerator has two DMA channels (accelerator input and output) to connect to the internal memory. The DMA controller fetches the data and coefficients from memory and stores the result.

Chain Pointer DMA

The DMA controller supports circular buffer chain pointer DMA. One transfer control block (TCB) needs to be configured for each channel. The TCB contains:

- A control register value to configure the filter parameters (such as number of biquads, window size) for each channel
- DMA parameter register values for the input data
- DMA parameter register values for coefficient load
- DMA parameter register values for output data

The chain pointer (CPIIR) field of the last channel's TCB should point to the first channel's TCB. This is so that when the IIR accelerator is enabled, 1) it first loads the coefficients (Ak, Bk) and state variables (Dk) for all the channels in to it's local coefficient memory and 2) it loops back to first channel again to start fetching the input data for processing.

As shown in "IIR Accelerator TCB" on page 3-18 and Figure 7-13, the accelerator loads the TCB into its internal registers and uses these values to fetch coefficients and data and to store results. After processing a window of data for any channel, the accelerator writes back the IIRII (input index register) and IIROI (output index register) values to the TCB in memory, so that data processing can begin from where it left off during the next time slot of that channel.

For 32-bit mode, the write back values for the index registers is equal to IIRII + W and IIROI + W.

For 40-bit mode, the write back values are:
`IIRII` + 2 × W and `IIROI` + 2 × W.

Accelerator input and output channels connect to internal memory.

The `IIRCTL2` register is part of the IIR TCB. This allows to program individual IIR channels having different control attributes.

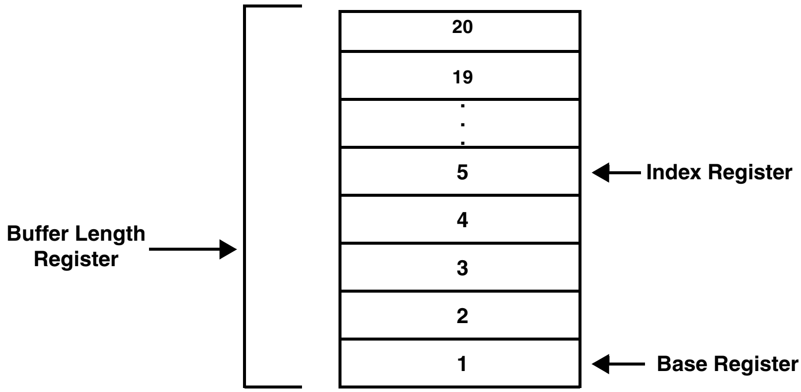

Figure 7-13. Circular Buffer Addressing

Interrupts

Table 7-5 provides an overview of IIR interrupts.

Table 7-5. IIR Interrupt Overview

Default Programmable Interrupt	Sources	Masking	Service
ACC0I/ACC1I not connected by default	Input DMA complete Output DMA complete	N/A	ROC from IIRDMASTAT + RTI instruction
	MAC IEEE floating point exceptions		ROC from IIRMACSTAT + RTI instruction

Sources

The IIR module drives two interrupt signals, ACC0I for the DMA status and ACC1I for the MAC status. The IIR module generates interrupts as described in the following sections.

Window Complete

This interrupt is generated at the end of each channel when all the output samples are calculated corresponding to a window and updated index values are written back.

All Channels Complete

This interrupt is generated when all the channels are complete or when one iteration of time slots completes. The interrupt follows the access completion rule, where the interrupt is generated when all data are written back to internal memory.

Chained DMA

For chained DMA, if the PCI bit is cleared (= 0), the DMA complete interrupt is generated only after the entire chained DMA access is complete. If the PCI bit is set (= 1), then a DMA interrupt is generated for each TCB.

MAC Status

A MAC status interrupt is generated under these conditions

- Multiplier result zero – Set if Multiplier result is zero
- Multiplier Result Infinity – Set if Multiplier result is Infinity
- Multiply Invalid – Set if Multiply operation is Invalid
- Adder result zero – Set if Adder result is zero

- Adder result infinity – Set if Adder result is infinity
- Adder invalid – Set if Addition is invalid

Masking

The ACC0I and ACC1I signals are not routed by default to programmable interrupts. To service the ACCxI, unmask (set = 1) any programmable interrupt bit in the IMASK/LIRPTL register

Service

When a DMA interrupt occurs, programs can find whether the input or output DMA interrupt occurred by reading the DMA status register (IIRDMASTAT). The DMA interrupt status bits are sticky and are cleared when the DMA status register is read. When a MAC status interrupt occurs, programs can find this by reading the MAC status register (IIRMACSTAT). The MAC interrupt status bits are sticky and are cleared by a read.

The status interrupt sources are derived from the IIRMACSTAT register. If the status interrupt occurs as a result of the last set of MAC operations of a processing iteration corresponding to a particular channel, the interrupt is generated continuously and cannot be stopped, even after disabling the accelerator. The interrupt can only be stopped by another processing iteration that results in a non-zero or valid multiply/add result. However, in this situation it is difficult to isolate whether the interrupt corresponds to the previous processing iteration or that of the current one. This makes the use of status interrupts impractical.

An alternate way is to poll status bits of the IIRMACSTAT register inside the DMA interrupt service routine. However, the behavior of the status bits, as described below, should be kept in mind. The status bits in the IIRMACSTAT registers are sticky. Once a status bit is set, it gets cleared only when the IIRMACSTAT register is read and the previous set of MAC operations resulted in a non-zero/valid output. Therefore, if the last set of MAC operations of a particular processing iteration results in a zero/non-valid output, the corresponding status bit won't be cleared, even after reading the IIRMACSTAT register. To avoid a false indication in the next processing iteration, it is necessary to ensure that all the status bits are cleared after the current iteration finishes.

The solution is to read the IIRMACSTAT register twice inside the DMA interrupt service routine. The first read is used to identify which status bits are set. The second read is used to discover if the status bit was set because of the last set of MAC operations. If the status bit was not set because of the last set of MAC operations, it provides a zero result.

Otherwise, the bit was set because of the last set of MAC operations. In that case, the status bit must be cleared by performing a simple dummy IIR processing iteration (biquads = 1 and window size = 1) by choosing the appropriate coefficients and input buffer and reading the IIRMACSTAT register after the processing is complete. For more information, see "IIR MAC Status Register (IIRMACSTAT)" on page A-89.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

IIR Accelerator Effect Latency

After the IIR registers are configured the effect latency is 1.5 PCLK cycles minimum and 2 PCLK cycles maximum. Writes to the PMCTL1 register have an effect latency of two PCLK cycles. Wait for at least four CCLK cycles after selecting another accelerator before accessing any of its registers.

IIR Throughput

Data throughput is one 32-bit data word per peripheral clock cycle for writes to memory, provided there are no conflicts. Read throughput from memory, throughput is one 32-bit data word per two peripheral clock cycles.

IIR throughput is calculated as follows:

Total number of peripheral clock cycles = (TCB load + 5 × B × W) × C where:

- B = number of bi-quads
- W = Window size
- C = number of channels
- TCB load = 36 `PCLK` cycles for 32-bit mode and 38 `PCLK` cycles for 40-bit mode
- 5 × B – Number of cycles to calculate B biquads (This does not include the coefficient loading cycles as coefficients need to be loaded only once.)

The loading of input data and writing of output data is pipelined with the computation operation. The expression 5 × B × W includes input data loading, compute, and output data write back operations. This expression does not include the first input data loading, last output data write back, and write back of the updated input and output index registers, the latency of which is included in the TCB load.

ⓘ 14 `PCLK` cycles are required for TCB loading for coefficients and save state operation.

Programming Model

The IIR supports up to 24 channels which are time division multiplexed (TDM). Each channel can have a maximum of 12 cascaded biquads. The window size for each channel is configurable using control registers. A window size of 1 corresponds to sample based operation and the maximum window size is 64.

The coefficients are initially stored in internal memory and one TCB per channel is created in internal memory with each channels' TCB pointing to the next channels'. The TCB also contains channel specific control registers, input data buffer parameters and output data buffer parameters.

> The TCB of the last channel should point to the TCB of first channel.

The total number of channels is configured using the `IIRCTL1` register and DMA is enabled.

The procedure that the accelerator uses to process biquads is shown in Figure 7-14 and described in the following procedure.

1. The controller loads all coefficients of all the channels into local storage.
2. Once all the coefficients are loaded, the controller goes to the first biquad of the first channel and calculates the output of the first biquad and updates the intermediate results for that biquad.
3. Then, the accelerator moves to the next biquad of that channel and repeats the process until all the biquads for that channel are completed and the results are stored to memory.
4. This process is repeated with next sample until one window of the corresponding channel is processed.
5. After one window of the channel accelerator is processed, the accelerator moves to the next channel and computes the results.

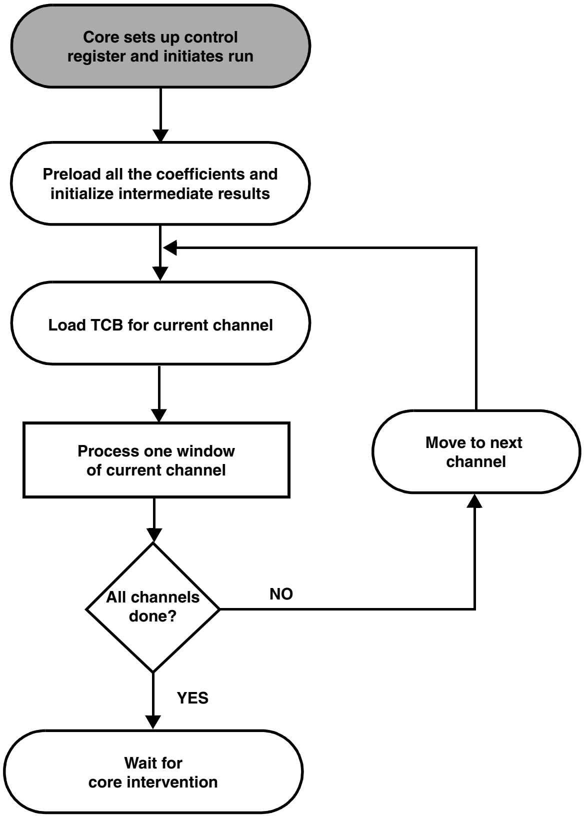

Figure 7-14. Biquad Processing Program Flow

Dynamic Coefficient Processing Notes

The IIR accelerator loads the coefficients for all the channels only once when the IIR accelerator is enabled. In order to re-load the new coefficients, the accelerator has to be disabled and re-enabled.

Writing to Local Memory

1. Enable IIR module in `PMCTL1` register.
2. Wait at least 4 `CCLK` cycles.

3. Clear the IIR_DMAEN bit in the IIRCTL1 register.
4. Set the IIR_DBGMODE, IIR_DBGMEM and IIR_HLD bits in the IIRDEBUGCTL register.
5. Set the IIR_ADRINC bit in IIRDEBUGCTL register for address auto increment.
6. Write start address to the IIRDBGADDR register. If bit 11 is set, coefficient memory is selected.
7. Wait at least 4 CCLK cycles.
8. Write data to the IIRDBGWRDATA_L register.
9. Write data to the IIRDBGWRDATA_H register.

Reading from Local Memory

1. Enable IIR module in PMCTL1 register.
2. Wait at least 4 CCLK cycles.
3. Clear the IIR_DMAEN bit in the IIRCTL1 register.
4. Set the IIR_DBGMODE, IIR_DBGMEM and IIR_HLD bits in the IIRDEBUGCTL register.
5. Set the IIR_ADRINC bit in the IIRDEBUGCTL register for address auto increment.
6. Write start address to the IIRDBGADDR register. If bit 11 is set, coefficient memory is selected.
7. Wait at least 4 CCLK cycles.
8. Read data from the IIRDBGRDDATA_L register.
9. Read data from the IIRDBGRDDATA_H register.

Single Step Mode

Single step mode can be used for debug purposes. An additional debug register is used in this mode.

1. Enable stop DMA during breakpoint hit in the emulator settings.
2. Clear the IIR_HLD bit and enable IIR_DBGMODE and IIR_RUN bits in IIRDEBUGCTL register.
3. Program FIR module according to the application.
4. In single step each iteration is updated in the emulator session.

Save Biquad State of the IIR

The following steps are required to resume IIR processing after being interrupted by another accelerator module.

1. When starting the accelerator for the first time, set the IIR_EN, IIR_DMAEN and IIR_SS bits.
2. The core waits for the first set of IIR processing to conclude or performs some other task.
3. The accelerator writes back the updated DMA index registers and the updated Dk values after the processing completes.
4. Disable the accelerator by clearing the IIR_EN bit. Optionally, clear the IIR_DMAEN bit.
5. The core and accelerator wait for the next set of data to be ready. (The FIR/FFT accelerator can be used for a completely different purpose during this time.)

6. Once the next block is ready for processing, enable the IIR accelerator again by setting the IIR_EN and IIR_DMAEN bits. The coefficients and the Dk values will be re-loaded back into the local memory.

7. The core waits for the current set of IIR processing to conclude or performs some other task.

Programming Example

In this example, an application needs IIR filtering for two channels of data; channel 1 has six biquads and channel 2 has eight biquads. The window size for all channels is 32.

1. Create a circular buffer in internal memory for each channel's data. The buffer should be large enough to avoid overwriting data before it is processed by the accelerator.

2. Configure internal memory buffers containing the 6 × 5 coefficients and the 6 × 2 Dk values for the channel 1 biquads, and the 8 × 5 coefficients and 8 × 2 Dk values of the channel 2 biquads.

3. Configure two TCBs in internal memory with each channel's chain pointer entry pointing to the next channel's and the last channel's chain pointer entry pointing to the first in a circular fashion.

4. Program the IIRCTL2 register to use channel 1 TCB for 6 biquads and a window size of 32, and channel 2 for 8 biquads and a window size of 32.

5. Configure the index, modifier, and length entries in the TCBs to point to the corresponding channel's data buffer, coefficient buffer and output data buffer.

   The location of the first channel's TCB is written to the chain pointer register in the accelerator.

6. Program the global control register `IIR_NCH` bit for 2 channels.
   a. The accelerator starts and loads the first channel's TCB, loads coefficients and Dk values of all the 6 biquads into local storage, then loads the TCB of the second channel, and finally loads coefficients and Dk values of all the 8 biquads.
   b. Once all the coefficients and Dk values are loaded, the controller loads the TCB of first channel and fetches the input sample. It then starts calculating the first biquad of the first channel.
   c. The accelerator calculates the output of the first biquad and then updates the intermediate results for that biquad. Then it moves to the next biquad of that channel and repeats the biquad processing until all the biquads for that channel are done and the final result is stored to memory.
   d. The accelerator repeats this process with next sample until one window of the corresponding channel is processed. Once the window is done, the accelerator saves the index values to memory and moves to the next channel. After both channels are done, the accelerator waits for core intervention.

Throughput Comparison – Accelerator Versus Core

The following sections provide throughput comparisons between the FIR/IIR/FFT hardware module and the core. The cycles for the accelerators were calculated using the throughput expressions provided in earlier sections while that for the core were calculated with the help of the optimized C library functions. In each case, the solid line is used for the core and the dashed line for the accelerator.

FFT

Figure 7-15 shows a graphical comparison between the number of `CCLK` cycles consumed by the core and the FFT accelerator to perform the FFT operation for different numbers of points (N).

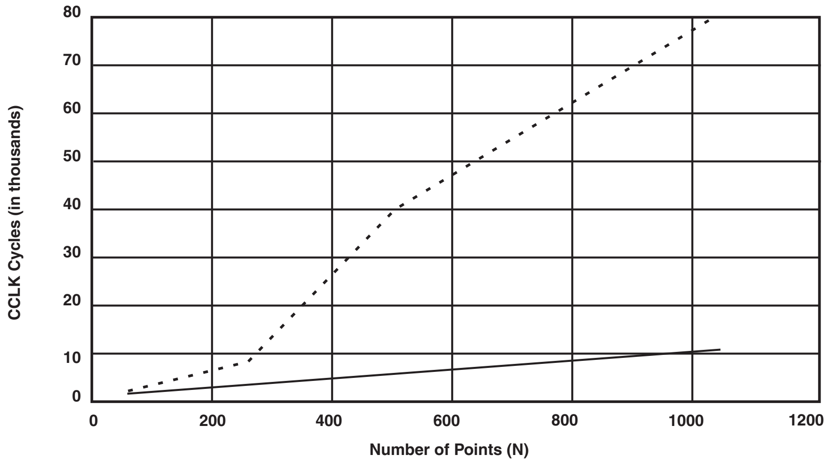

Figure 7-15. FFT Accelerator Comparison to Core

In most of the cases, the core takes fewer cycles than the FFT accelerator. However, the difference in number of cycles is comparatively less for N<=256 (small FFT) and becomes much more significant for N>256 (large FFTs).

FIR

Figure 7-16 through Figure 7-18 show a graphical comparison between the number of CCLK cycles consumed by the core and the FIR accelerator to perform the FIR operation for different window (block) size (W) and tap length (N) values.

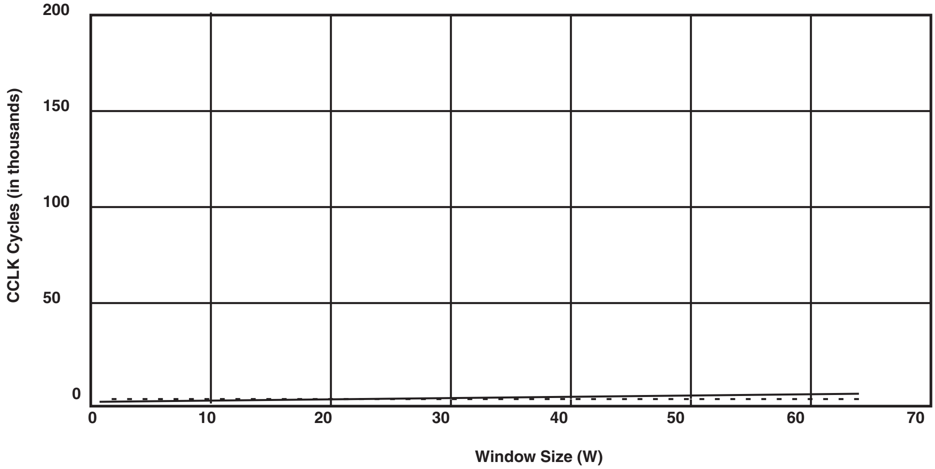

Figure 7-16. FIR Core/Accelerator Comparison, N = 256

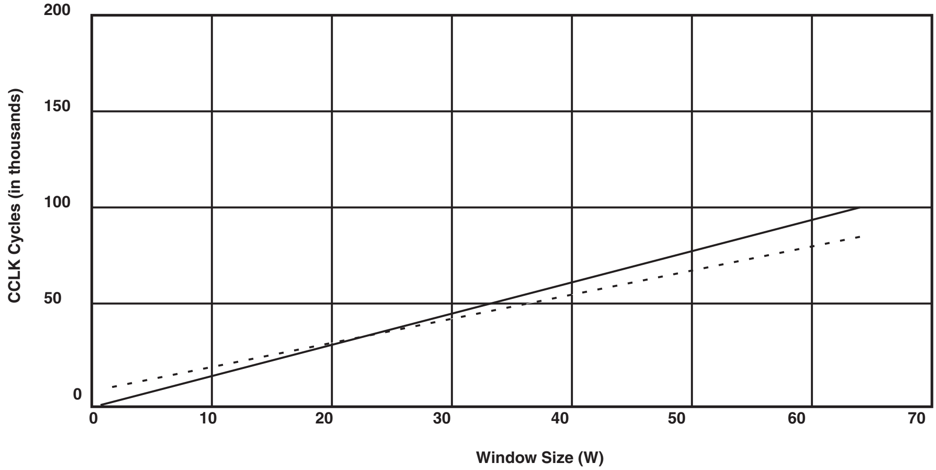

Figure 7-17. FIR Core/Accelerator Comparison, N = 2048

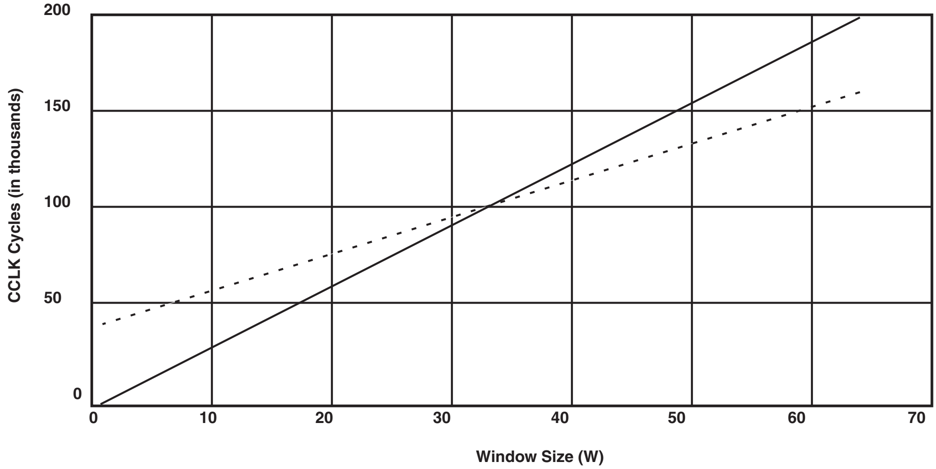

Figure 7-18. FIR Core/Accelerator Comparison, N = 4096

The cycles for the accelerator were calculated using the throughput expression provided in one of the previous sections while those for the core were calculated with the help of one of the C FIR library functions.

The figures show that for a fixed tap length, the larger the window size, the better performance is achieved using the FIR accelerator. As the window size increases, initially the accelerator consumes more cycles than the core. However, after a threshold, the accelerator performs better than the core. The reason of such behavior is the trade-off between the four MACs (each running simultaneously at the `PCLK` rate) and the overhead for initial pre-loading of the delay line and the coefficient memory. The cycles required by the pre-loading become less significant as the window size increases while the performance achieved using the four MACs in parallel become more significant.

IIR

Figure 7-19 through Figure 7-21 show a graphical comparison between the number of `CCLK` cycles consumed by the core and the IIR accelerator to perform the IIR operation for different block size (W) and filter order (N) or number of biquads (B = N/2).

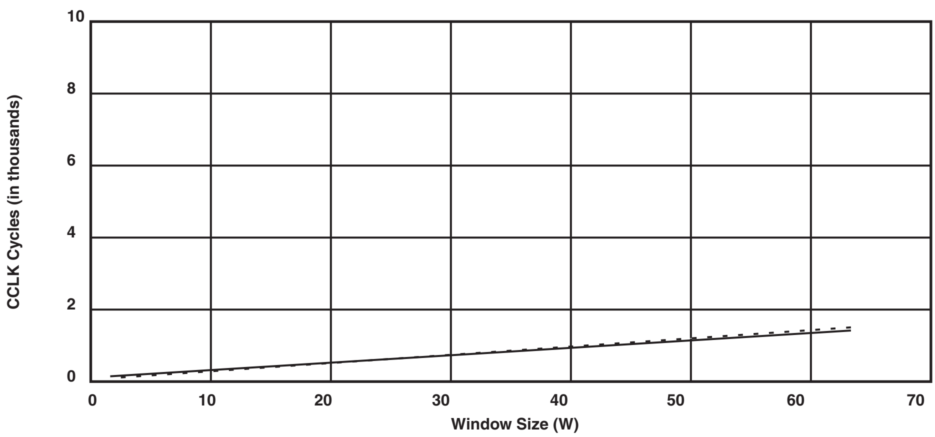

Figure 7-19. IIR Core/Accelerator Comparison, N = 4, B = 2

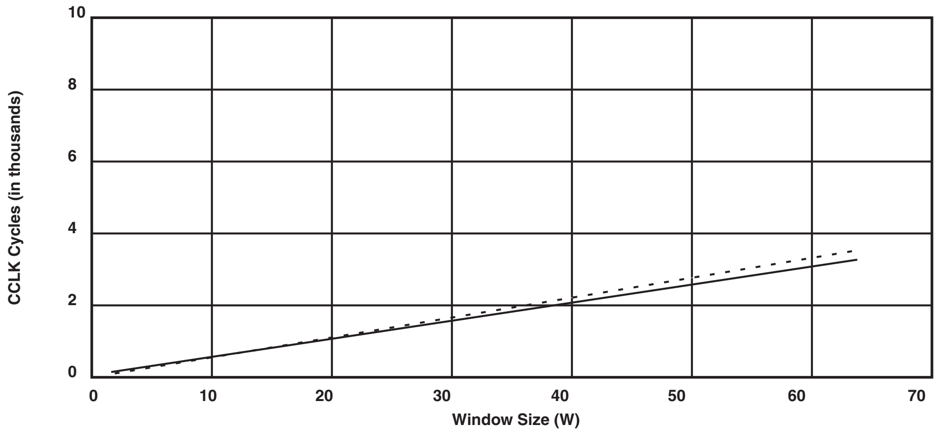

Figure 7-20. IIR Core/Accelerator Comparison, N = 12, B = 6

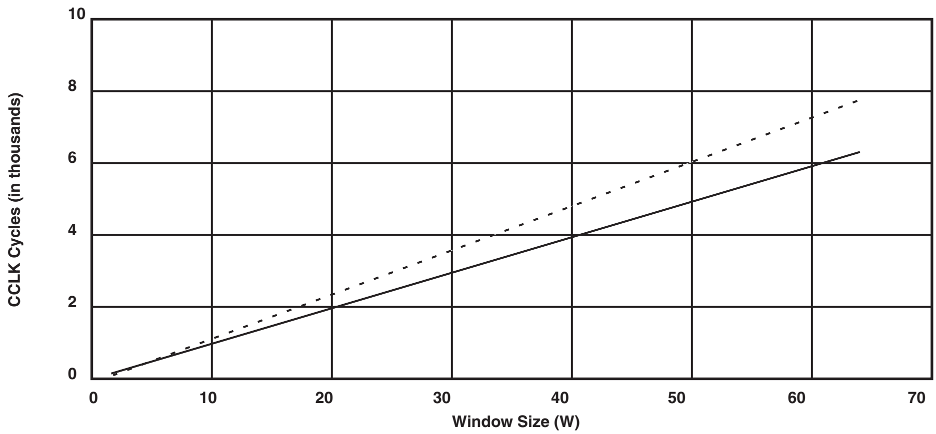

Figure 7-21. IIR Core/Accelerator Comparison, N = 24, B = 12

In most of these cases the core takes fewer cycles than the IIR accelerator. The difference between the cycles taken by the core and the accelerator is almost negligible for lower order IIR operations but becomes more significant for higher order IIR operations. This difference increases as the window size is increases.

Debug Features

The following sections describe the debugging features available on the accelerator.

Local Memory Access

The contents of IIR delay line and coefficient memories are made observable for debug by setting the IIR_DBGMODE/IIR_DBGMEM and IIR_HLD bits in the IIRDEBUGCTL control register. The debug address register (IIRDBGADDR) and four data registers are provided for debug operations. Bit 11 of the this register selects coefficient memory if set (=1) and selects delay line memory in cleared (=0).

The 40-bit wide debug mode read data register is organized as:

- The IIRDBGRDDATA_L register holds the lower 32 bits
- The IIRDBGRDDATA_H register holds the upper 8 bits

The 40-bit wide debug mode write data register is organized as:

- The IIRDBGWRDATA_L register holds the lower 32 bits and
- The IIRDBGWRDATA_H register holds the upper 8 bits

A read from the IIRDBGRDDATA_L register followed by a read from the IIRDBGRDDATA_H register returns the content of the 40-bit memory location pointed to by the address register. Data can be written into any memory location using the IIRDBGWRDATA_L register followed by the IIRDBGWRDATA_H register.

If the address auto increment bit (IIR_ADRINC) is set, the address register auto increments on IIRDBGWRDATA_H/L writes and IIRDBGRDDATA_H/L reads. During auto increment, the IIR_DBGADDR register cannot cross the data memory/coefficient memory boundary. The address boundary for data memory is 1024 locations and for coefficient memory 2048 locations.

Single Step Mode

Programs can single step through the MAC operations and observe the memory contents after each step. The `IIR_DBGMODE`/`IIR_HLD` and `IIR_RUN` bits control the IIR MAC units.

Emulation Considerations

In IIR debug mode, the DMA operations are not observable.

Application Guidelines for Effective Use of the FIR/IIR/FFT Accelerators

From the throughput numbers mentioned in the earlier sections of this chapter, one might find that the FIR/IIR/FFT accelerators/off-load engines may not finish the same processing task faster than using the core. This implies that using the accelerators in the conventional sequential approach and making the core wait until the accelerator finishes processing may not be advantageous from the performance point of view. The only way to get the most out of the accelerators is to make sure that the core is busy doing something useful when the accelerator is performing a processing task. Thus, in order to use these accelerators effectively, special care must be taken while designing the application framework. The following sections describe two approaches to consider.

Sample Versus Block Processing Operation

It may be difficult to achieve the required performance by using sample based processing (Window size = 1). Increasing the window size provides more computation time and the ability to perform the real time processing. It is a common practice to use block based processing (Window size greater 1).

In the following example, the core is running at 450 MHz with a single channel tap length of 512. Sampling frequency = 96 kHz and the maximum processing time is 10 μs.

The computation of the filter requires:
TCB load + 4 × N + W(N/4 + 2)) × C.

For window size = 1

49 + 4 × 512 + 1 × (512/4 + 2) = 2227 PCLK = 9.8 μs.

For window size = 10

49 + 4 × 512 + 10 × (512/4 + 2) = 3397 PCLK = 15 μs.

Therefore, 10 samples at 96 kHz = 10 × 10 = 100 us of available processing time: Actual time used is 15 μs.

Adding Pipeline Stages

The first approach is to pipeline the input and output data buffers wherever filter/FFT engines are to be used. This can be done by adding a ping-pong buffer mechanism where the core and the filter engines operate on two different blocks of the data. The data blocks move from/to the core and to/from the off-load engine in a pipelined fashion. This ensures that the core and filter engine operate totally independently and without the need to wait for each other. The disadvantage of this approach is that it adds more latency in the final output. This latency increases with the increase in the pipeline stages.

Splitting Tasks

The second approach is to off-load the core's tasks to the accelerators (relevant for applications where multichannel data is involved). This is accomplished by splitting the processing task between core and the accelerator. This doesn't require much change in the existing application framework as no data pipe-lining is used. When processing multichannel data, a few channels can be off-loaded to the filter/FFT engine and the rest to the processor core. With proper partitioning, the application ensures that the core isn't in an idle state after processing its channels.

8 PULSE WIDTH MODULATION

Pulse width modulation (PWM) is a technique for controlling analog circuits with a microprocessor's digital outputs. PWM is employed in a wide variety of applications, ranging from measurement to communications to power control and conversion. The interface specifications are shown in Table 8-1.

Table 8-1. PWM Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	Yes, (External port)
SRU DAI Required	No
SRU DAI Default Routing	N/A
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	N/A
Transmission Simplex	N/A
Transmission Half Duplex	N/A
Transmission Full Duplex	N/A
Access Type	
Data Buffer	No
Core Data Access	N/A
DMA Data Access	N/A

Table 8-1. PWM Specifications (Cont'd)

Feature	Availability
DMA Channels	N/A
DMA Chaining	N/A
Boot Capable	N/A
Local Memory	No
Clock Operation	f_{PCLK}

Features

The following is a brief summary of the features of this interface.

- Four independent PWM units
- 2-phase output timing unit
- Center or edge aligned PWM
- Single or double update PWM timer period
- Output logic allows redirection of 2-phase output timing
- PWM units can operate synchronized to each other
- Complementary outputs allows bridge based applications

A block diagram of the module is shown in Figure 8-1. The generation of the four output PWM signals on pins AH to BL is controlled by four primary blocks.

- The two-phase PWM timing unit, which is the core of the PWM controller, generates two pairs of complemented center based PWM signals.

- The emergency dead time insertion is implemented after the 'ideal' PWM output pair, including crossover, is generated.
- The output control unit allows the redirection of the outputs of the two-phase timing unit for each channel to either the high-side or the low-side output. In addition, the output control unit allows individual enabling/disabling of each of the four PWM output signals.
- The PWM interrupt controller generates an interrupt at the start of the PWM period which is shared for all modules.

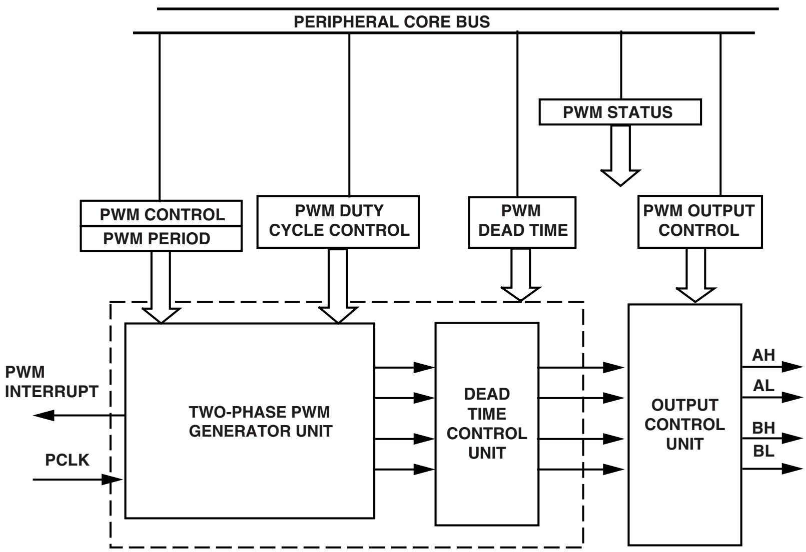

Figure 8-1. PWM Module Block Diagram

Pin Descriptions

The PWM module has four groups of four PWM outputs each, for a total of 16 PWM outputs. These outputs are described in Table 8-2.

Table 8-2. PWM Pin Descriptions

Multiplexed Pin Name	Direction	Description
PWM_AH3–0	O	PWM output of pair A produce high side drive signals.
PWM_AL3–0	O	PWM output of pair A produce low side drive signals. Note in paired mode, this pin is the complement of AH3-0.
PWM_BH3–0	O	PWM output of pair B produce high side drive signals.
PWM_BL3–0	O	PWM output of pair B produce low side drive signals. Note in paired mode, this pin is the complement of BH3-0.

Multiplexing Scheme

By default the PWM output pins are disabled. To enable the PWM units refer to Table 24-15 on page 24-30. Table 8-3 shows the connection to the PWM outputs on the external port pins. For more information, see "Pin Multiplexing" on page 24-28.

Table 8-3. PWM Connections

PWM Unit	Pin Multiplexing[1]
PWM0	ADDR8 = AL0 ADDR9 = AH0 ADDR10 = BL0 ADDR11 = BH0
PWM1	ADDR12 = AL1 ADDR13 = AH1 ADDR14 = BL1 ADDR15 = BH1

Table 8-3. PWM Connections (Cont'd)

PWM Unit	Pin Multiplexing[1]
PWM2	ADDR16 = AL2 ADDR17 = AH2 ADDR18 = BL2 ADDR19 = BH2
PWM3	ADDR20 = AL3 ADDR21 = AH3 ADDR22 = BL3 ADDR23 = BH3

1 For ADSP-2146x products the pins are AMI_ADDRx.

SRU Programming

The ADSP-2147x and ADSP-2148x can output the PWM3–1 units over the DPI pins. The `PWMONDPIEN` bit (bit 30 in the `SYSCTL` register) enables the routing output logic for the DPI group B register.

Register Overview

This section provides brief descriptions of the major registers. For complete register information, see Appendix A, Register Reference.

- **PWM Global Control Register (PWMGCTL).** Enables or disables the four PWM groups simultaneously in any combination for synchronization between the PWM groups.
- **PWM Global Status Register (PWMGSTAT).** Provides the status of each PWM group.
- **PWM Control Registers (PWMCTLx).** Used to set the operating modes of each PWM block. This register also allows programs to disable interrupts from individual groups.

- **PWM Status Registers (PWMSTATx).** Report the phase and mode status for each PWM group.

The traditional read-modify-write operation to enable/disable a peripheral is different for the PWMs. For more information, see "Global Control Register (PWMGCTL)" on page A-66.

Clocking

The fundamental timing clock of the PWM controllers is peripheral clock (PCLK). The clock to this module may be shut off for power savings. For more information, see "Disabling Peripheral Clocks" on page 23-11.

Functional Description

The individual elements shown in Figure 8-1 are described in detail in the following sections.

Two-Phase PWM Generator

Each PWM group is able to generate complementary signals on two outputs in paired mode or each group can provide independent outputs in non-paired mode.

Switching Frequencies

The 16-bit read/write PWM period registers, PWMPERIOD3-0, control the PWM switching frequency.

The PWM generator does not support external synchronization mode.

The fundamental timing unit of the PWM controller is PCLK. Therefore, for a 200 MHz peripheral clock, the fundamental time increment is 5 ns.

The value written to the PWMPERIODx register is effectively the number of PCLK clock increments in a PWM period (edge aligned mode) or in a half PWM period (center aligned mode) in half a PWM period.

Therefore, the PWM switching period, T_s, can be written as:

$T_s = 2 \times PWMPERIOD \times t_{PCLK}$ (edge aligned)
$T_s = PWMPERIOD \times t_{PCLK}$ (center aligned)

For example, for a 200 MHz PCLK and a desired PWM center aligned switching frequency of 10 kHz (T_s = 100 μs), the correct value to load into the PWMPERIODx register is:

$$PWMPERIOD = \frac{200 \times 10^6}{2 \times 10 \times 10^3} = 10000$$

The largest value that can be written to the 16-bit PWMPERIODx register is 0xFFFF = 65,535 which corresponds to a minimum PWM switching frequency of:

$$f_{(PWM),min} = \frac{200 \times 10^6}{2 \times 65535} = 1523 Hz$$

PWMPERIOD values of 0 and 1 are not defined and should not be used when the PWM outputs or PWM sync is enabled.

Duty Cycles

The two 16-bit read/write duty cycle registers, PWMA and PWMB, control the duty cycles of the four PWM output signals on the PWM pins. The two's-complement integer value in the PWMA register controls the duty cycle of the signals on the PWM_AH and PWM_AL. The two's-complement integer value in the PWMB register controls the duty cycle of the signals on PWM_BH and PWM_BL pins. The duty cycle registers are programmed in two's-complement integer counts of the fundamental time unit, PCLK, and define the desired on-time of the high-side PWM signal produced by the two-phase timing unit over half the PWM period. The duty cycle register range is from:

(–*PWPERIOD* ÷ 2 – *PWMDT*) to (+*PWPERIOD* ÷ 2 + *PWMDT*)

which, by definition, is scaled such that a value of 0 represents a 50% PWM duty, cycle. The switching signals produced by the two-phase timing unit are also adjusted to incorporate the programmed dead time value in the PWMDT register. The two-phase timing unit 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 (in this case for PWM_AH and PWM_AL) from the timing unit are shown in Figure 8-2 for operation in single update mode. All illustrated time values indicate the integer value in the associated register and can be converted to time by simply multiplying by the fundamental time increment, (PCLK) and comparing this to the two's-complement counter. Note that the switching patterns are perfectly symmetrical about the midpoint of the switching period in single update mode since the same values of the PWMAx, PWMPERIODx, and PWMDTx registers are used to define the signals in both half cycles of the period.

Further, the programmed duty cycles are adjusted to incorporate the desired dead time into the resulting pair of PWM signals. As shown in Figure 8-2, the dead time is incorporated by moving the switching instants of both PWM signals (PWM_AH and PWM_AL) away from the instant set by the PWMAx registers. Both switching edges are moved by an equal amount (PWMDT × PCLK) to preserve the symmetrical output patterns. Also shown is the PWM_PHASE bit of the PWMSTAT register that indicates whether operation is in the first or second half cycle of the PWM period.

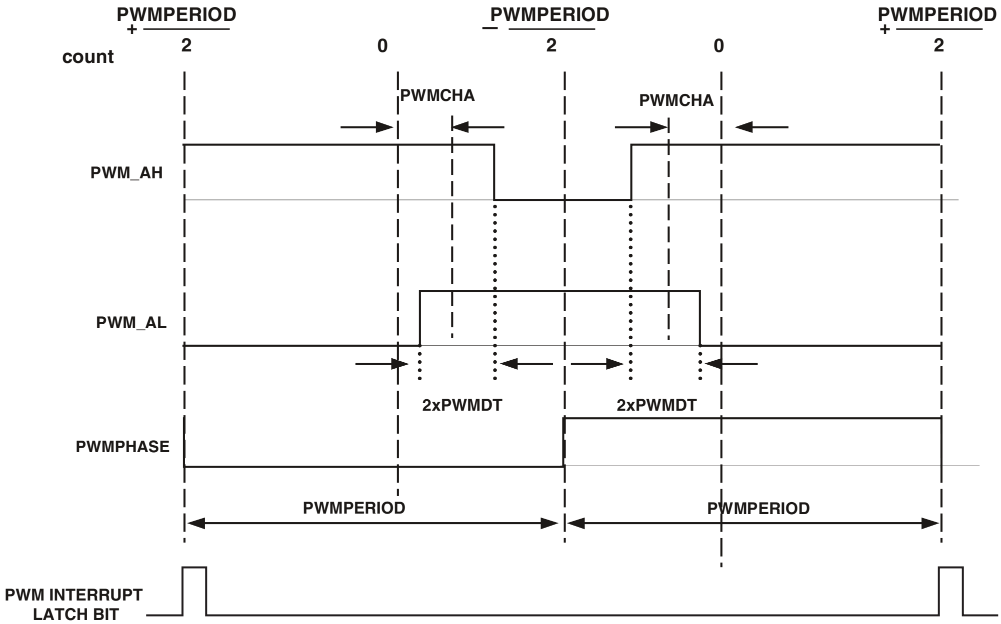

Figure 8-2. Center-Aligned Paired PWM in Single Update Mode, Low Polarity

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 Figure 8-2 may be written as:

The range of T_{AH} is:

$$[0 - 2 \times PWMPERIOD \times t_{PCLK}]$$

and the corresponding duty cycles are:

$$T_{AH} = (PWMPERIOD - 2 \times (PWMCHA + PWMDT)) \times t_{PCLK}$$

The range of T_{AL} is:

$$[0 - 2 \times PWMPERIOD \times t_{PCLK}]$$

and the corresponding duty cycles are:

$$d_{AH} = \frac{t_{AH}}{T_S} = \frac{1}{2} + \frac{PWMCHA - PWMDT}{PWMPERIOD}$$

$$d_{AL} = \frac{t_{AL}}{T_S} = \frac{1}{2} + \frac{PWMCHA - PWMDT}{PWMPERIOD}$$

The minimum permissible value of T_{AH} and T_{AL} is zero, which corresponds to a 0% duty cycle, and the maximum value is T_S, the PWM switching period, which corresponds to a 100% duty cycle. Negative values are not permitted.

The output signals from the timing unit for operation in double update mode are shown in Figure 8-3. This illustrates a general case where the switching frequency, dead time, and duty cycle are all changed in the second half of the PWM period. The same value for any or all of these quantities can be used in both halves of the PWM cycle. However, there is no guarantee that a symmetrical PWM signal will be produced by the timing unit in this double update mode. Additionally, Figure 8-3 shows that the dead time is inserted into the PWM signals in the same way as in single update mode.

In general, the on-times (active low) of the PWM signals over the full PWM period in double update mode can be defined as:

$$T_S = (PWMPERIOD_1 + PWMPERIOD_2) \times t_{PCLK}$$

$$T_{AL} = \left(\frac{PWMPERIOD_1}{2} + \frac{PWMPERIOD_2}{2} - PWMCHA_1 - PWMCHA_2 - PWMDT_1 - PWMDT_2\right) \times t_{PCLK}$$

$$T_{AH} = \left(\frac{PWMPERIOD_1}{2} + \frac{PWMPERIOD_2}{2} + PWMCHA_1 + PWMCHA_2 - PWMDT_1 - PWMDT_2\right) \times t_{PCLK}$$

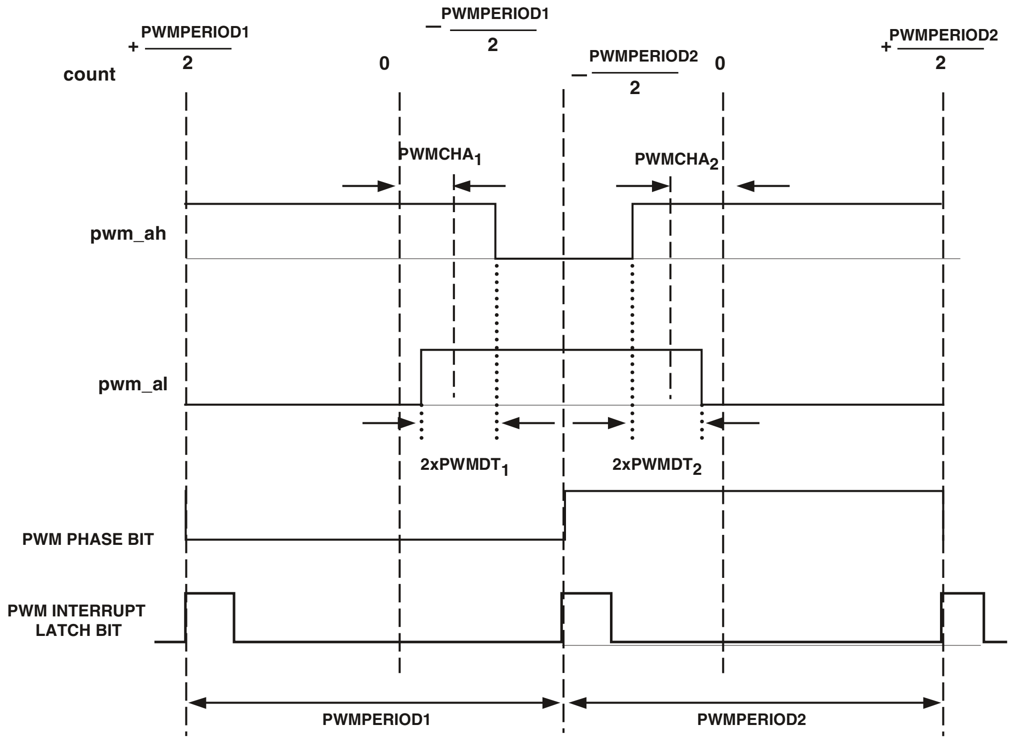

Figure 8-3. Center-Aligned Paired PWM in Double Update Mode, Low Polarity

where subscript 1 refers to the value of that register during the first half cycle and subscript 2 refers to the value during the second half cycle. The corresponding duty cycles are:

$$d_{AL} = \frac{T_{AL}}{T_S} = \frac{1}{2} - \frac{(PWMCHA_1 + PWMCHA_2 + PWMDT_1 + PWMDT_2)}{(PWMPERIOD_1 + PWMPERIOD_2)}$$

$$d_{AH} = \frac{T_{AH}}{T_H} = \frac{1}{2} + \frac{(PWMCHA_1 + PWMCHA_2 - PWMDT_1 - PWMDT_2)}{(PWMPERIOD_1 + PWMPERIOD_2)}$$

since for the general case in double- update mode, the switching period is given by:

$$T_S = (PWMPERIOD_1 + PWMPERIOD_2) \times t_{PCLK}$$

Again, the values of T_{AH} and T_{AL} are constrained to lie between zero and T_S. Similar PWM signals to those illustrated in Figure 8-2 and Figure 8-3 can be produced on the BH and BL outputs by programming the `PWMBx` registers in a manner identical to that described for the `PWMAx` registers.

Dead Time

The second important parameter that must be set up in the initial configuration of the PWM block is the switching dead time. This is a short delay time introduced between turning off one PWM signal (say AH) and turning on the complementary signal, AL. This short time delay is introduced to permit 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 `PWMDT3-0` registers control the dead time. The dead time, T_d, is related to the value in the `PWMDTx` registers by:

$$T_d = PWMDT \times 2 \times t_{PCLK}$$

Therefore, a PWMDT value of 0x00A (= 10), introduces a 200 ns delay between when the PWM signal (for example `AH`) is turned off and its complementary signal (`AL`) is turned on. The amount of the dead time can therefore be programmed in increments of 2 × PCLK (or 10 ns for a 200 MHz peripheral clock). The PWMDTx registers are 10-bit registers, and the maximum value they can contain is 0x3FF (= 1023) which corresponds to a maximum programmed dead time of:

$$T_{d,max} = 1023 \times 2 \times t_{PCLK} = 1023 \times 2 \times 5 \times 10^{-9} = 10.2\mu s$$

This equates to an PCLK rate of 200 MHz. Note that dead time can be programmed to zero by writing 0 to the PWMDTx registers (see "Pulse Width Modulation Registers" on page A-66).

Output Control Unit

The PWMSEG register contains four bits (0 to 3) that can be used to individually enable or disable each of the 4 PWM outputs.

Output Enable

If the associated bit of the PWMSEG register is set (=1), then the corresponding PWM output is disabled, regardless of the value of the corresponding duty cycle register. This PWM output signal remains disabled as long as the corresponding enable/disable bit of the PWMSEGx register is set. In single update mode, changes to this register only become effective at the start of each PWM cycle. In double update mode, the PWMSEG register can also be updated at the mid-point of the PWM cycle.

After reset, all four enable bits of the PWMSEG register are cleared so that all PWM outputs are enabled by default.

Output Polarity

The polarity of the generated PWM signals is programmed using the PWMPOLARITY3-0 registers, so that either active high or active low PWM patterns can be produced. The polarity values can be changed on the fly if required, provided the change is done a few cycles before the next period change.

Complementary Outputs

The PWM controller can be operated in paired or non paired mode (PWMCTLx register).

In non paired mode (default) both outputs (high and low side) are driven independently. Since paired mode drives the output logic of the PWM in a complementary fashion (low side = /high side), this feature may be useful in PWM bridge applications.

Crossover

The PWMSEG3-0 registers contain two bits (AHAL_XOVR and BHBL_XOVR), one for each PWM output. If crossover mode is enabled for any pair of PWM signals, the high-side PWM signal from the timing unit (for example, AH) is diverted to the associated low side output of the output control unit so that the signal ultimately appears at the AL pin.

The corresponding low side output of the timing unit is also diverted to the complementary high side output of the output control unit so that the signal appears at the AH pin. Following a reset, the two crossover bits are cleared so that the crossover mode is disabled on both pairs of PWM signals. Even though crossover is considered an output control feature, dead time insertion occurs after crossover transitions to eliminate shoot-through safety issues.

Note that crossover mode does not work if:

1. One signal of PWM_AL–PWM_AH or PWM_BL–PWM_BH is disabled.
2. PWM_AL and PWM_AH or PWM_BL and PWM_BH have different polarity settings from PWMPOLx registers.

In other words, both PWM_AL and PWM_AH or PWM_BL and PWM_BH should be enabled and both should have same polarity for proper operation of cross-over mode.

Emergency Dead Time for Over Modulation

The PWM timing unit is capable of producing PWM signals with variable duty cycle values at the PWM output pins. At the extreme side of the modulation process, settings of 0% and 100% modulation are possible. These two modes are termed full off and full on respectively.

Full off and full on over-modulation is entered by virtue of the commanded duty cycle values in conjunction with the setting in the PWMDTx registers. Settings that fall between the extremes are considered normal modulation. These settings are explained in more detail below.

Full On. The PWM for any pair of PWM signals operates in full on when the desired high side output of the two-phase timing unit is in the on state (low) between successive PWM interrupts.

Full Off. The PWM for any pair of PWM signals operates in full off when the desired high side output of the two-phase timing unit is in the off state (high) between successive PWMSYNC pulses.

Normal Modulation. The PWM for any pair of PWM signals operates in normal modulation when the desired output duty cycle is other than 0% or 100% between successive PWMSYNC pulses.

There are certain situations, when transitioning either into or out of either full on or full off, where it is necessary to insert additional *emergency dead time* delays to prevent potential *shoot-through conditions* in the inverter. These transitions are detected automatically and, if appropriate, the emergency dead time is inserted to prevent the shoot through conditions.

Inserting additional emergency dead time into one of the PWM signals of a given pair during these transitions is only needed if both PWM signals would otherwise be required 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 (if turning on during this dead time region), of this signal is delayed by an amount of $2 \times \text{PWMDT} \times \text{PCLK}$ 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.

Figure 8-4 illustrates two examples of such transitions. In (a), when transitioning from normal modulation to full on at the half cycle boundary in double update mode, no special action is needed. However in (b), when transitioning into full off at the same boundary, an additional emergency dead time is necessary. This inserted dead time is a little different from the normal dead time as it is impossible to move one of the switching events back in time because this would move the event into the previous modulation cycle. Therefore, the entire emergency dead time is inserted by delaying the turn on of the appropriate signal by the full amount.

Output Control Feature Precedence

The order in which output control features are applied to the PWM signal is significant and important. The following lists the order in which the signal features are applied to the PWM output signal.

1. Duty Cycle Generation
2. Crossover
3. Output Enable
4. Emergency Dead Time Insertion
5. Output Polarity

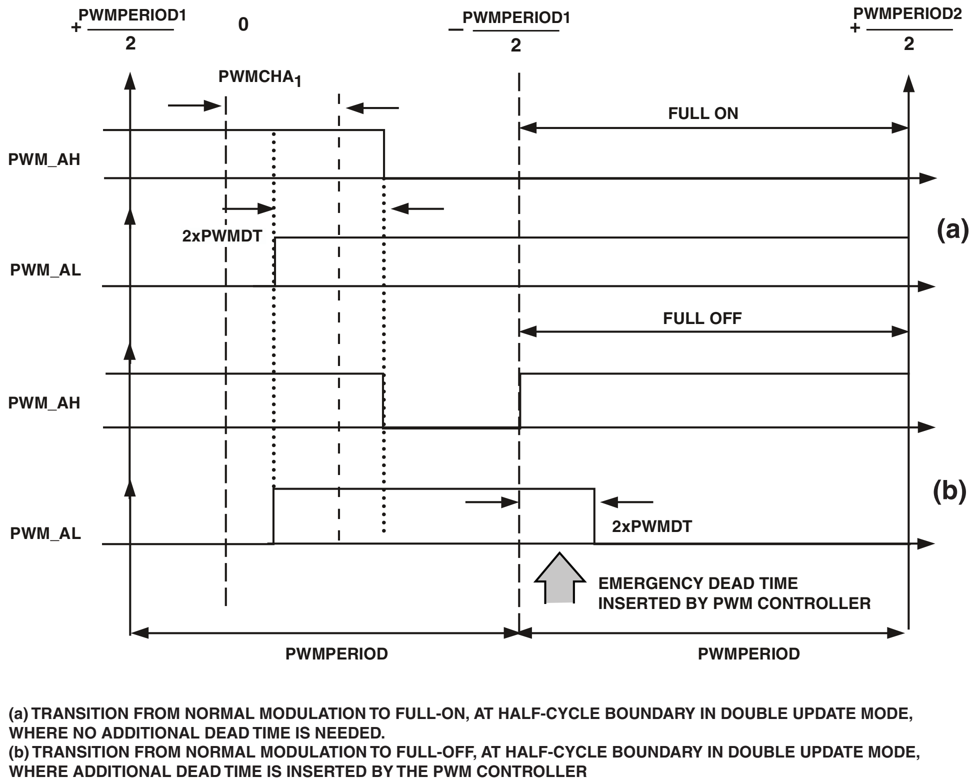

Figure 8-4. Normal Modulation to Full ON to Full OFF Transition

Operation Modes

The following sections provide information on the operating modes of the PWM module.

Waveform Modes

The PWM module can operate in both edge- and center-aligned modes. These modes are described in the following sections.

Edge-Aligned Mode

In edge-aligned mode, shown in Figure 8-5, the PWM waveform is left-justified in the period window. A duty value of zero, programmed through the PWMAx registers, produces a PWM waveform with 50% duty cycle. For even values of period, the PWM pulse width is exactly period/2, whereas for odd values of period, it is equal to period/2 (rounded up). Therefore for a duty value programmed in two's-complement, the PWM pulse width is given by:

To generate constant logic high on PWM output, program the duty register with the value ≥ + period/2.

To generate constant logic low on PWM output, program the duty register with the value ≥ – period/2.

For example, using an odd period of p = 2n + 1, the counter within the PWM generator counts as (–n...0...+n). If the period is even (p = 2n) then the counter counts as (–n+1...0...n).

The PWM switching period time for edge aligned mode is:
$T_s = t_{PCLK} \times$ PWMPERIOD.

For more information see "Pulse Width Modulation Registers" on page A-66.

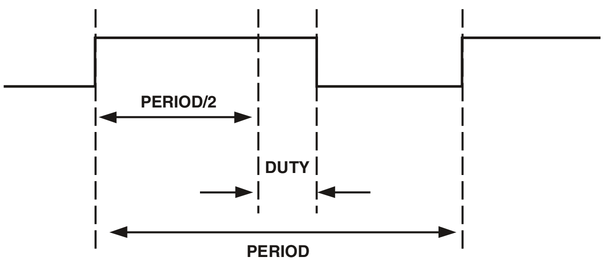

Figure 8-5. Edge Aligned PWM Wave with High Polarity

Center-Aligned Mode

Most of the following description applies to paired mode, but can also be applied to non-paired mode, the difference being that each of the four outputs from a PWM group is independent. Within center aligned mode, shown in Figure 8-6 there are several options to choose from.

Center-Aligned Single Update Mode. Duty cycle values are programmable only once per PWM period, so that the resultant PWM patterns are symmetrical about the mid-point of the PWM period.

Center-Aligned Double Update Mode. Duty cycle values are programmable only twice per PWM period. This second updating of the PWM registers is implemented at the mid-point of the PWM period, producing asymmetrical PWM patterns that produce lower harmonic distortion in two-phase PWM inverters.

Center-Aligned Paired Mode. Generates complementary signals on two outputs.

Center-Aligned Non-Paired Mode. Generates independent signals on two outputs.

In paired mode, the two's-complement integer values in the 16-bit read/write duty cycle registers, `PWMAx` and `PWMBx`, control the duty cycles of the four PWM output signals on the `PWM_AL`, `PWM_AH`, `PWM_BL` and `PWM_BH` pins respectively. The duty cycle registers are programmed in two's-complement integer counts of the fundamental time unit, `PCLK` and define the desired on time of the high side PWM signal over one-half the PWM period.

The duty cycle register range is from (–PWMPERIOD/2 – PWMDT) to (+PWMPERIOD/2 + PWMDT), which, by definition, is scaled such that a value of 0 represents a 50% PWM duty cycle.

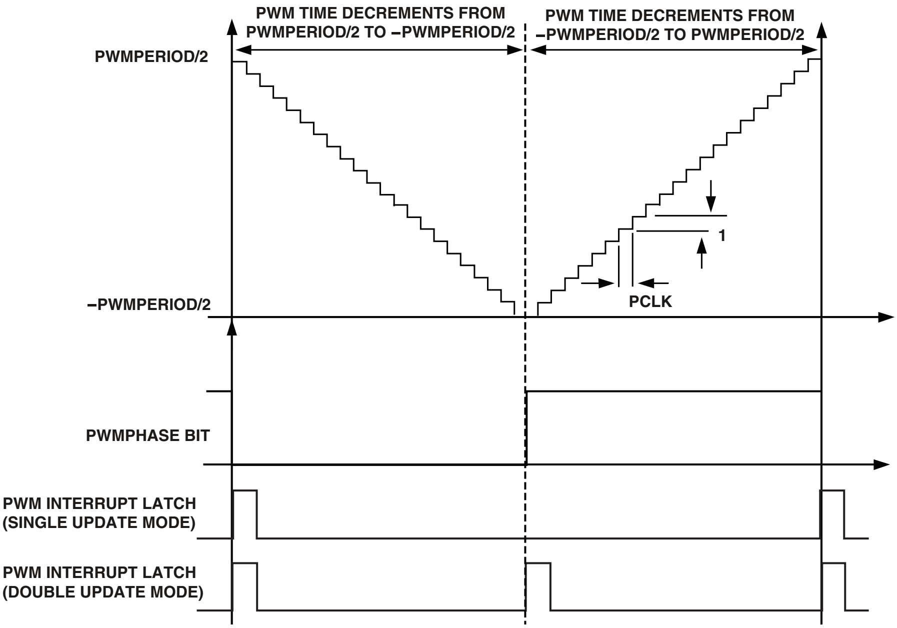

Figure 8-6. Operation of Internal PWM Timer (Center Aligned)

Each group in the PWM module (0–3) has its own set of registers which control the operation of that group. The operating mode of the PWM block (single or double update mode) is selected by the `PWM_UPDATE` bit (bit 2) in the PWM control (`PWMCTRL3-0`) registers. Status information about each individual PWM group is available to the program in the PWM status (`PWMSTAT3-0`) registers. Apart from the local control and status registers for each PWM group, there is a single PWM global control register (`PWMGCTL`) and a single PWM global status register (`PWMGSTAT`). The global control register allows programs to enable or disable the four groups in any combination, which provides synchronization across the four PWM groups.

The global status register shows the period completion status of each group. On period completion, the corresponding bit in the `PWMGSTAT` register is set and remains sticky. The program first reads the global status register and clears all the intended bits by explicitly writing 1.

PWM Timer Edge Aligned Update

The internal operation of the PWM generation unit is controlled by the PWM timer which is clocked at the peripheral clock rate, PCLK. The operation of the PWM timer over one full PWM period is illustrated in Figure 8-7. It can be seen that during the first half cycle, the PWM timer decrements from PWMPERIOD/2 to 0 using a two's-complement count.

At this point, the count direction changes and the timer continues to increment from 0 to the PWMPERIOD/2 value.

Also shown in Figure 8-7 are the PWM interrupt pulses for operation in edge aligned mode. An PWM interrupt is latched at the beginning of every PWM cycle. Note that the PWMPHASE bit (PWMSTAT register) has no meaning in this mode and is always set.

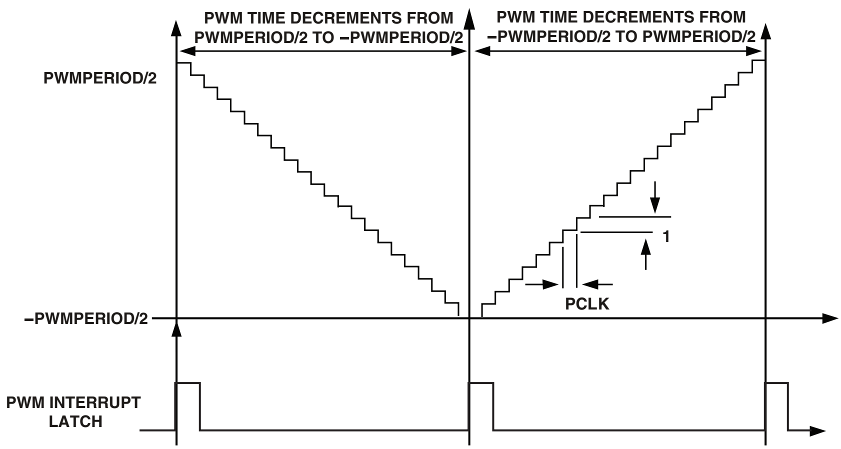

Figure 8-7. Operation of Internal PWM Timer (Edge Aligned)

Single Update Mode

In single update mode, a single PWM interrupt is produced in each PWM period. The rising edge of this signal marks the start of a new PWM cycle and is used to latch new values from the PWM configuration registers (PWPERIOD and PWMDT) and the PWM duty cycle registers (PWMCHx) into the two-phase timing unit. In addition, the PWMSEG register is also latched into the output control unit on the rising edge of the PWM interrupt latch pulse. In effect, this means that the characteristics and resultant duty cycles of the PWM signals can be updated only once per PWM period at the start of each cycle. The result is that PWM patterns that are symmetrical about the mid-point of the switching period are produced.

Double Update Mode

In double update mode, there is an additional PWM interrupt latch pulse produced at the mid-point of each PWM period. The rising edge of this new PWM pulse is again used to latch new values of the PWM configuration registers, duty cycle registers and the PWMSEG register. As a result, it is possible to alter both the characteristics (switching frequency and dead time) as well as the output duty cycles at the mid-point of each PWM cycle. Consequently, it is possible to produce PWM switching patterns that are no longer symmetrical about the mid-point of the period (asymmetrical PWM patterns).

In double update mode, it may be necessary to know whether operation at any point in time is in either the first half or the second half of the PWM cycle. This information is provided by the PWMPHASE bit of the PWMSTAT register which is cleared during operation in the first half of each PWM period (between the rising edge of the original PWM interrupt latch pulse and the rising edge of the new PWM interrupt pulse introduced in double update mode). The PWMPHASE bit of the PWMSTAT register is set during operation in the second half of each PWM period. This status bit allows programs to make a determination of the particular half-cycle during implementation of the PWM interrupt service routine, if required.

The advantage of the double update mode is that the PWM process can produce lower harmonic voltages and faster control bandwidths are possible. However, for a given PWM switching frequency, the interrupts occur at twice the rate as in double update mode. Since new duty cycle values must be computed in each PWM interrupt service routine, there is a larger computational burden on the processor in the double update mode. Alternatively, the same PWM update rate may be maintained at half the switching frequency to give lower switching losses.

Effective Accuracy

The PWM has 16-bit resolution but accuracy is dependent on the PWM period. In single update mode, the same values of PWMA and PWMB are used to define the on times in both half cycles of the PWM period. As a result, the effective accuracy of the PWM generation process is 2 x PCLK (or 10 ns for a 200 MHz clock). Incrementing one of the duty cycle registers by one changes the resultant on time of the associated PWM signals by PCLK in each half period (or 2 x PCLK for the full period). In double update mode, improved accuracy is possible since different values of the duty cycles registers are used to define the on times in both the first and second halves of the PWM period. As a result, it is possible to adjust the on-time over the whole period in increments of PCLK. This corresponds to an effective PWM accuracy of PCLK in double update mode (or 10 ns for a 200 MHz clock). The achievable PWM switching frequency at a given PWM accuracy is tabulated in Table 8-4. In Table 8-4, PCLK = 200 MHz.

Table 8-4. PWM Accuracy in Single- and Double Update Modes

Resolution (bits)	Single Update Mode PWM Frequency (kHz)	Double Update Mode PWM Frequency (kHz)
8	200 MHz $\div$ 2 $\times$ 2^8 = 390.63	200 MHz $\div$ 2^8= 781.25
9	195.3	390.6
10	97.7	195.3
11	48.8	97.7

Table 8-4. PWM Accuracy in Single- and Double Update Modes (Cont'd)

Resolution (bits)	Single Update Mode PWM Frequency (kHz)	Double Update Mode PWM Frequency (kHz)
12	24.4	48.8
13	12.2	24.4
14	6.1	12.2

Synchronization of PWM Groups

The PWMGCTL register enables or disables the four PWM groups in any combination. This provides synchronization across the four PWM groups.

The PWM_SYNCENx bits in this register can be used to start the counter without enabling the outputs through PWM_EN. So when PWM_ENx is asserted, the 4 PWM outputs are automatically synced to the initially programmed period. In most cases, all SYNC bits can be initialized to zero, enabling the PWM_ENx bits of the four PWM groups at the same time synchronizes the four groups.

The PWM sync enable feature allows programs to enable the PWM_SYNCENx bits to independently start the main counter without enabling the corresponding PWM module using the PWM_ENx bits. To synchronize different groups, enable the corresponding group's PWM_ENx bit at the same time. In order to stop the counter both the PWM_DISx and PWM_SYNCDISx bits should be set in this register.

Interrupts

Table 8-5 provides an overview of PWM interrupts.

Table 8-5. Overview of PWM Interrupts

Default Programmable Interrupt	Sources	Masking	Service
PWMI not connected by default	PWM period start	PWM_IRQEN bit (PWMCTLx)	RW1C to PWMGSTAT + RTI instruction

Sources

The PWM module drives one interrupt signal, PWMI. All four PWM unit interrupts are logically ORed into the interrupt signal. The PWM ports can generate interrupts under these conditions.

PWM Period

Whenever a period starts, the PWM interrupt is generated. The interrupt latch bit is set 1 PCLK cycle after the PWM counter resumes.

Masking

The PWMI signal is not routed by default to programmable interrupts. To service the PWM port, unmask (set = 1) any programmable interrupt bit in the IMASK/LIRPTL register. For interrupt execution, the specific PWM_IRQEN bit in the PWMCTLx register must be set.

Service

Since all four PWM units share the same interrupt vector, the interrupt service routine should read the PWMGSTAT register in order to determine the source of the interrupt. Next, the ISR needs to clear the status bits of the PWMGSTAT register by explicitly writing 1 into the status bit (RW1C operation) as shown in Listing 8-1.

Listing 8-1. Writing 1 Into the Status Bit

```
GPWM_ISR:
ustat2=dm(PWMGSTAT);          /* read global status reg */
bit tst ustat2 PWM_STAT2;     /* test PWM2 status */
if tf jump PWM2_ISR;          /* jump to PWM2 routine */
instruction;
instruction;
PWM2_ISR:
r1=PWM_STAT2;
dm(PWMGSTAT)=r1;          /* W1C to clear PWM2 interrupt */
r10=dm(PWMCTL2);          /* dummy read for write latency */
instruction;
rti;
```

Typically the PWM interrupt is used to periodically execute an interrupt service routine (ISR) to update the two PWM channel duty registers according to a control algorithm based on expected system operation. The PWM interrupt can trigger the ADC to sample data for use during the ISR. During processor boot the PWM is initialized and program flow enters a wait loop. When a PWM interrupt occurs, the ADC samples data, the data is algorithmically interpreted, and new PWM channel duty cycles are calculated and written to the PWM. More sophisticated implementations include different startup, runtime, and shutdown algorithms to determine PWM channel duty cycles based on expected behavior and further features.

During initialization, the `PWMPERIOD` register is written to define the PWM period and the `PWMCHx` registers are written to define the initial channel pulse widths. The `PWMSEG` and `PWMCHx` registers are also written, depending on the system configuration and modes. During the PWM interrupt driven control loop, only the `PWMCHx` duty values are typically updated. The `PWMSEG` register may also be updated for other system implementations requiring output crossover.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

PWM Effect Latency

After the PWM registers are configured the effect latency is 1 `PCLK` cycle minimum and 2 `PCLK` cycles maximum.

Debug Features

The module contains four debug status registers (`PWMDBG3-0`), which can be used for debug aid. Each register is available per unit. The registers return current status information about the AH, AL, BH, BL output pins.

Status Debug Register

The module contains four debug status registers (`PWMDBG3-0`), which can be used for debug aid. Each register is available per unit. The registers return current status information about the AH, AL, BH, BL output pins.

Emulation Considerations

An emulation halt does not stop the PWM period counter.

9 MEDIA LOCAL BUS

Media Local Bus (MediaLB®) is an on-PCB or inter-chip communication bus, which allows an application to access the MOST network data. Media Local Bus supports all the MOST network data transport methods including synchronous stream data, asynchronous packet data, control message data and isochronous data. The media local bus topology supports communication among the MLB controller and MLB devices, where the MLB controller is the interface between the MLB devices and the MOST network.

More information on the MediaLB protocol can be found in the MediaLB Device Specification at `www.smsc-ais.com`.

Table 9-1 shows the interface specifications.

Table 9-1. MLB Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	No
SRU DAI Default Routing	N/A
SRU2 DPI Required	No
SRU2 DPI Default Routing	N/A
Interrupt Control	Yes

Table 9-1. MLB Specifications (Cont'd)

Feature	Availability
Protocol	
Master Capable	No
Slave Capable	Yes
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	31
DMA Chaining	No
Boot Capable	No
Local Memory	Yes
Clock Operation	50 MHz

The MLB module in the ADSP-214xx serves as an interface between the MediaLB and ADSP-214xx, implementing the requirements of the physical layer and the link layer outlined in the MediaLB specification. It supports up to 31 logical channels with up to 124 bytes of data per MediaLB frame. Transmit and receive data can be transferred between MediaLB and on-chip memory with single word core-driven transfers or with DMA block transfers.

> The MLB interface supports MOST25 and MOST50 streaming port data rates. Isochronous modes of transfer are not supported.

Features

The MLB device has the following features.

- Support for both 3-pin and 5-pin MediaLB interfaces
- Selectable MediaLB clock rate: 256Fs, 512Fs and 1024Fs
- Support for control, streaming and packet data
- Support for 31 channels, configured for any channel type (synchronous, asynchronous, control) and direction (transmit and receive)
- DMA and core-driven data transport methods
- Memory for channel data buffering
- System channel command handling
- Hardware loop-back test mode support
- Support for transmit command and data transmission
- Support for data reception and receive status response transmission
- Programmable threshold and depth for all buffers
- Support for Big-Endian and Little-Endian data formats
- Support for streaming channel frame synchronization

Pin Descriptions

The MediaLB pin descriptions can be found in the product-specific data sheet.

By default the MLB module is programmed as a 3-pin interface. For more information, see "Device Control Configuration Register (MLB_DCCR)" on page A-93.

Register Overview

The following sections provide brief descriptions of the registers used by the MediaLB interface. For complete bit descriptions, see "Media Local Bus Registers" on page A-93.

Device Configuration and Status Registers

These registers are used to set up the basic features of the interface and to report status of the MLB network. They include the following registers.

- **Device Control Configuration Register (MLB_DCCR).** Controls the device pin connectivity, enable/disable, clock rate, lock status and addressing of the MLB.
- **System Status Register (MLB_SSCR), System Data Configuration Register (MLB_SDCR), System Mask Configuration Register (MLB_SMCR).** Controls system features of the MediaLB interface and system interrupt handling.
- **Synchronous Base Register (MLB_SBCR), Asynchronous Base Register (MLB_ABCR).** Defines the base address for control Rx/Tx system memory buffers.

Channel Registers

These registers are used to configure and monitor individual MLB channels. They include the following registers.

- **Channel Control Registers (MLB_CECRx).** Defines basic attributes about a given logical channel, such as the channel enable, channel type, channel direction, and channel address. The definition of the bit fields in this register vary depending on the selected channel type.
- **Channel Interrupt Status Register (MLB_CICR).** Reflects the channel interrupt status of the individual logical channels. These bits are set by hardware when a channel interrupt is generated. The channel interrupt bits are sticky and can only be reset by software.
- **Channel Status Configuration Registers (MLB_CSCRx).** Reflects the status of the current buffer and previous buffer for a given logical channel. The definition of the bit fields in this register vary dependant on the selected channel type.
- **Channel Current Buffer Configuration Registers (MLB_CCBCRx), Channel Next Buffer Configuration Registers (MLB_CNBCRx), Local Buffer Configuration Registers (MLB_LCBCRx).** These registers allow programs to control and monitor the buffers used in the MLB network.

Clocking

The MLB controller provides an external clock pin—the media local bus clock. This clock is generated by the MLB controller that is synchronized to the MOST network and provides the timing for the entire MLB interface at 49.152 MHz at FS = 48 kHz. The clock to this module may be shut off for power savings. For more information, see “Disabling Peripheral Clocks” on page 23-11.

Functional Description

Figure 9-1 illustrates the MLB high level architecture. The MLB core serves as an interface between the MediaLB and the ADSP-214xx, implementing the requirements of the physical layer and the link layer outlined in the MediaLB specification. The MLB core has the following responsibilities.

- Transmit commands/data operating as transmit device associated with a *Channel Address*.
- Receive data and transmit RxStatus responses when functioning as the receiving device associated with a *Channel Address*.
- MLB lock detection.
- System channel command handling.

The MLB local channel buffer is a single ported RAM which implements the local channel buffering for the MLB device. It is 36 bits wide and 124 words long.

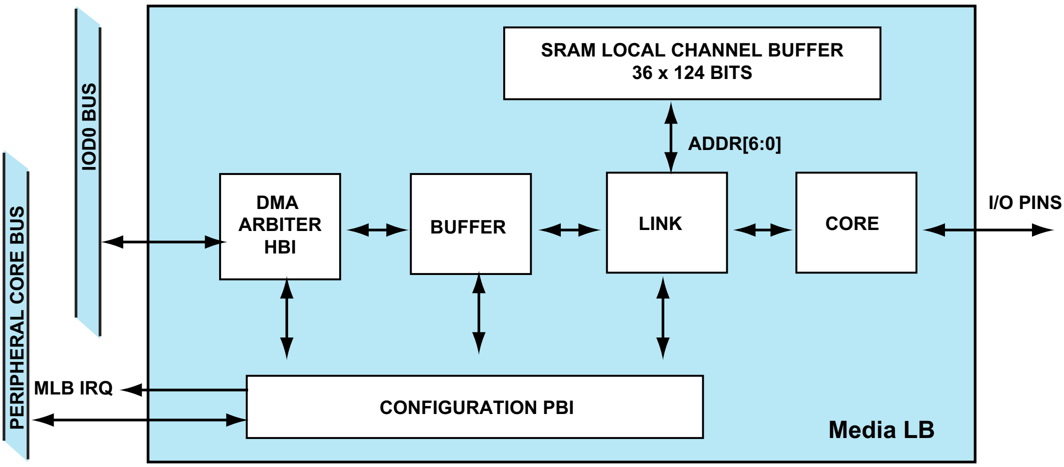

Figure 9-1. Media LB Block Diagram

Media LB Protocol

Once per MOST network frame, the MLB controller generates a unique frame sync pattern on the MLBSIG line. The end of the frame sync pattern defines the byte boundary and the channel boundary for the MLBSIG and MLBDAT lines of all MLB devices.

The MLB controller manages the arbitration for all the channels on the MLB and grants bandwidth for all the MLB devices. A MLB *physical channel* is defined as four bytes wide, or a quadlet. Physical channels can be grouped into multiple quadlets (which do not have to be consecutive) to form a MLB *logical channel*, which is defined by a unique channel address.

As shown in Figure 9-2, the MLB controller initiates communication by sending out a channel address on the MLBSIG line for each physical channel. The channel address indicates which MLB device is transmitting and which MLB devices are receiving in the following physical channel. Therefore, four bytes after the controller outputs the channel address on the MLBSIG line, the transmitting device outputs a command byte command on the MLBSIG line and outputs the respective data on the MLBDAT line, concurrently. The MLB command byte contains the type of data currently being transmitted (for example synchronous, asynchronous or control).

The MLB device receiving the channel data outputs a status byte, RxStatus, on the MLBSIG line immediately after the transmitting device outputs the command byte. The status response can indicate that the receiving device is busy and cannot receive the data at present, or the device is ready to receive the data. Since synchronous stream data is sent in a broadcast fashion, receiving devices cannot return a busy status and should not drive RxStatus onto the MLBSIG line.

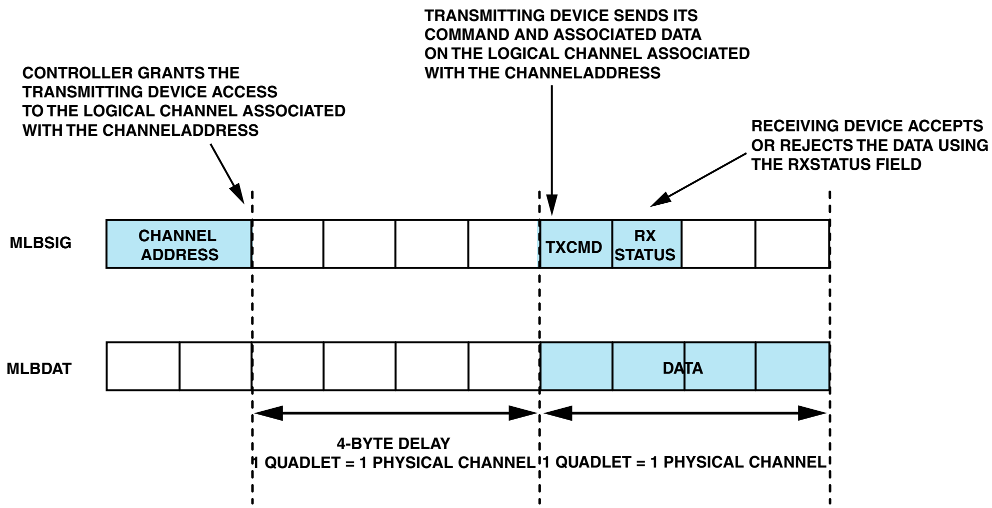

Figure 9-2. MLB Data Structure

Operating Modes

The following sections describe the operating modes of the MLB interface. The channel type selection enables the logical channels to operate in synchronous, asynchronous or control channels described here.

ⓘ The logical channels can be any combination of channel type (for example synchronous, asynchronous, or control) and direction (transmit or receive).

Logical Channel Control

Table 9-2 provides an overview of the control register settings for the logical channels (`MLB_CECR30-0`). The `CTYPE` bits (bits 28 and 29) are used to configure whether the channel is synchronous, asynchronous, or control.

Table 9-2. MLB_CECRx Register Bit Settings

Bit	Synchronous Channel	Asynchronous/Control Channel
7–0	CA (channel address)	
12-8	FSPC (frame sync physical channel count)	PCTH (packet count threshold, I/O mode only)
14–13	Reserved	
15	FSCD (frame sync channel disable)	Reserved
16	Reserved	IMASK0 (protocol error)
17	Reserved	IMASK1 (break detect)
18	IMASK2 (IO RX service request/DMA buffer done)	
19	IMASK3 (IO TX service request/DMA buffer start)	
20	IMASK4 (buffer error)	
21	Reserved	
22	IMASK6 (lost frame sync)	
23–24	Reserved	
26–25	MDS (channel mode select)	
27	FSE (frame sync enable)	PCE (packet count enable, I/O mode only)
29–28	CTYPE = synchronous	CTYPE = asynchronous/control
30	CTRAN (channel data direction)	
31	CE (channel enable)	

Synchronous Channels

Synchronous channels are enabled if the channel type select CTYPE bits in the MLB_CECRx register are configured accordingly (default).

Synchronous channel is used for real time streaming data which are continuously synchronous to the MOST network.

Certain types of streaming applications require data to be synchronous with the MediaLB frame, including: stereo, 5.1 audio, and generic synchronous packet format (GSPF) DTCP. This feature is provided as an optional programmable synchronous logical channel by setting the FSE bit in the MLB_CECRx register. When enabled, the synchronous logical channel begins transmitting data only at a MediaLB frame boundary. A maskable interrupt is generated when the loss of frame synchronization occurs.

In order to use this option, system software must program the FSPC bits in the MLB_CECRx register with the expected number of physical channels per frame for the logical channel, and unmask the STS interrupt bit (bit 6) in the MLB_CSCRx register by setting the MASK bit in MLB_CECRx register. An interrupt is generated when the actual number of physical channels detected during a MLB frame does not match the expected value. Software can also set the FSCD bit in MLB_CSCRx register, which causes hardware to automatically disable a logical channel (clear the channel enable bit in the MLB_CECRx register) when synchronization is lost.

Asynchronous Channels

Asynchronous channels are enabled if the channel type select CTYPE bits in the MLB_CECRx register are configured accordingly. Asynchronous channels are used for packed data which is packetized and is transferred in bursts, as foe example with internet, GPS map and e-mail.

Frame synchronization is not supported for asynchronous channels.

Control Channels

Control channels are enabled if the channel type select CTYPE bits in the MLB_CECRx register are configured accordingly. Control channels are used for data containing control/diagnostic and status information of devices on the MOST network.

Frame synchronization is not supported for control channels.

Data Transfer

Two modes of operation are supported for transferring channel data between the MLB and internal memory. DMA allows the multi-channel DMA engine to manage data transfers without core intervention. Core driven mode (I/O mode) allows software to manage the transfer of data between MLB and internal memory.

All hardware channels must use the same data transfer method. Mixed mode operation where hardware channels operate in both I/O mode and DMA mode is not supported.

Core Access

Core driven mode is an interrupt driven data transfer method between hardware channels and internal memory. Core mode and data direction are configured by the channel mode select (MDS) bits and the CTRAN bit in the MLB_CECRx register.

Transmit and receive service request interrupts are generated when data is to be transferred from/to internal memory to/from MLB local channel buffer.

In IO mode the MLB_CCBCRx register is used as the receive buffer and the MLB_CNBCRx register as the transmit buffer.

Threshold Depth

A single ported SRAM implements the local channel buffer for the Media LB device. Its capacity is 36-bits x 124 quadlets (words). The local channel buffer configuration register (MLB_LCBCRx) allows software to optimize use of the local channel buffer memory. The buffer depth, buffer threshold and buffer start address for each channel is programmed using the respective register.

As shown in Figure 9-3, once a quadlet is received on the MLB bus, it is transferred to the corresponding local channel buffer. The data transfer takes place from this local channel buffer to internal memory. After reset each channel has four quadlets each.

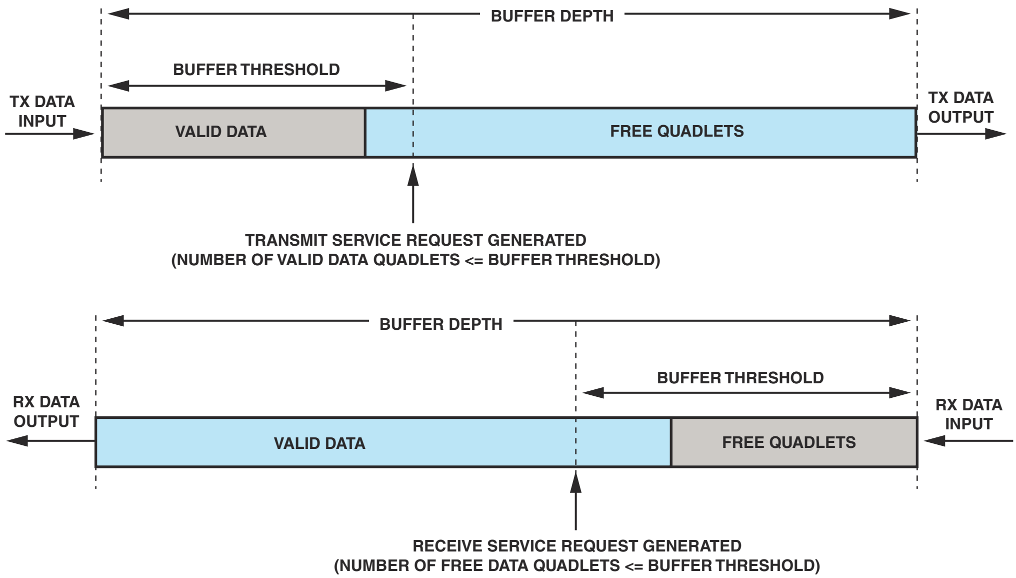

Figure 9-3. Local Channel Buffer Threshold Mechanism

The byte order in which data is transferred between local channel buffers and internal memory is determined by enabling either Big-Endian or Little-Endian mode. Both data transfer methods, DMA and IO mode support Big-Endian and Little-Endian system memory data formats.

The buffer depth, buffer threshold and buffer start address for each channel is programmed using the respective `MLB_LCBCRx` register.

> The local buffer threshold is used by hardware channels in I/O mode only. Hardware uses the local channel buffer threshold to determine when to issue an I/O service request to system software.

Hardware uses the local channel buffer threshold to determine when to issue an I/O service request to system software. I/O service requests are generated when:

- The number of valid quadlets in the local channel buffer falls below the threshold for transmit channels or:
  valid quadlets <= MLB_LCBCRx (programmed using TH bits)
- The number of free quadlets in the local channel buffer falls below the threshold for receive channels or:
  free quadlets <= MLB_LCBCRx (programmed using TH bits)
- A receive channel detects a broken packet (ReceiverBreak, AsyncBreak, ControlBreak or ReceiverProtocolError).

Configuring local channel buffer memory is accomplished using the MLB_LCBCRx register. For more information, see "Programming Model" on page 9-20.

Status

The local channel buffer is available by reading the buffer MLB_CSCRn register. The full/empty local buffer threshold status is depending on the MLB_LCBCRx register settings.

Flushing the Buffer

The MLB local buffer is only flushed by a system hard ($\overline{\text{RESET}}$) or software reset (SYSCTL).

DMA

There are two modes of DMA—Ping-pong buffering and circular buffering.

There are 31 DMA channels for the 31 logical channels. Each channel can address up to 16k words.

The DMA address is comprised of:

- A 5-bit base configured in the MLB base register set (MLB_SBCR, MLB_ABCR, MLB_CBCR or MLB_CBCR for the corresponding channel data type). The 5-bit base is the same for all the channels of the same type (for example for all synchronous RX channels MLB_SB-CR31-16 act as the base).

- A 14-bit offset configured using the BCA bits in the MLB_CCBCRx register. The register holds the lower 14 bits (bits 31–18 for start address and 15–2 for end address). Bits 17–16 and bits 1–0 are reserved and must always be written with zero.

For example, if the internal address is 0xC0100, then 19 bits of the address translates to address 0x40100 because the internal memory offset for the ADSP-214xx is 0x0008 0000.

The lower 14 bits (00 0001 0000 0000) and the 2 reserved bits “00” are written in the MLB_CNBCRx register (write 0000 0100 0000 0000 = 0x0400 to bits 31–16 in the MLB_CNBCRx register for the start address or bits 15–0 for the end address). The remaining higher 5 bits (1 0000) are written in one of the base address registers, depending on the transfer mode. For example, if using synchronous mode, write bits 31–16 = 0x0010 for receive and bits 15–0 for transmit in the MLB_SBCR register.

MLB DMA Group Priority

The MLB module has up to 31 DMA channels. When multiple channels have data ready, the channel arbitrates using a rotating round robin method (first arbitration stage). The winning channel requests the DMA bus arbiter to get control of the peripheral DMA bus (second arbitration stage).

For the I/O processor, the 31 DMA channels are considered as a group and one arbitration request. For more information, see “Peripheral DMA Arbitration” on page 3-36.

Ping-Pong DMA

Logical channels operate in core mode when the channel mode select MDS bits in the MLB_CECRx register are configured accordingly (default).

When MLB is configured in ping-pong mode, the MLB_CCBCRx and MLB_CNBCRx registers are used to configure and monitor the system memory current buffer (ping) and next buffer (pong) respectively. Ping-pong DMA is available for all data types.

When receiving and transmitting asynchronous and control packet data, the current buffer and next buffer are independent internal memory buffers. This allows hardware to support the ping-pong buffering. Each buffer is addressed using two 16-bit address fields in the MLB_CNBCRx and MLB_CCBCRx registers as described below.

- Current buffer address (BCA bits, CCBCRx register) – start of current buffer in internal memory.
- Current buffer final address (BFA bits, CCBCRx register) – end of current buffer in internal memory.
- Next buffer start address (BSA bits, CNBCRx register) – start of next buffer in internal memory.
- Next buffer end address (BEA bits, CNBCRx register) – end of next buffer in internal memory.

For ping-pong DMA mode, transmit and receive for all data types are handled in the following manner.

- At the start of buffer processing, the beginning of the next buffer becomes the beginning of the current buffer, as the BSA bits from the MLB_CNBCRx register are loaded into the BCA bit field of the MLB_CCBCRx register. Additionally, the end of the next buffer becomes the end of the current buffer, as the BEA bit field from the MLB_CNBCRx register is loaded into the BFA bit field of the MLB_CCBCRx register.

- A current buffer start interrupt is generated (STS bit in the MLB_CSCRx register), which informs the software that hardware has updated the MLB_CCBCRn register, cleared the local channel RDY bit, and is available to accept the next buffer. Software may then prepare the next buffer by writing: BSA, BEA, and RDY bits.
- During the processing of the current buffer, BCA bits continue to mark which quadlet of data or packet is currently being processed.
- A current buffer done interrupt is generated when the last quadlet in the current buffer has been successfully transmitted/received.

The current buffer and the next buffer can be configured for either multi-packet or single-packet buffering, when receiving and transmitting asynchronous and control packet data.

- Multi-packet buffering allows the system to reduce the interrupt load at the expense of larger DMA buffers.
- Single-packet buffering allows DMA buffer size to be reduced at the expense of increasing the interrupt rate.

For more information, see “Programming Model” on page 9-20.

Circular Buffer DMA

Logical channels operate in circular buffer DMA mode when the channel mode select bits in the MLB_CECRx register are configured accordingly. In contrast to ping-pong buffering, circular buffering uses a single, circular memory buffer to process channel data.

Circular buffer DMA is available for synchronous channels only.

For circular buffer mode, synchronous data is handled in the following manner:

- Before buffer processing can begin, the BSA bits in the MLB_CNBCRx register and the BEA bits in the MLB_CNBCRx register should be programmed with the beginning and the ending address of the circular buffer. Set the RDY bit in the MLB_CSCRx register to initiate buffer processing.
- At the start of buffer processing, the beginning address of the circular buffer (BSA) is loaded into BCA field of the MLB_CCBCRx register. Additionally, the ending address of the circular buffer (BEA bits) is loaded into the BFA bit field of the MLB_CCBCRx register.
- During the processing of the circular buffer, the BCA bits are updated to indicate which quadlet of the synchronous data is currently being processed.
- Once the end of the buffer is reached and BCA = BFA, the BCA field is reloaded to point to the beginning address of the circular buffer (BSA).

Unlike in ping-pong DMA, the RDY bit remains set during the processing of the circular buffer DMA. Software must clear this bit to halt buffer processing. For more information, see “Programming Model” on page 9-20.

Interrupts

Table 9-3 provides an overview of MLB interrupts.

Table 9-3. Overview of MLB Interrupts

Default Programmable Interrupt	Sources	Masking	Service
MLBI not connected by default	MLB Lock/Unlock Network Lock/Unlock Sub command Reset	Unmask MLB_SMCR	RW1C to MLB_SSCR + RTI instruction
	Core buffer service DMA start DMA complete Break detect Lost frame sync Buffer error RX protocol error	Unmask MLB_CECRx	RW1C to MLB_CSCRx + RTI instruction

Sources

The MLB module generates a total of 38 local interrupts which are grouped into seven system status and 31 logical channel status interrupts. All 38 signals are logically ORed into 1 MLB interrupt signal which must be routed into a programmable interrupt.

The MLB interface generates interrupts as described in the following sections.

Core Buffer Service Request

When I/O mode is enabled and the processor core reads from the receive buffer (`MLB_CCBCRx` register) or writes to the transmit buffer (`MLB_CNBCRx` register) an interrupt is generated when the local channel buffer is not empty or not full respectively.

Threshold Transmit Request

In I/O mode the logical channel interrupt requests are generated when the number of valid quadlets in the local channel buffer falls below the threshold for transmit channels or: valid quadlets ≤ MLB_LCBCRX (programmed using the TH bits).

Threshold Receive Request

For I/O mode the logical channel interrupt requests are generated when the number of free quadlets in the local channel buffer falls below the threshold for receive channels or: free quadlets ≤MLB_LCBCRx (programmed using the TH bits).

DMA Complete

When ping-pong or circular mode is enabled the DMA complete is generated after the count is zero.

Receive Channel Errors

A receive channel detects errors such as:

- Receiver Break,
- Asynchronous Break,
- Control Break
- Receiver Protocol Error

Masking

The MLBI signal is not routed by default to programmable interrupts. To service the MLBI, unmask (set = 1) any programmable interrupt bit in the IMASK/LIRPTL register.

The system status interrupts are unmasked by setting the corresponding bits in the MLB_SMCR register.

The logical channel status interrupts are unmasked by setting the corresponding bits in the MLB_CECR30-0 register. To enable core buffer service request the MDS bit must be configured for I/O mode. To enable DMA interrupts the MDS bit must be configured for ping-pong or circular buffering mode.

Service

For system status interrupts RW1C to the corresponding bit in the MLB_SSCR register (except for the SSRE bit which is cleared by hardware).

The global interrupt channel status register (MLB_CICR) reflects the global status of 31 logic channels together. Reading identifies the logical channel. RW1C the corresponding status bit in the local channel register (MLB_CSCRx) which also clear its MLB_CICR bits.

Programming Model

The following sections provide procedures that are helpful when programming media local bus interface.

I/O Interrupt Mode

To configure the MLB interface for I/O mode using interrupts, use the following procedure.

1. Reset the MLB device.
2. Program the appropriate bits in the PICRx register to generate MLB interrupt.

3. Unmask the appropriate bits in the MLB_SSCR register in order to monitor the MLB network.

4. Configure the MLB control register (MLB_DCCR) with the appropriate settings and enable the MediaLB device.

5. Check for MLB lock using the status bit in the MLB_SSCR register using polling or interrupt.

6. Configure the MLB_LCBCRx register for channel buffer threshold, depth and start address.

7. Configure the logical channel using the MLB_CECRx register for I/O mode, transfer direction, channel type, channel address and interrupt generation.

   For a transmit, a transmit service request, (STS bit 1 in the MLB_CSCRx register) an interrupt is generated when the local channel buffer can accept data. Within the ISR, check if this status bit is set. If set, write the data into transmit data buffer (MLB_CNBCRx).

   For a receive, a receive service request, (STS bit 2 in the MLB_CSCRx register) an interrupt is generated when the local channel buffer has data to be read. Within the ISR, check if this status bit is set. If set, read the data from the receive data buffer (MLB_CCBCRx).

8. Clear all interrupts by writing 0x0000FFFF to the MLB_CSCRx register.

DMA Modes

MLB channels can be configured for circular buffer DMA mode by programming the channel mode select bits (MLB_CECRx register, bits 25–26 = 01) for synchronous channels only. In contrast to DMA mode with ping-pong buffering, circular buffering uses a single, circular memory buffer to process channel data.

To configure a ping-pong or circular buffered DMA, use the following procedure.

1. Reset the MLB device.
2. Program the appropriate bits in the PICRx register to generate MLB interrupt.
3. Unmask the appropriate bits in the MLB_SSCR register in order to monitor the MLB network.
4. Configure the MLB control register (MLB_DCCR) with the appropriate settings and enable the MediaLB device.
5. Configure the base address register (MLB_SBCR, MLB_ABCR or MLB_CBCR) based on the data type configured for the logical channel.
6. Check for MLB lock using the status bit in the MLB_SSCR register using polling or interrupt.
7. Configure the MLB_LCBCRx register for channel buffer threshold, depth and start address.
8. Configure the logical channel using the MLB_CECRx register for ping-pong or circular buffer DMA mode, transfer direction, channel type, channel address and also to generate appropriate interrupts.
9. Configure the MLB_CNBCRx register with the buffer start and end address.
10. Set the RDY bit in the MLB_CSCRx register to start the DMA.

    Hardware automatically the clears the RDY bit ping-pong DMA but not for circular buffer DMA. Therefore, for circular buffer DMA, this bit should be cleared manually by the software to stop buffer processing.

11. An interrupt is generated depending on the bit unmasked in the MLB_CECRx register. Within the ISR check that the appropriate status bit (in the MLB_CSCRx register) is set.

12. Clear all interrupts by writing 0x0000FFFF to the MLB_CSCRx register.

Debug Features

The following sections provide information to assist in MediaLB debug.

Loop-Back Test Mode

Loop-back test mode is used for debug operations and is enabled by setting the LBM bit in the MLB_DCCR register. This mode provides basic testing capabilities for the MediaLB pads, physical layer, link layer, channel protocol and the local channel buffer by enabling a single receive channel and a single transmit channel. When loop-back test mode is enabled, a data path is enabled which allows receive data from even channel N to be sent out as transmit data on channel N + 1, where N = {0, 2, 4 …30}.

10 DIGITAL APPLICATION/ DIGITAL PERIPHERAL INTERFACES

The digital application interface (DAI) and the digital peripheral interface (DPI) are comprised of a groups of peripherals and their respective signal routing units (SRU and SRU2). The inputs and outputs of the peripherals are not directly connected to external pins. Rather, the SRUs connect the peripherals to a set of pins and to each other, based on a set of configuration registers. This allows the peripherals to be interconnected to suit a wide variety of systems. It also allows the SHARC processors to include an arbitrary number and variety of peripherals while retaining high levels of compatibility without increasing pin count.

The routing unit specifications are listed in Table 10-1.

Table 10-1. Routing Unit Specifications

Feature	DAI	DPI
Pin Buffers		
Number	20	14
Input	Yes	Yes
Output	Yes	Yes
Open-drain	Yes	Yes
Polarity Change	Yes	No
High Impedance	Yes	Yes
Programmable Pull-up	No	No
I/O Level Status Register	Yes	Yes

Table 10-1. Routing Unit Specifications (Cont'd)

Feature	DAI	DPI
Local Memory	No	No
Max Clock Operation	$f_{PCLK}/4$	$f_{PCLK}/4$

SRU Features

In a typical processor, static (multiplexed) pins are assigned to specific peripherals. When certain peripherals are not required for an application, these pins are unnecessary and expensive because they may need to be defined as high/low to prevent any illegal conditions.

The signal routing unit on the SHARC processors addresses this by controlling a number of "general-purpose pins" which can be assigned flexibly (a *virtual connectivity* between peripherals) depending on system requirements. This virtual connectivity includes pin buffers and routing logic (multiplexer) and offers the following advantages.

- Flexibility – connections can be made via software and during run-time, no hard-wiring is required.
- Control is provided via memory-mapped control registers organized in groups. This is useful for interconnects (for example clocks, frame sync, data).
- At reset a default routing scheme is already programmed including boot enabled peripherals.
- Connectivity can be internally between peripherals, externally between pin buffers, or a mix of both.
- Status of the pin buffers can be programmed for conditional execution or interrupts.
- Some pin buffers allow control of signal polarity changes.

- For shared bus systems such as SPI or TWI, open drain configuration possible.
- No fan-out limitation, a peripheral/pin buffer output can be routed to multiple peripheral/pin buffer inputs.
- Two independent routing systems are available— the DAI and the DPI. Signals can't be interconnected between both routings units with exception of the precision clock generator (PCG).

Analog Devices offers macros that are included with the CrossCore or VisualDSP++ tools, greatly easing code development in the SRU.

Register Overview

The SRU for the DAI contains six register sets that are associated with the DAI groups.

Clock Routing Registers (SRU_CLKx). Associated with Group A, routes clock signals.

Serial Data Routing Registers (SRU_DATx). Associated with group B, routes data.

Frame Sync Routing Control Registers (SRU_FSx). Associated with group C, routes frame syncs or word clocks to the serial ports, the SRC, the S/PDIF, and the IDP.

Pin Signal Assignment Registers (SRU_PINx). Associated with group D, routes physical pins (connected to a bonded pad).

Miscellaneous Signal Routing Registers (SRU_MISCx). Associated with group E, allows programs to route to the DAI interrupt latch, PBEN input routing, or input signal inversion.

DAI Pin Buffer Enable Registers (SRU_PBENx). Associated with group F, Activate the drive buffer for each of the 20 DAI pins.

DAI Shift Registers Clock (SRU_CLK_SHREG). Associated with group H, routes all shift register clock signals (ADSP-2147x).

DAI Shift Registers Data (SRU_DAT_SHREG). Associated with group I, routes all shift register serial data signals (ADSP-2147x).

The SRU2 for DPI contains three register sets associated with the DPI groups.

Miscellaneous Signal Routing Registers (SRU2_INPUTx). Associated with group A, used to route the 14 external pin signals to the inputs of the other peripherals.

Pin Assignment Signal Routing (SRU2_PINx). Associated with group B routes pin output signals to the DPI pins.

Pin Enable Signal Routing (SRU2_PBENx). Associated with group C used to specify whether each DPI pin is used as an output or an input by setting the source for the pin buffer enable.

The DAI/DPI registers are unique in that they work as groups to control other peripheral functions. The register groups and routings are described in detail in "DAI/DPI Group Routing" on page 10-18, "DAI Signal Routing Unit Registers" on page A-124 and "DPI Signal Routing Unit Registers" on page A-209.

Clocking

The fundamental timing clock of the DAI/DPI modules is peripheral clock/4 (PCLK/4). The clock to the DAI may be shut off for power savings. For more information, see "Disabling Peripheral Clocks" on page 23-11.

Functional Description

Figure 10-1 and Figure 10-2 shows how the DAI/DPI pin buffers are connected via the SRU/SRU2. This allows for very flexible signal routing.

The DAI/DPI are comprised of four primary blocks:

- Peripherals (A/B/C) associated with DAI/DPI
- Signal Routing Units (SRU, SRU2)
- DAI/DPI I/O pin buffers
- Miscellaneous buffers

The peripherals shown in Figure 10-1 and Figure 10-2 can have up to three connections (if master or slave capable); one acts as a signal input, one as a signal output and the 3rd as an output enable. The SRUs are based on a group of multiplexers which are controlled by registers to establish the desired interconnects. The DAI/DPI pin buffers have three signals which are used for input/output to/from off-chip and the 3rd for output enable.

The miscellaneous buffers have an input and an output and are used for group interconnection.

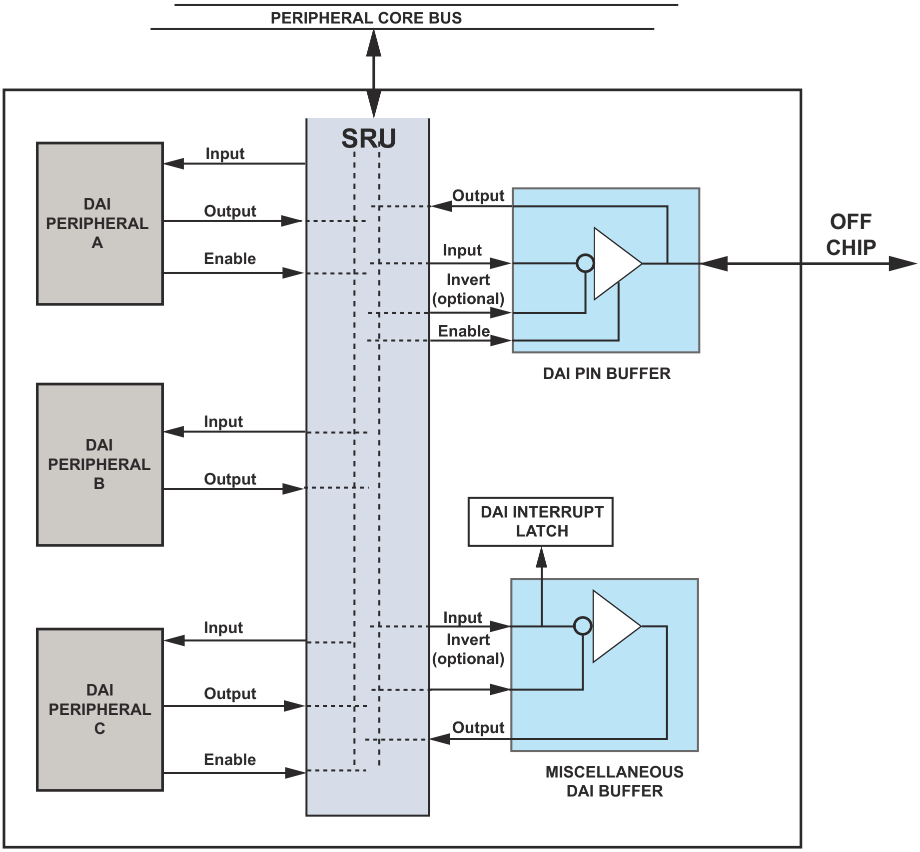

Figure 10-1. DAI Functional Block Diagram

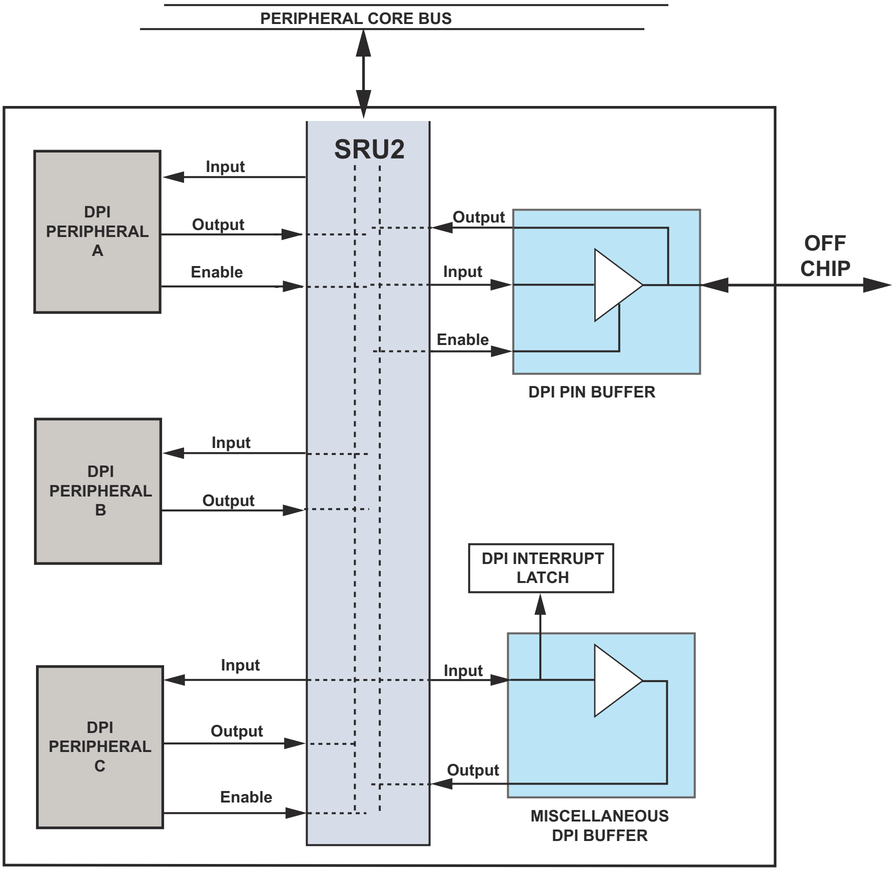

Figure 10-2. DPI Functional Block Diagram

Note that the figures are simplified representation of a DAI and DPI system. In a real representation, the SRU and DAI would show several types of data being routed from several sources including the following.

- Serial ports (SPORT)
- Precision clock generators (PCG)
- Input data port (IDP)

- Asynchronous sample rate converters (SRC)
- S/PDIF transmitter
- S/PDIF receiver
- Shift register
- DAI interrupts (miscellaneous)

Similarly, the DPI pin buffers are connected via the SRU2. The DPI makes use of several types of data from a large variety of sources, including:

- Peripheral timers
- Serial Peripheral Interfaces (SPI)
- Precision clock generators (PCG)
- Universal asynchronous Rx/Tx ports (UART)
- Two-wire interface (TWI)
- GPIO flags (external port)
- DPI interrupts (miscellaneous)

Note that the precision clock generator (units C/D) can be assigned to access DAI and/or DPI pins.

DAI/DPI Signal Naming Conventions

The peripherals associated with the DAI/DPI do not have any dedicated I/O pins for off-chip communication. Instead, the I/O pin is only accessible in the chip internally and is known as an *internal node*. Every internal node of a DAI peripheral (input or output) is given a unique mnemonic. The convention is to begin the name with an identifier for the peripheral that the signal is coming to/from followed by the signal's function.

A number is included if the DAI contains more than one peripheral type (for example, serial ports), or if the peripheral has more than one signal that performs this function (for example, IDP channels). The mnemonic always ends with _I if the signal is an input, or with _O if the signal is an output (Figure 10-3).

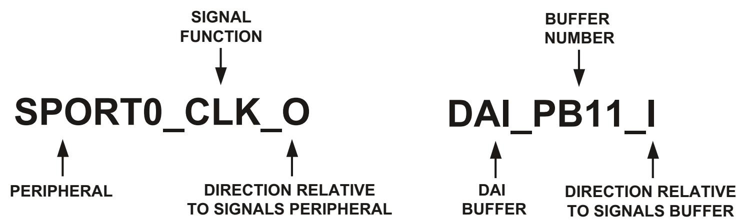

Figure 10-3. Example SRU Mnemonics

I/O Pin Buffers

Within the context of the SRU, physical connections to the DAI pins are replaced by a logical interface known as a *pin buffer*. This is a three terminal active device capable of sourcing/sinking output current when its driver is enabled, and passing external input signals when disabled. Each pin has an input, an output, and an enable as shown in Figure 10-4. The inputs and the outputs are defined with respect to the pin, similar to a peripheral device. This is consistent with the SRU naming convention.

Pin Buffer Signals

The pin buffer is based on three signals shown in Figure 10-4 described in the following sections.

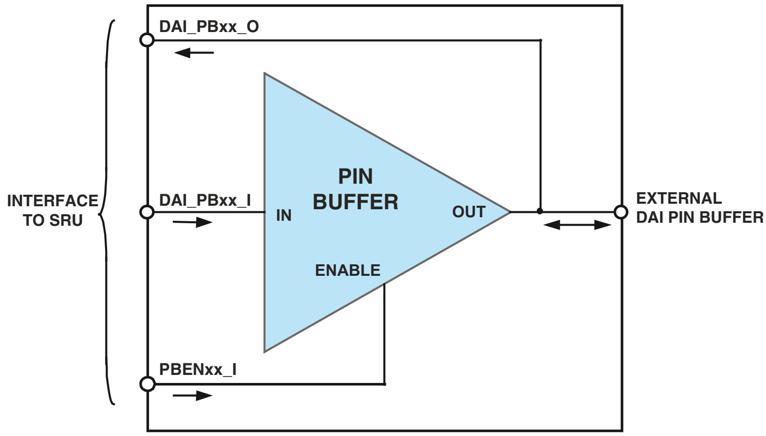

Figure 10-4. Pin Buffer Example

Pin Buffer Input Signal

A pin buffer input (`DAI_PBxx_I`, `DPI_PBxx_I`) is driven as an output from the processor when the pin buffer enable is set (=1). Each physical pin (connected to a bonded pad) may be connected via the SRU to any of the outputs of the DAI/DPI peripherals, based on the bit field values. The SRU also may be used to route signals that control the pins in other ways. Many signals may be configured for use as control signals.

Pin Buffer Enable Signal

When a pin buffer enable (`PBENxx_I`) is set (=1), the signal present at the corresponding pin buffer input (`PBxx_I`) is driven off-chip as an output. When a pin buffer enable is cleared (=0), the signal present at the corresponding pin buffer input is ignored. The pin enable control registers activate the drive buffer for each of the DAI/DPI pins. When the pins are not enabled (driven), they can be used as inputs. There are two options to control the pin buffer enable signal; setting the level high for a static solution, or connecting the dedicated peripheral's pin buffer output enable signal to its pin buffer, which automatically enables the pin buffer.

Pin Buffer Functions

Pin buffers may be configured as inputs, outputs or as open drain as described in the following sections.

Pin Buffers as Signal Input

When the DAI pin is to be used only as an input, connect the corresponding pin buffer enable to logic low as shown in Figure 10-5. This disables the buffer amplifier and allows an off-chip source to drive the value present on the DAI pin and at the pin buffer output. When the pin buffer enable (PBENxx_I) is cleared (= 0), the pin buffer output (PBxx_O) is the signal driven onto the DAI pin by an external source, and the pin buffer input (PBxx_I) is not used.

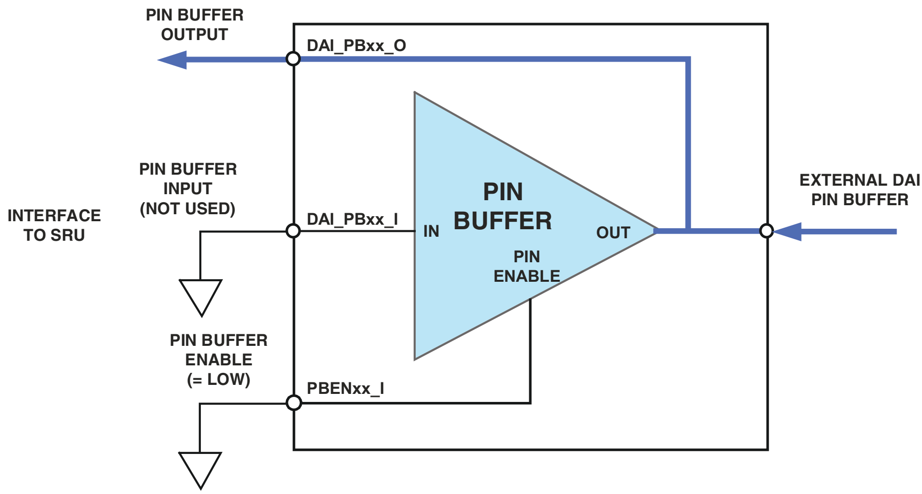

Figure 10-5. Pin Buffer as Input

Whether programmed as input or output, a DAI/DPI buffer input always routes the same signal to an output internally.

Pin Buffers As Signal Output

In a typical embedded system, most pins are designated as either inputs or outputs when the circuit is designed, even though they may have the ability to be used in either direction. Each of the DAI pins can be used as either an output or an input. Although the direction of a DAI pin is set simply by writing to a memory-mapped register, most often the pin's direction is dictated by the designated use of that pin.

For example, if the DAI pin were to be hard wired to only the input of another interconnected circuit, it would not make sense for the corresponding pin buffer to be configured as an input. Input pins are commonly tied to logic high or logic low to set the input to a fixed value. Similarly, setting the direction of a DAI pin at system startup by tying the pin buffer enable to a fixed value (either logic high or logic low) is often the simplest and cleanest way to configure the SRU.

When the DAI pin is to be used only as an output, connect the corresponding pin buffer enable to logic high as shown in Figure 10-6. This enables the buffer amplifier to operate as a current source and to drive the value present at the pin buffer input onto the DAI pin and off-chip. When the pin buffer enable (PBENxx_I) is set (=1), the pin buffer output (PBxx_O) is the same signal as the pin buffer input (PBxx_I), and this signal is driven as an output.

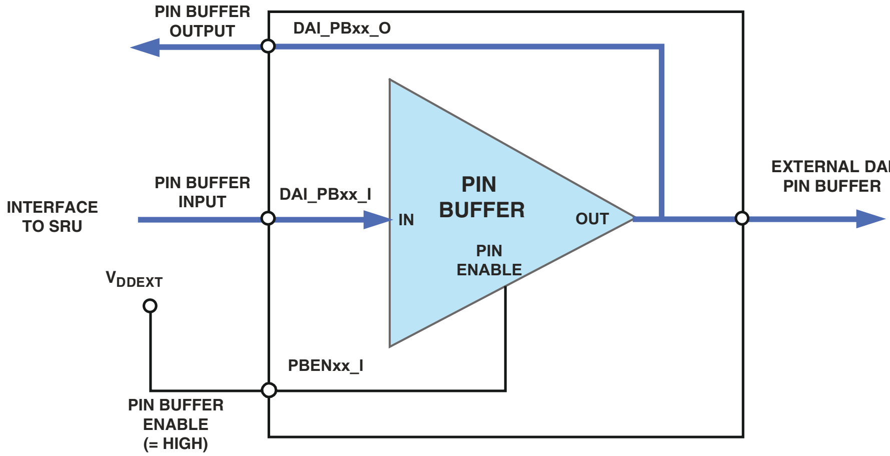

Figure 10-6. Pin Buffer as Output

Pin Buffers as Open Drain

Certain peripherals (for example the TWI or SPI) may be required to run in multiprocessing environments. These peripherals will need their pin drivers to work in open drain mode for transmit and receive operation as shown in Figure 10-7. The signal input of the assigned pin buffer is tied low. The peripheral's data output signal is connected to the `PBEN` signal. In open drain mode, if `PBEN` = low, the level on the pin depends on the bus activities. If `PBEN` = high, the driver is conducting (input always low level) and ties the bus level low. Note that for the SPI the `ODP` bit in the `SPICTL` register must be enabled.

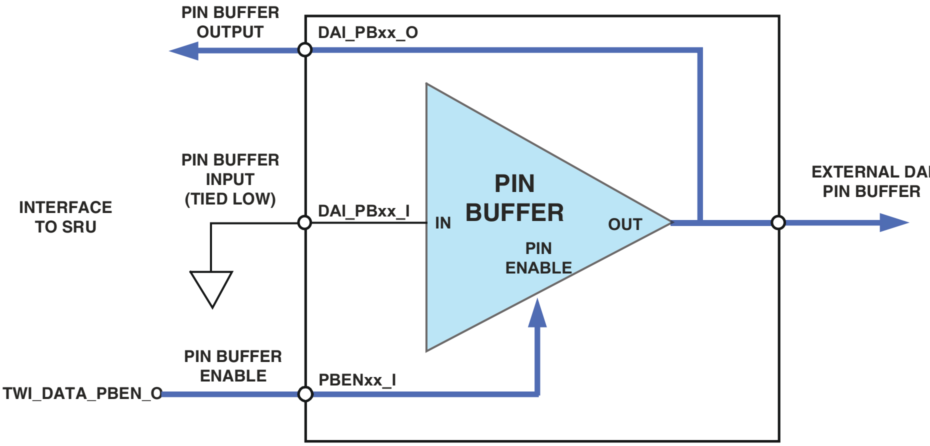

Figure 10-7. Pin Buffer as Open Drain

DAI/DPI Pin Buffer Status

The signal levels on the DAI/DPI pins can be read with the `DAI/DPI_PIN_STAT` registers. This allows conditions like for example:

```
ustat2=dm(DAI_PIN_STAT);
bit tst ustat2 DAI_PB10;
if TF jump DAI_PB10_high;
```

DAI/DPI Peripherals

There are two categories of peripherals associated with the DAI and DPI. These are described in the following sections.

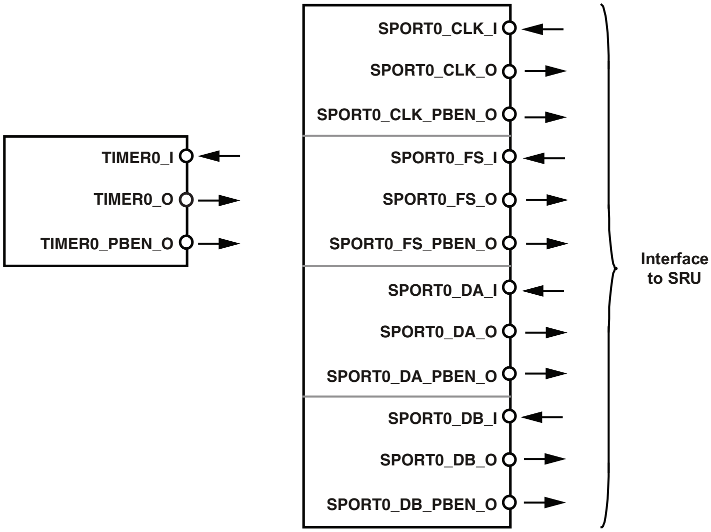

Figure 10-8. SRU Connections for Timer and SPORT0

Output Signals With Pin Buffer Enable Control

Many peripherals within the DAI/DPI that have bidirectional pins generate a corresponding pin enable signal. Typically, the settings within a peripheral's control registers determine if a bidirectional pin is an input or an output, and is then driven accordingly.

Though most peripherals are capable of operating bidirectionally, it is not required that all of a peripheral's `_I` and `_O` signals should be connected to the pin buffer. If the system design only uses a signal in one direction, it is simpler just to connect the pin buffer accordingly.

All available pin buffer output enables must be routed to their pin buffer input enable signals in cases where data streaming connections are used. This will guarantee timing requirements (for example a gated clock for the SPI).

Output Signals Without Pin Buffer Enable Control

Some peripherals have signal outputs without automated pin buffer control enable as shown in Figure 10-9 (PDAP_STRB_O, MISCx_O, BLK_START_O).

The operation of these peripherals is simplified as the routing to a DAI/DPI pin buffer enable input requires a static high from the SRU. In order to disable the pin buffer output, software must clear the pin buffer enable input accordingly.

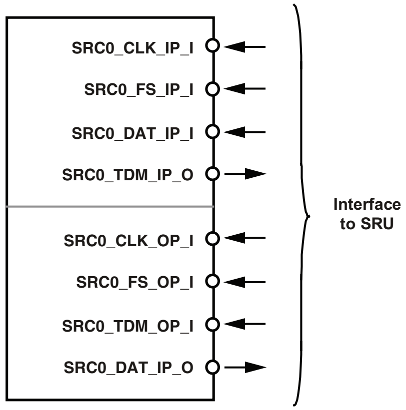

Figure 10-9. SRU Connections (no PBEN Signals)

Signal Routing Units (SRUs)

The following sections provide more detail specific to the SRUs.

Signal Routing Matrix by Groups

The SRU can be likened to a set of patch bays, which contains a bank of inputs and a bank of outputs. For each input (destination), there is a set of permissible output (source) options. Outputs can feed to any number of inputs in parallel, but every input must be patched to exactly one valid output source. Together, the set of inputs and outputs are called a group. The signal's inputs and outputs that comprise each group all serve similar purposes. They are compatible such that almost any output-to-input patch makes functional sense. With the grouping, the multiplexing scheme becomes highly efficient since it wouldn't make sense (for example) to route a frame sync signal to a data signal.

The SRU for the DAI contains six groups named A through F; the ADSP-2147x alone has two additional groups. Each group routes a unique set of signals with a specific purpose as shown below.

- Group A routes clock signals
- Group B routes serial data signals
- Group C routes frame sync signals
- Group D routes pin signals
- Group E routes miscellaneous signals
- Group F routes pin output enable signals
- Group H routes all shift register clock signals (ADSP-2147x only)
- Group I routes all shift register data signals (ADSP-2147x only)

Together, the SRU groups include all of the inputs and outputs of the DAI peripherals, a number of additional signals from the core, and all of the connections to the DAI pins.

The SRU2 for DPI contains three groups that are named sequentially A through C. Each group routes various signals with a specific purpose:

- Group A routes miscellaneous signals
- Group B routes pin output signals
- Group C routes pin output enable signals

Unlike the SRU in the DAI module, all types of functionality, such as clock and data, are merged into the same group in the DPI SRU2.

Note that it is not possible to connect a signal in one group directly to signal in a different group (analogous to wiring from one patch bay to another). However, group D (DAI) or group B (DPI) is largely devoted to routing in this vein.

DAI/DPI Group Routing

Each group has a unique encoding for its associated output signals and a set of configuration registers. For example, DAI group A is used to route clock signals. The memory-mapped registers, SRU_CLKx, contain bit fields corresponding to the clock inputs of various peripherals. The values written to these bit fields specify a signal source that is an output from another peripheral. All of the possible encodings represent sources that are clock signals (or at least could be clock signals in some systems). Figure 10-10 diagrams the input signals that are controlled by the group A register, SRU_CLKx. All bit fields in the SRU configuration registers correspond to inputs. The value written to the bit field specifies the signal source. This value is also an output from some other component within the SRU.

The SRU is similar to a set of patch bays. Each bay routes a distinct set of outputs to compatible inputs. These connections are implemented as a set of memory-mapped registers with a bit field for each input. The outputs are implemented as a set of bit encodings. Conceptually, a patch cord is used to connect an output to an input. In the SRU, a bit pattern that is associated with a signal output (shown in Figure 10-10) is written to a bit field corresponding to a signal input.

The same encoding can be written to any number of bit fields in the same group. It is not possible to run out of patch points for an output signal.

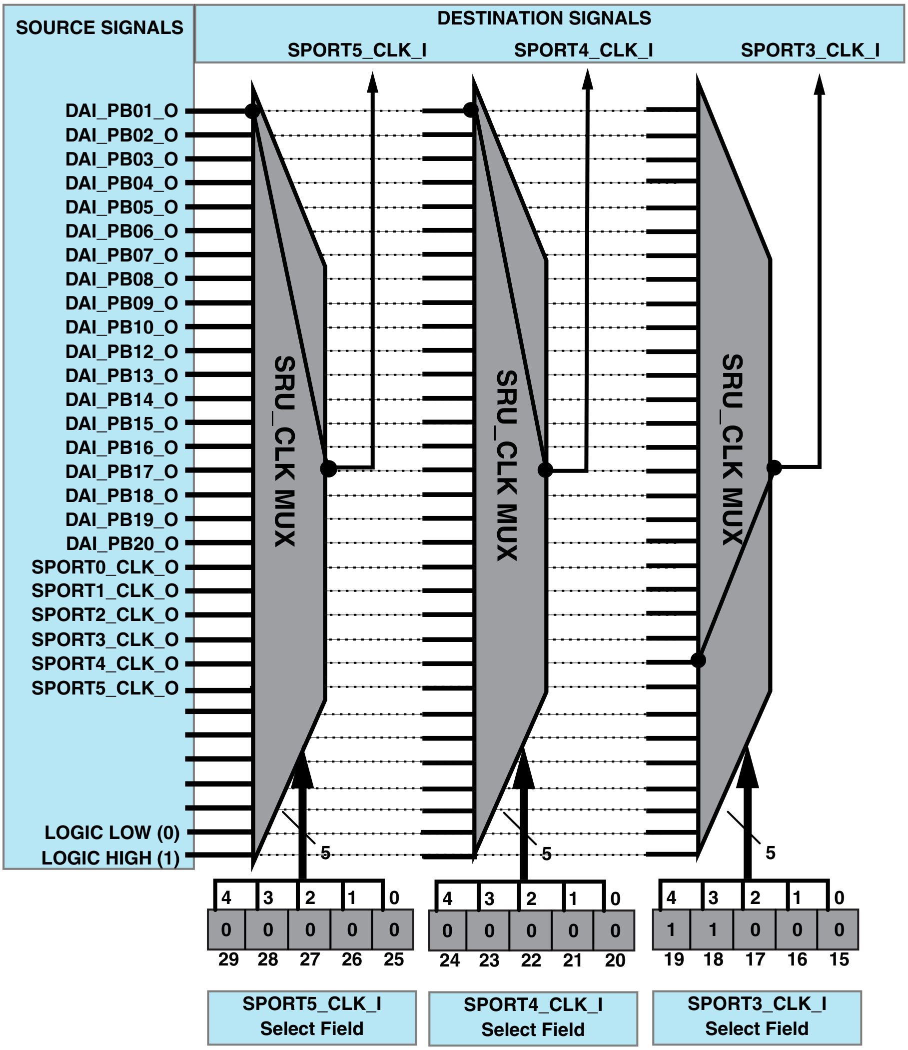

Figure 10-10. Example DAI SRU Group A Multiplexing (SRU_CLKx)

Just as group A routes clock signals, each of the other groups route a collection of compatible signals. Group B routes serial data streams while group C routes frame sync signals. Note that all of the groups have an encoding that allows a signal to flow from a pin output to the input being specified by the bit field.

Group D routes signals to pins so that they may be driven off-chip (required to route a signal to the pin input). Group F routes signals to the pin enables, and the value of these signals determines if a DAI pin is used as an output or an input. One pin's input can be patched to another pin's output, allowing board-level routing under software control.

Rules for SRU Connections

There are two rules which apply to all routing:

1. One source (output node) can drive different destinations (input nodes).
2. One destination (input node) can only be assigned to one source (output node).

As an example from Figure 10-10:

- `DAI_PB01_O` is routed to `SPORT5_CLK_I`
- `DAI_PB01_O` is routed to `SPORT4_CLK_I`
- `SPORT4_CLK_O` is routed to `SPORT3_CLK_I`

Inputs may only be connected to outputs.

Miscellaneous Buffers and Functions

The SRU group E provides miscellaneous buffers used for group interconnect which is explained below.

DAI group E or DPI group C connections are slightly different from the others in that the inputs and outputs being routed vary considerably in function. This group routes control signals and provides a means of connecting signals between groups.

For the DAI (Figure 10-11, Table 10-2), the `MISCAx_I` signals appear as inputs in group E (also connected to the DAI interrupt logic), but do not directly feed any peripheral. Rather, the `MISCAx_O` signals reappear as outputs in group F.

Table 10-2. DAI MISCAx SRU Signal Connections

MISCA Source	DAI Connection	MISCA Destination
	Group E	MISCA5–0_I
MISCA5–0_O	Group F	

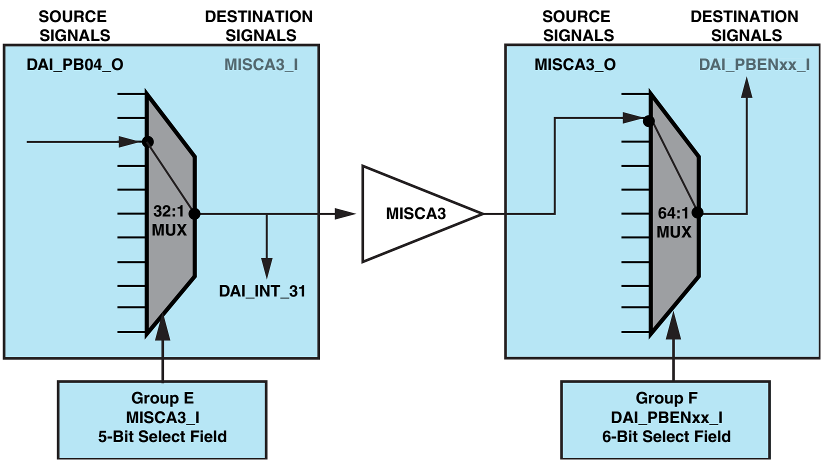

Figure 10-11. Miscellaneous DAI Connections

For the DPI (Figure 10-12, Table 10-3), the `MISCBx_I` signals appear as inputs in group A (also connected to DPI interrupt logic), but do not directly feed any peripheral. Rather, the `MISCBx_O` signals reappear as outputs in group C.

Additional connections among groups provide a great amount of utility. Since the output groups F (DAI) and C (DPI) dictate pin direction, these few signal paths enable a number of possible uses and connections for the DAI/DPI pins. Other examples include:

- A pin input can be patched to another pin's enable, allowing an off-chip signal to gate an output from the processor.
- Any of the DAI pins can be used as interrupt sources or general-purpose I/O (GPIO) signals.

Table 10-3. DPI MISCBx SRU2 Signal Connections

MISCB Source	DAI Connection	MISCB Destination
	Group A	MISCB8–0_I
MISCB8–0_O	Group C	

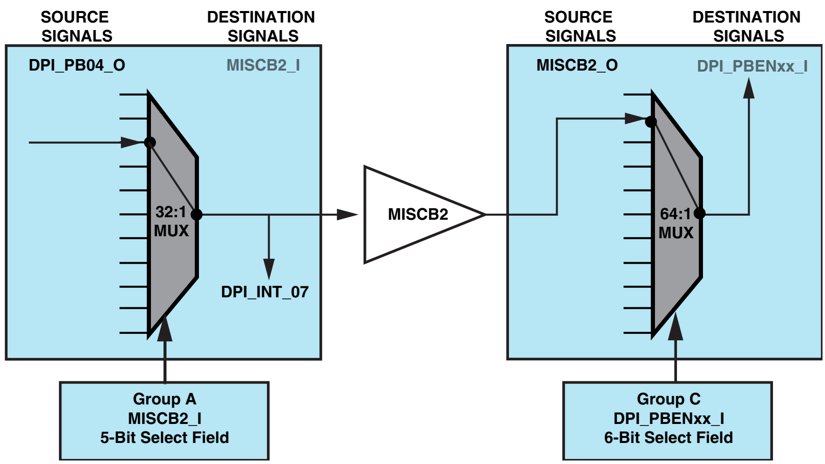

Figure 10-12. Miscellaneous DPI Connections

In summary the SRU enables many possible functional changes, both within the processor as well as externally. Used creatively, it allows system designers to radically change functionality at runtime, and to potentially reuse circuit boards across many products.

DAI/DPI Routing Capabilities

This section describes the routing options to aid in designing a system using the DAI/DPI.

In Table 10-4 and Table 10-5 the left column represents the source signals, the right column the destination signals, and the central column the group to which the signal belongs. A valid connection is built by connecting a source to destination signal within a group. For example in the DAI group A the source signal `SPORT2_CLK_O` needs to be connected to the sample rate converter0 destination signal `SRC0_CLK_I`.

In practice a macro is provided and forms the style `SRU(source_O, destination_I)` so the example appears as: `SRU(SPORT2_CLK_O, SRC0_CLK_I)`. The routing registers are described in "DAI Signal Routing Unit Registers" on page A-124 and "DPI Signal Routing Unit Registers" on page A-209.

DAI Routing Capabilities

Table 10-4 provides an overview of the different routing capabilities for the DAI unit. For information on an individual peripherals routing, see the "SRU Programming" section of that peripheral's chapter.

Table 10-4. DAI Routing Capabilities

Source Signals – Output (xxxx_O)		DAI Group	Destination Signals – Input (xxxx_I)
SPORT5–0 PCG A, B S/PDIF Rx (clock, TDM clock)	DAI Pin Buffer20–1 Logic level high Logic level low	A–Clocks	SPORT7–0 SRC3–0 IDP7–0 PCG A–D (Ext. clock, Ext. FS) S/PDIF-Tx (clock, HF Clock)
SPORT7–0 A, B SRC3–0 (data, TDM data) S/PDIF Tx/Rx		B–Data	SPORT7–0 A, B SRC3–0 (data, TDM data) IDP7–0 S/PDIF Tx/Rx
SPORT5–0 PCG A, B S/PDIF Rx		C–Frame Sync	SPORT7–0 SRC3–0 IDP7–0
SPORT7–0 (clock, FS, TDV, data) S/PDIF Rx (clock, TDM clock, FS, data) S/PDIF Tx (data, block start) PDAP (strobe) PCG C, D (clock, FS) (also in DPI)		D–Pin Buffer Inputs	DAI Pin Buffer 20–1 Options: DAI Pin Buffer 20–19 Polarity Change
SPORT5–0 (FS) PCG A (clock) PCG B (clock, FS) S/PDIF Tx (block start)		E–Miscella-neous Signals	**DAI Interrupt 31–22** **MISCA5–0** Options: MISCA5–4 Polarity Change
SPORT7–0 (clock, FS, data, TDV) MISCA5–0	Logic level high Logic level low	F–Pin Buffer Enable	DAI Pin Buffer Enable 20-1
ADSP-2147x Only			
SR_SCLK, SR_LAT (dedi-cated pins)	DAI Pin Buffer 8–1 Logic level high Logic level low	H–SR Clocks	SR_SCLK SR_LAT
SPORT7–0A, B (clock, FS) PCGA–B (clock, FS) SR_SDI (dedicated pin)		I–SR Data	SR_SDI

DPI Routing Capabilities

Table 10-5 provides an overview about the different routing capabilities for the DPI unit. For information on an individual peripherals routing, see the "SRU Programming" section of that peripheral's chapter.

Table 10-5. DPI Routing Capabilities

<table>
<tr><th colspan="2">Source Signals – Output (xxxx_O)</th><th>DPI Group</th><th>Destination Signals – Input (xxxx_I)</th></tr>
<tr><td>TIMER1–0
UART0 Tx Data</td><td rowspan="2">DPI Pin Buffer 14–1
Logic level high
Logic level low</td><td>A–Miscellaneous Signals</td><td>TIMER1–0
UART0 Rx data
SPIB (data, clock, control)
FLAG15–4
TWI (clock, FS)
MISCB8–0
DPI Interrupt 13–5</td></tr>
<tr><td>TIMER1–0
UART0 Tx data
SPI (data, clock, control)
FLAG/PWM15–4
PCG (C, D) (clock, FS)
(Also DAI)</td><td>B–Pin Buffer Input</td><td>DPI Pin Buffer14–1</td></tr>
<tr><td>TIMER1–0
UART0 Tx data
SPIB (data, clock, control)
FLAG15–4
TWI (clock, FS)
MISCB8–0</td><td>Logic level high
Logic level low</td><td>C–Pin Buffer Enable</td><td>DPI Pin Buffer Enable14–1</td></tr>
</table>

DAI Default Routing

When the processor comes out of reset, the SPORT junctions are bidirectional to the DAI pin buffers (Figure 10-13, Figure 10-14). This allows systems to use the SPORTs as either master or slave (without changing the routing scheme). Therefore, programs only need to use the SPORT control register settings to configure master or slave operation. Note that all DAI inputs which are not routed by default are tied to signal low.

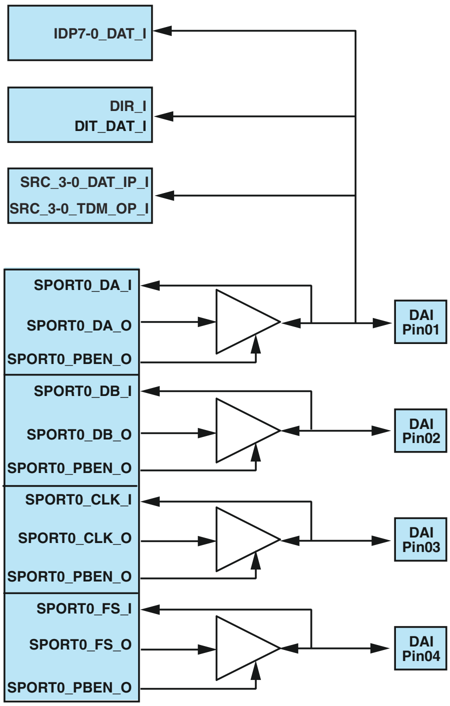

Figure 10-13. DAI Default Routing

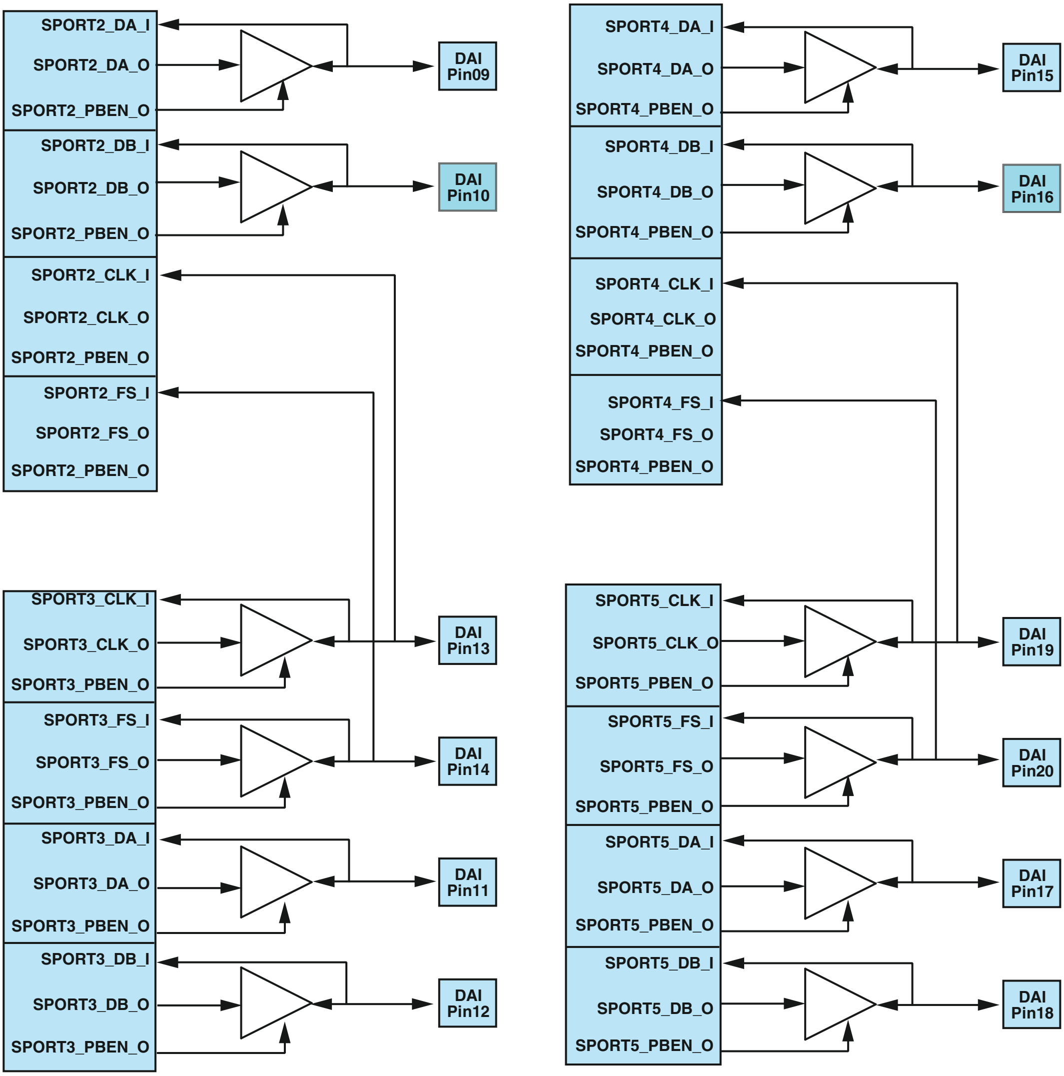

Figure 10-14. DAI Default Routing (Con't)

All DAI input buffers which are not routed by default are driven low and all DAI pin buffer enable signals are driven low.

DPI Default Routing

When the processor comes out of reset, some default routing is established (Figure 10-15). This scheme allows systems to use the SPI as either master or slave (without changing the routing scheme). Programs only need to use the SPI control register settings to configure master or slave operation.

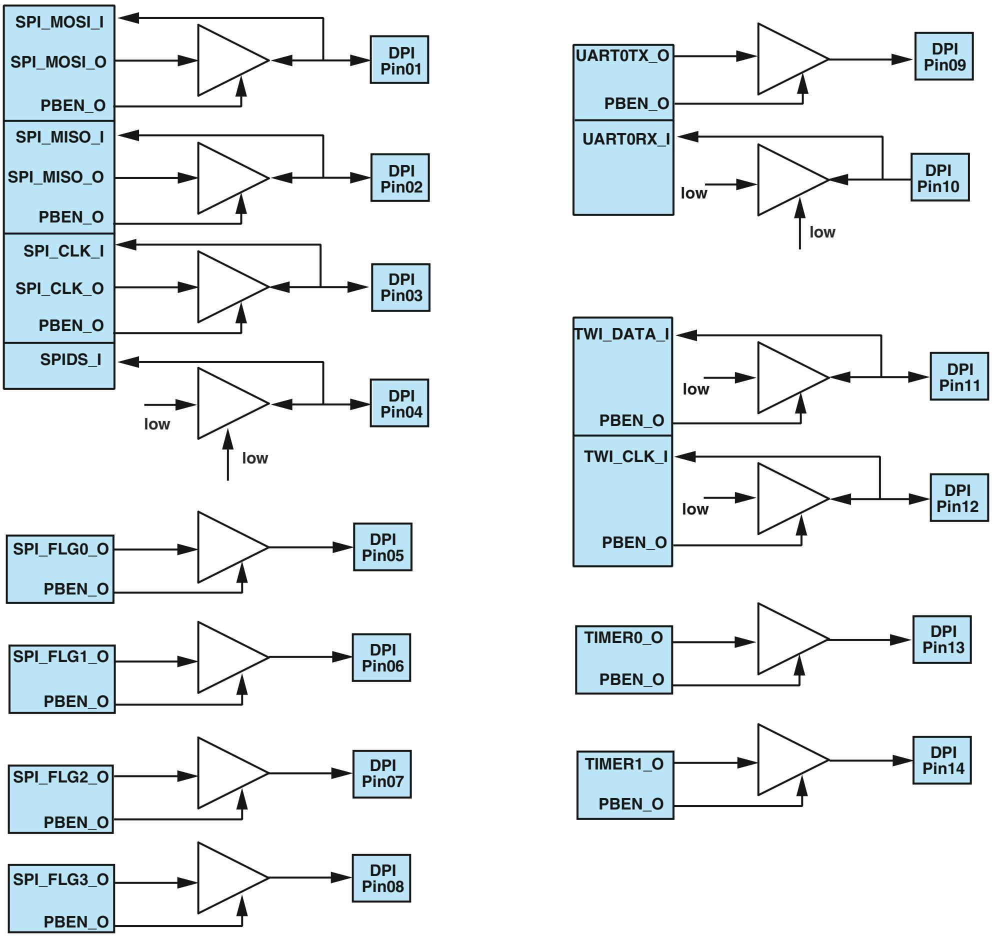

Figure 10-15. DPI Default Routing

All DPI input buffers which are not routed by default are driven low and all DPI pin buffer enable signals are driven low. For SPI boot, the DPI pin buffer enable signals 1 and 2 change depending on master or slave boot configuration.

Unused DAI/DPI Connections

As shown in previous sections, the SRUs have a default general-purpose routing scheme which may be modified to suit any number of different system designs. Regardless of the system design, it is good practice to tie all unused inputs to a high or low level to reduce dynamic power consumption. An example is shown in Listing 10-1.

Listing 10-1. Tieing Unused Pins Low

```
/* SPORT5 operates as transmitter only */
SRU(SPORT5_DA_O, DAI_PB03_I);   /* DAI pin 03 Data A output */
SRU(SPORT5_DB_O, DAI_PB04_I);   /* DAI pin 04 Data B output */
SRU(LOW, SPORT5_DA_I);          /* Data A input tied low */
SRU(LOW, SPORT5_DB_I);          /* Data B input tied low */

/* DAI Pin buffer 12 operates as input only */
SRU(LOW, DAI_PB12_I);                /* Input tied low */
SRU(SPORT5_PBEN_O, DAI_PBEN12_I);    /* Output Enable pin tied
                                        high */
```

Operating Modes

Some buffers allow polarity changes which are described below.

DAI Pin Buffer Polarity

As shown in Figure 10-16, the DAI pin buffer 20–19 can change the polarity of the input signal if the corresponding control bits `INV_PB[20-19]` in the `SRU_PIN4` register are set. These bits can be set during runtime and the buffer should not loopback to itself.

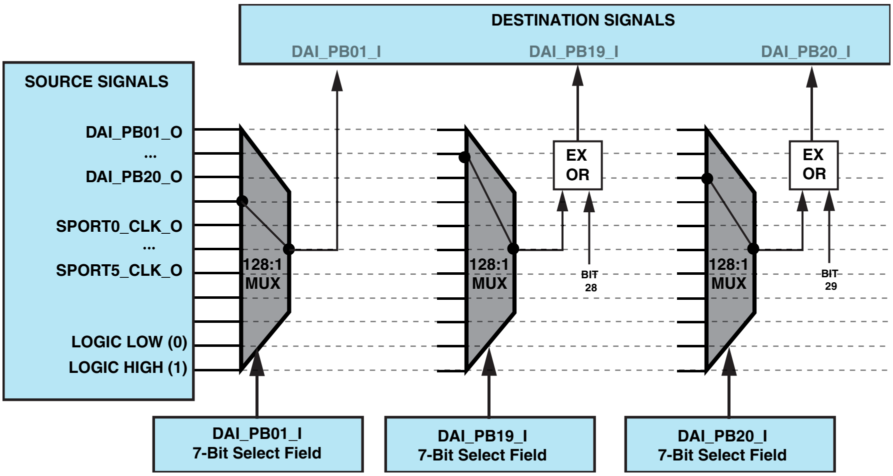

Figure 10-16. Pin Buffer Polarity

DAI Miscellaneous Buffer Polarity

As shown in Figure 10-16 the A5–4 miscellaneous buffers can change the polarity of the input signal if the corresponding control bits `INV_MISCA[5-4]` in the `SRU_EXT_MISCA` register are set. Both buffers are not connected to the DAI interrupt latch register. Note these bits can be set during runtime.

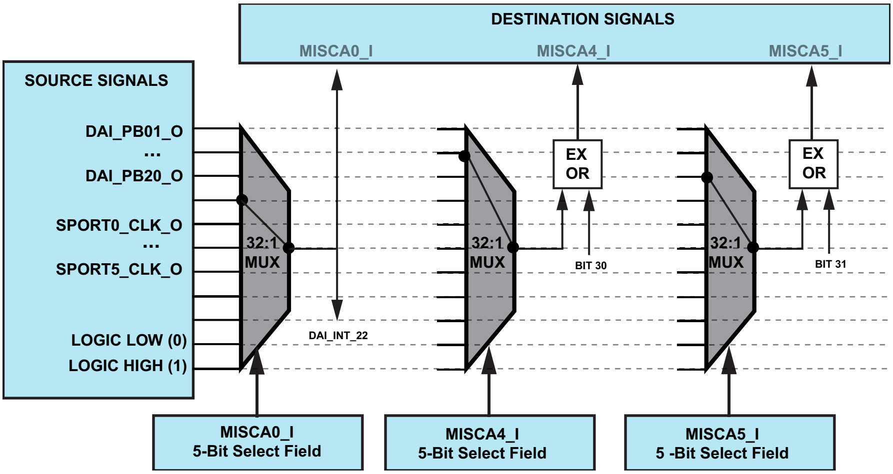

Figure 10-17. Miscellaneous Buffer Polarity

Interrupts

The following sections provide information on interrupt capabilities that are DAI/DPI specific. For information on DAI/DPI system interrupts, see Chapter 2, Interrupt Control.

DAI/DPI Miscellaneous Interrupts

Table 10-6 provides an overview of DAI/DPI miscellaneous interrupts.

Table 10-6. DPI Miscellaneous Interrupt Overview

Default Programmable Interrupt	Sources	Masking	Service
DAIHI = P0I DAILI = P12I DPII = P14I	Rising edge detect Falling edge detect Rising and falling edge detect	DAI_IMASK_RE DAI_IMASK_FE DPI_IMASK_RE DPI_IMASK_FE	ROC from DAI_IRPTL_x + RTI instruction ROC from DPI_IRPTL + RTI instruction

Sources

The DAI module generates 10 local miscellaneous interrupts and the DPI nine local miscellaneous interrupts. All the miscellaneous signals are connected into the DAI_IRPTL_x or DPI_IRPTL latch registers.

The MISC port can generate interrupts under these conditions. For some applications for instance, SIC needs information about interrupt sources that correspond to waveforms (not event signals). As a result, the falling edge of the waveform may be used as an interrupt source as well.

Edge Detection

Programs may select any of these three conditions:

- Latch on the rising edge
- Latch on the falling edge
- Latch on *both* the rising and falling edge

Masking

The DAIHI and DAILI signals are routed by default to programmable interrupt. To service the DAIHI, unmask (set = 1) the P0I bit in the IMASK register. To service the secondary DAILI, unmask (set = 1) the P12IMSK bit in the LIRPTL register. For DAI system interrupt controller the DAI_IMASK_RE or DAI_IMASK_FE register must be unmasked.

The DPI signal is routed by default to programmable interrupt. To service the DPI, unmask (set = 1) the P14I bit in the IMASK register. For DPI system interrupt controller the DPI_IMASK_RE or DPI_IMASK_FE register must be unmasked. For example:

```
bit set IMASK P0I;       /* unmasks P0I interrupt */
bit set LIRPTL P12IMSK;    /* unmasks P12I interrupt */
bit set IMASK P14I;        /* unmasks P14I interrupt */
```

Service

To clear the interrupt request, the interrupt service routine needs to read from the `DAI_IRPTL_x` or `DPI_IRPTL` register.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

Signal Routing Unit Effect Latency

After the DAI/DPI registers are configured the effect latency is 2 `PCLK` cycles minimum and 3 `PCLK` cycles maximum.

Programming Model

As discussed in the previous sections, the signal routing unit is controlled by writing values that correspond to signal sources into bit fields that further correspond to signal inputs. The SRU is arranged into functional groups such that the registers that are made up of these bit fields accept a common set of source signal values.

In order to ease the coding process, the header file `SRU.H` is included with the CrossCore or VisualDSP++ tools. This file implements a macro that abstracts away most of the work of signal assignments and functions. The macro has identical syntax in C/C++ and assembly, and makes a single connection from an output to an input as shown below.

```
SRU(Output Signal, Input Signal);
```

The names passed to the macro are the names given "DAI Signal Routing Unit Registers" on page A-124.

The code in Listing 10-2 shows how the macro is used.

Listing 10-2. DAI Macro Code

```
#include <sru.h>;
/* The following lines illustrate how the macro is used: */
/* Route SPORT 1 clock output to pin buffer 5 input */
       SRU(SPORT1_CLK_O,DAI_PB05_I);

   /* Route pin buffer 14 out to IDP3 frame sync input */
       SRU(DAI_PB14_O,IDP3_FS_I);

   /* Connect pin buffer enable 19 to logic low */
       SRU(LOW,PBEN19_I);
```

Additional example code is available on the Analog Devices Web site.

There is a macro that has been created to connect peripherals used in a DAI configuration. This code can be used in both assembly and C code. See the INCLUDE file SRU.H.

There is also a software plug-in called the Expert DAI that greatly simplifies the task of connecting the signals described in this chapter. This plug-in is described in Engineer-to-Engineer Note EE-243, "Using the Expert DAI for ADSP-2126x and ADSP-2136x SHARC Processors". This EE note is also found on the Analog Devices Web site.

Making SRU Connections

In this section, three types of SRU routing are demonstrated.

1. Listing 10-3 and Figure 10-18 show the SRU connection between the DAI and pin buffers.
2. Listing 10-4 on page 10-37 and Figure 10-19 on page 10-37 show the SRU connection between the DAI pin buffers and SPORTs.
3. Listing 10-5 on page 10-38 and Figure 10-20 on page 10-38 show SRU connection from the SPORT/PCG to the MISC/DAI pin buffers.

These examples use a macro which is provided by the CrossCore or VisualDSP++ tools. Also see "Programming Model" on page 10-33.

Listing 10-3. SRU Connection Between DAI Pin Buffers

```
SRU(HIGH, PBEN03_I);       // DAI pin 3 output
nop;
SRU(LOW, PBEN14_I);        // DAI pin 14 input
nop;
SRU(LOW, DAI_PB14_I);       // DAI pin 14 input level low
nop;
SRU(DAI_PB14_O, DAI_PB03_I); // connect DAI pin 14 to DAI pin 3
nop;
SRU(DAI_PB14_O, DAI_INT_22_I); // connect DAI pin 14 to DAI
                                   interrupt 22
```

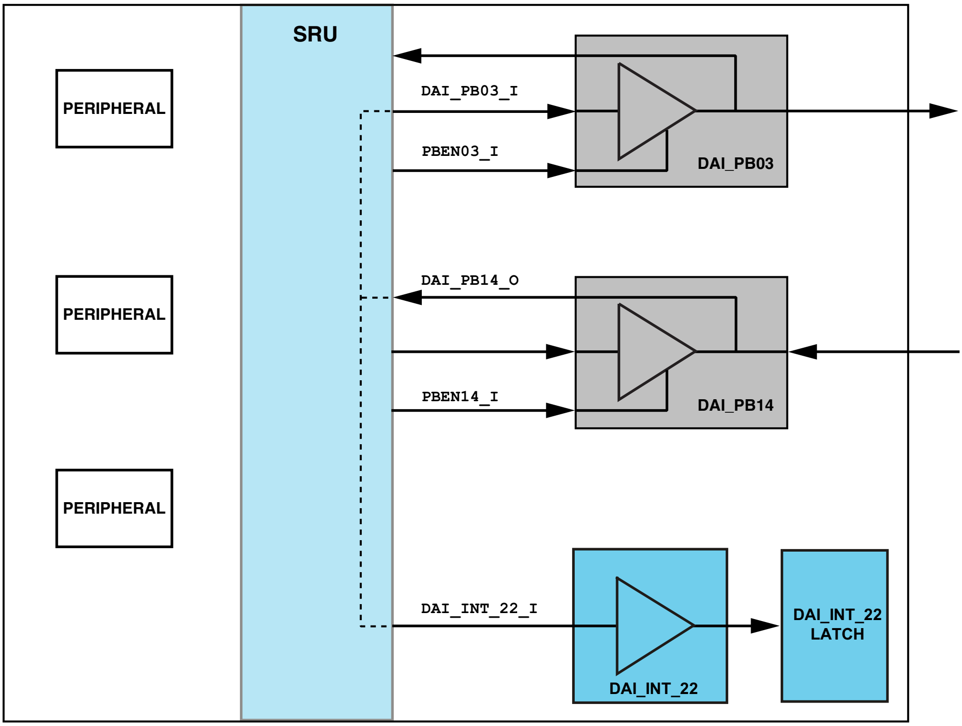

Figure 10-18. SRU Connection Between DAI Pin Buffers

Listing 10-4. SRU Connection Between DAI Pin Buffers and SPORTs

```
SRU(SPORT0_CLK_PBEN_O, PBEN03_I);   // DAI pin 3 as output
nop;
SRU(SPORT0_CLK_O, DAI_PB03_I);      // connect to DAI pin 3
nop;
SRU(SPORT0_CLK_O, SPORT1_CLK_I);    // connect to SPORT1
nop;
SRU(SPORT0_CLK_O, SPORT2_CLK_I);    // connect to SPORT2
```

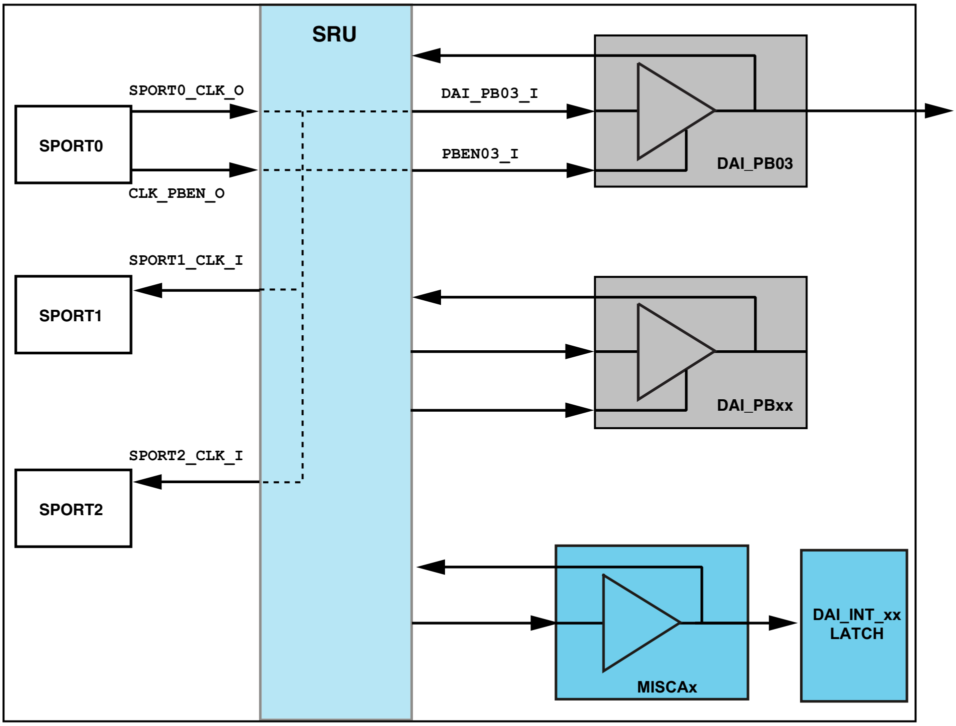

Figure 10-19. SRU Connection Between DAI Pin Buffers and SPORTs

Listing 10-5. SRU Connection SPORT/PCG to MISC/DAI Pin Buffers

```
SRU(HIGH, PBEN03_I);          // DAI pin 3 output
nop;
SRU(DAI_PB14_O, DAI_PB03_I);  // connect pin 3 and 14
nop;
SRU(PCG_CLKB_O, DAI_PB14_I);  // connect PCG and pin 14
nop;
SRU(SPORT2_FS_O, MISCA4_I);   // connect SPORT to MISCA
nop;
SRU(MISCA4_O, PBEN14_I);      // connect MISCA to PBEN14
nop;
SRU(HIGH, INV_MISCA4_I);      // invert MISCA4 input
```

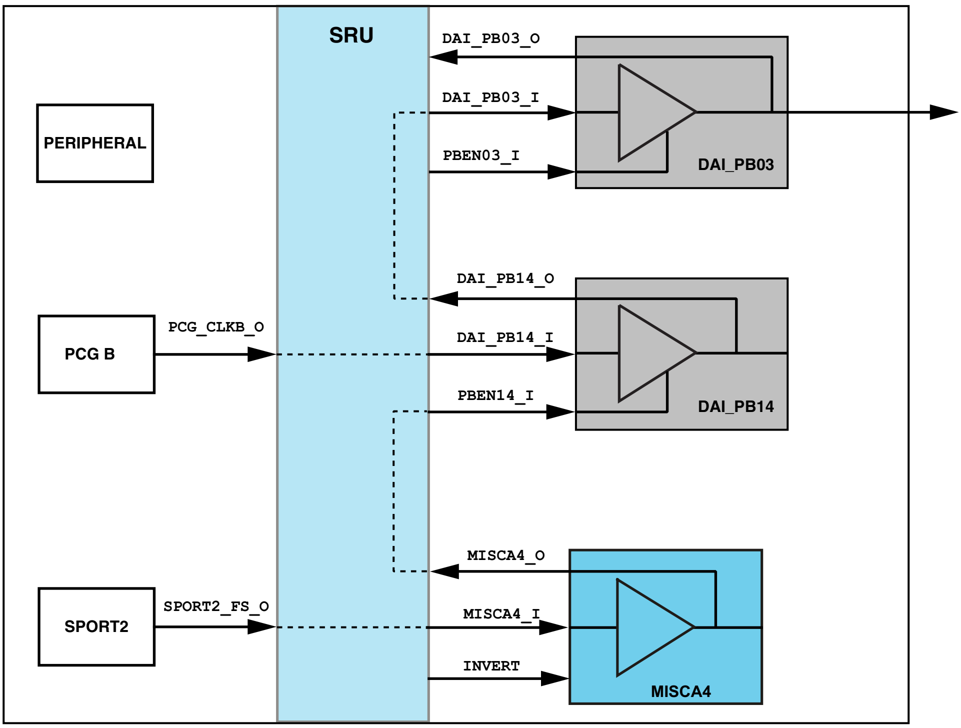

Figure 10-20. SRU Connection SPORT/PCG to MISC/DAI Pin Buffers

DAI Example System

A complete system using the DAI peripherals (SPORTs, PCG, S/PDIF) is shown in Figure 10-21.

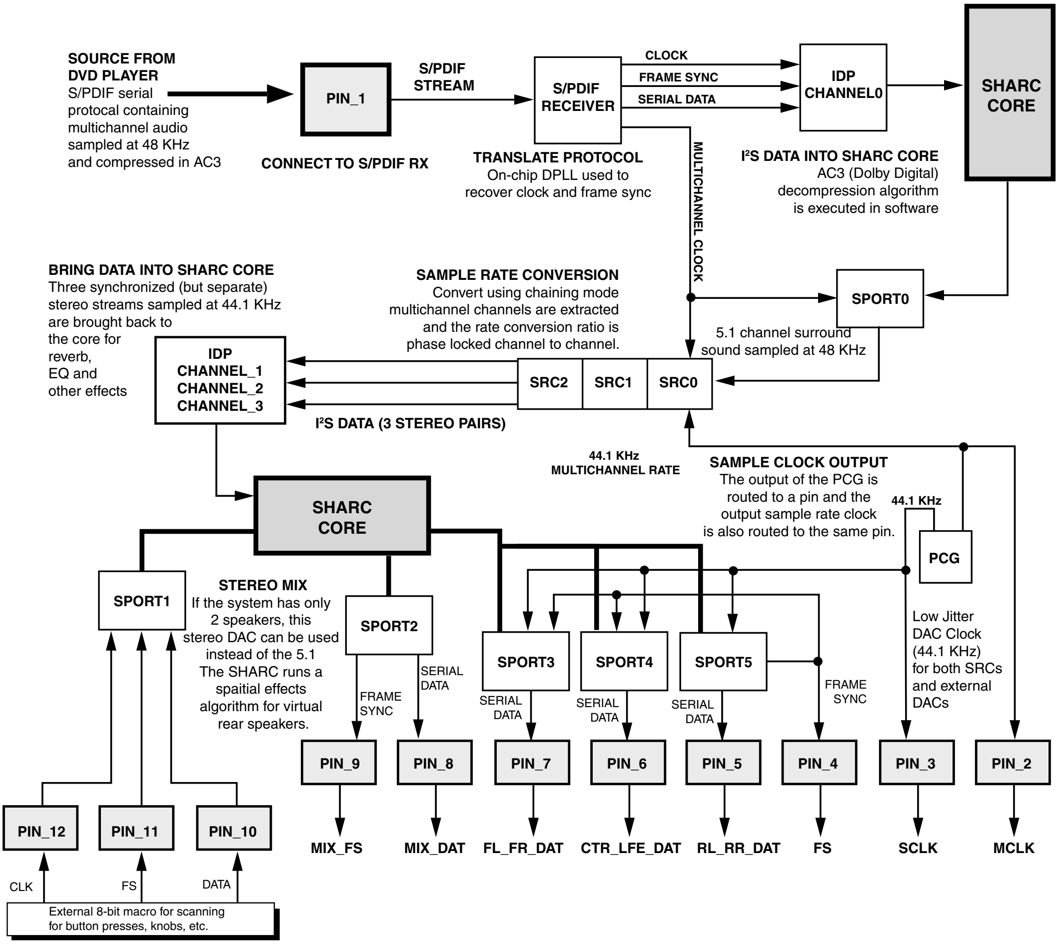

Figure 10-21. DAI Example

Debug Features

The following sections describe features that can be used to help in debugging the DAI.

Shadow Interrupt Registers

For more information, see “Debug Features” on page 2-15.

Loopback Routing

The serial peripherals (SPORT and SPI) support an internal loopback mode. If the loopback bit for each peripheral is enabled, it connects the transmitter with the receiver block internally (does not signal off-chip). The SRU can be used for this purpose. Table 10-7 describes the different possible routings based on the peripheral.

The peripheral’s loopback mode for debug is independent from both of the signal routing units.

Table 10-7. Loopback Routing

Peripheral	Loopback Mode	SRU/SRU2 Internal Routing for Loopback	SRU/SRU2 External Routing for Loopback
DAI			
IDP	N/A	N/A	N/A
SPORT	Yes	SPORTx_xx_O → SPORTx_xx_I	SPORTx_xx_O → DAI_PBxx_I DAI_PBxx_O → SPORTx_xx_I
S/PDIF Tx/Rx	No	DIT_O → DIR_I	DIT_O → DAI_PBxx_I DAI_PBxx_O → DIR_I
SRC	No	SRCx_DAT_OP_O → SRCx_DAT_IP_I	SRCx_DAT_OP_O → DAI_PBxx_I DAI_PBxx_O → SRCx_DAT_IP_I
DPI			
Timer	No	TIMERx_O → TIMERx_I	TIMERx_O → DPI_PBxx_I DPI_PBxx_O → TIMERx_I
SPI	Yes	No	SPIx_xx_O → DPI_PBxx_I DPI_PBxx_O → SPIx_xx_I
UART0	No	UART0_TX_O → UART0_RX_I	UART0_TX_O → DPI_PBxx_I DPI_PBxx_O → UART0_RX_I
TWI	No	No	TWI_xx_O → DPI_PBxx_I DPI_PBxx_O → TWI_xx_I

11 SERIAL PORTS (SPORTS)

The processors have eight independent, synchronous serial ports (SPORTs) that provide an I/O interface to a wide variety of peripheral devices and are optimized for multichannel audio applications. They are called SPORT0 to SPORT7. Each SPORT has its own set of control registers and data buffers. With a range of clock and frame synchronization options, the SPORTs allow a variety of serial communication protocols and provide a glueless hardware interface to many industry-standard data converters and CODECs. The interface specifications are shown in Table 11-1.

Table 11-1. Serial Port Specifications

Feature	SPORT7–0[AB]
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	Yes
SRU DAI Default Routing	Yes
SRU2 DPI Required	No
SRU2 DPI Default Routing	N/A
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	Yes
Transmission Simplex	Yes
Transmission Half Duplex	Yes

Table 11-1. Serial Port Specifications (Cont'd)

Feature	SPORT7–0[AB]
Transmission Full Duplex	No
Access Type	
Data Buffer	Yes
Core Data Access	Yes
DMA Data Access	Yes
DMA Channels	2 per SPORT
DMA Chaining	Yes
Miscellaneous	
Clock Power Management	Yes
Boot Capable	No
Local Memory	No
Clock Operation	$f_{PCLK}/4$ ($f_{PCLK}/8$ if SPORT is slave transmitter or master receiver)

Features

Serial ports offer the following features and capabilities:

- A variety of protocols are supported (see "Operating Modes" on page 11-29):
    1. Standard serial
    2. Left-justified
    3. I^2S
    4. Packed
    5. Multichannel

- Two bidirectional channels (A and B) per serial port, configurable as either transmitters or receivers. Each serial port can also be configured as two receivers or two transmitters, permitting two unidirectional streams into or out of the same serial port. This bidirectional functionality provides greater flexibility for serial communications. Further, two SPORTs can be combined to enable full-duplex, dual-stream communications.

  Serial ports can operate at a maximum of one-fourth the peripheral clock rate of the processor. If channels A and B are active, each SPORT has a maximum throughput of 2 x PCLK/4 rate.

- Chained DMA operations for multiple data blocks, see "Chained DMA" on page 3-32.

- SPORT DMA channels are assigned highest priority for fixed DMA arbitration mode.

- DMA Chain insertion mode allows the SPORTs to change DMA flow during chaining. See "Enter DMA Chain Insertion Mode" on page 11-59.

- Data words between 3 and 32 bits in length, either most significant bit (MSB) first or least significant bit (LSB) first.

- For multichannel/packed protocol, a SPORT pair can be combined together for full-duplex, dual-stream communications.

- 128-channel multichannel is supported in multichannel mode operation, useful for audio CODEC connections or H.100/H.110 and other telephony interfaces described in "Multichannel Protocol" on page 11-21.

- In multichannel mode active channel selection logic allows programs to enable/disable individual channels.

- μ-law and A-law companding hardware on transmitted (compression) and received (expansion) words when the SPORT operates in multichannel mode.
- Supports error event detection for unexpected Frame Syncs and not meeting real time requirements (data buffer under/overflow).

Receive comparison and 2-dimensional DMA are not supported.

Serial Port Versus Input Data Port Features

If the input stream requires I^2S, left-justified or right-justified protocols the IDP may be an appropriate interface to use. It supports up to 8 DMA channels and frees up SPORT resources. Table 11-2 shows an overview. For more information, see Chapter 12, Input Data Port (SIP, PDAP).

Table 11-2. Support Versus IDP Features

Feature	SPORT	IDP
Data inputs	Serial 32-bit	Serial 32-bit Parallel 20-bit
Data direction	Input/Output	Input
Data companding	Yes	No
Protocol	Standard serial, I^2S, left-justified, packed, multichannel	I2S, left justified, right-justified
Bus	Master/slave	Slave
Serial clock	Any	64 × FS
DMA modes	Standard/chained	Standard/Ping-pong
DMA channels	16 (8 × 2)	8

Pin Descriptions

Table 11-3 describes pin function.

Table 11-3. SPORT Pin Descriptions

Internal Node	Direction	Description
SPORT7–0_DA_I	I	**Data receive channel A.** Bidirectional data pin. If TRAN = 0, input to receive serial data.
SPORT7–0_DA_O	O	**Data transmit channel A.** Bidirectional data pin. If TRAN = 1, output to transmit serial data. The transmit data pin is always driven (and continues to drive last level of serial word) if the serial port is enabled and TRAN=1 unless it is in multichannel/packed mode and an inactive channel slot occurs.
SPORT7-0_DB_I	I	**Data receive channel B.** Bidirectional data pin. If TRAN = 0, input to receive serial data.
SPORT7–0_DB_O	I/O	**Data transmit channel B.** Bidirectional data pin. If TRAN = 1, output to transmit serial data. The transmit data pin is always driven (and continues to drive last level of serial word) if the serial port is enabled and TRAN=1 unless it is in multichannel/packed mode and an inactive channel slot occurs.
SPORT7–0_CLK_I/O	I/O	**Transmit/Receive Serial Clock.** This signal can be either internally or externally generated.
SPORT7–0_FS_I/O	I/O	**Transmit/Receive Frame Sync.** The frame sync pulse initiates shifting of serial data. This signal is either generated internally or externally. It can be active high or low or an early or a late frame sync, in reference to the shifting of serial data.
SPORT7–0_TDV_O	O	**Multichannel Transmit Data Valid.** This output only active in SPORT transmit multichannel/packed protocol mode. The signal is asserted during active transmit channel slots based on the active channel selection registers.

Table 11-3. SPORT Pin Descriptions (Cont'd)

Internal Node	Direction	Description
SPORT7–0_DA_PBEN_O	O	**Pin Buffer Enable Pins.** For correct SPORT operation the PBEN output signals should be routed to its pin buffer input signal. Note that the SCLK and FS outputs are only driven if programmed as master.
SPORT7–0_DB_PBEN_O	O	
SPORT7–0_CLK_PBEN_O	O	
SPORT7–0_FS_PBEN_O	O	
SPORT7–0_TDV_PBEN_O	O	

SRU Programming

Any of the serial port's signals can be mapped to digital applications interface (`DAI_Px`) pins through the signal routing unit (SRU) as shown in Table 11-4. For more information, see Chapter 10, Digital Application/Digital Peripheral Interfaces.

> ⓘ SPORTs 6 and 7 receive their clocks from other routed sources but cannot route their own clocks to other SPORTs or other peripherals internally through the SRU. If SPORTs 6 and 7 are needed externally, route them through the DAI pins.

Table 11-4. SPORT DAI/SRU Signal Connections

Serial Port Source	DAI Connection	Serial Port Destination
Inputs		
SPORT5–0_CLK_O	Group A	SPORT7–0_CLK_I
SPORT7–0_DA_O SPORT7–0_DB_O	Group B	SPORT7–0_DA_I
SPORT5–0_FS_O	Group C	SPORT7–0_FS_I

Table 11-4. SPORT DAI/SRU Signal Connections (Cont'd)

Serial Port Source	DAI Connection	Serial Port Destination
SPORT7–0_CLK_O SPORT7–0_DA_O SPORT7–0_DB_O SPORT7–0_FS_O SPORT7–0_TDV_O	Group D	
SPORT5–0_FS_O	Group E	
SPORT7–0_CLK_PBEN_O SPORT7–0_DA_PBEN_O SPORT7–0_DB_PBEN_O SPORT7–0_FS_PBEN_O SPORT7–0_TDV_PBEN_O	Group F	

SRU SPORT Receive Master

If the SPORT is operating as receive master, it must feed its master output clock back to its input clock. This is required to trigger the SPORT's state machine. Using SPORT 4 as an example receive master, programs should route SPORT4_CLK_O to SPORT4_CLK_I. This is not required if the SPORT is operating as a transmitter in master mode.

SRU SPORT Signal Integrity

There is some sensitivity to noise on the clock (SPORTx_CLK) and frame sync (SPORTx_FS) signals when the SPORT is configured as a master receiver. By correctly programming the signal routing unit (SRU) clock and frame sync registers, the reflection sensitivity in these signals can be avoided.

Figure 10-10 on page 10-19 shows the default routing of the serial port where the SRU maps to:

- The signal from the DAI pin (DAI_PBxx_O) back to the SPORT clock input (SPORTx_CLK_I)
- The SPORT clock output (SPORTx_CLK_O) to the pin buffer input (DAI_PBxx_I)

By redirecting the signals as shown in Figure 11-1 where the clock and frame sync outputs are routed directly back to their respective inputs, the signal sensitivity issue can be avoided.

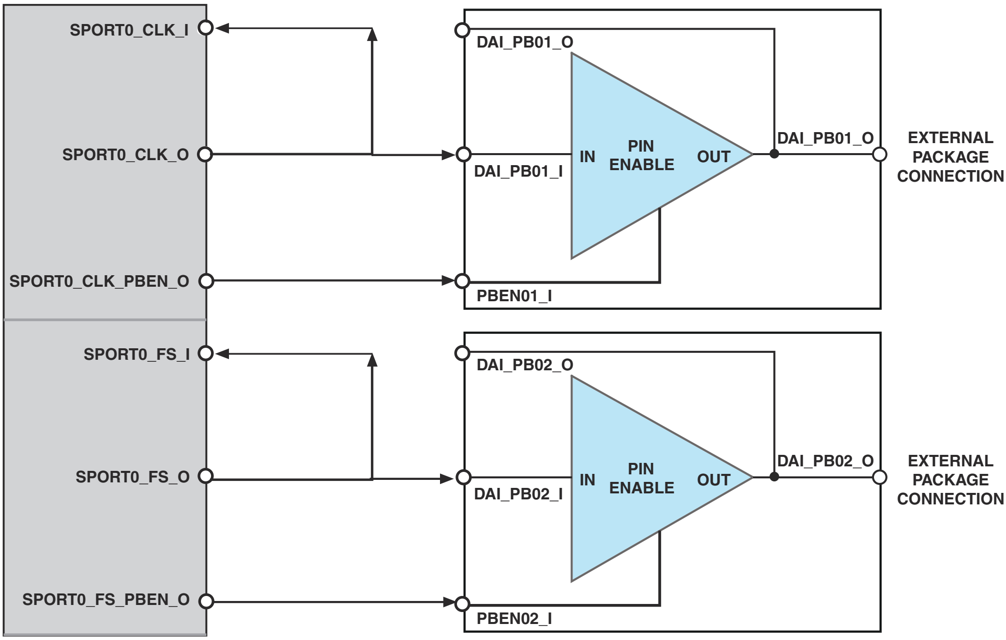

Figure 11-1. SRU Configuration when SPORT is Master Receiver.

Register Overview

This section provides brief descriptions of the major registers. For complete information, see"Serial Port Registers" on page A-155.

Master Clock Divider (DIVx). Contain divisor values that determine frequencies for internally-generated clocks and frame syncs. If your system requires more precision and less noise and jitter, refer to Chapter 15, Precision Clock Generator.

Serial Port Control/Status (SPCTLx). Control serial port modes and are part of the SPCTLx (transmit and receive) control registers. Other bits in these registers set up DMA related serial port features. For information about configuring a specific operation mode, refer to Table 11-8 on page 11-29 and "Operating Modes" on page 11-29.

Multichannel Control/Status (SPMCTLx). There is one global control and status register for each SPORT (SPORT7–0) for multichannel operation. These registers define the number of channels, provide the status of the current channel, enable multichannel operation, and set the multichannel frame delay.

Serial Port Control N (SPCTLNx). Control enhanced serial port modes and also allow compatibility mode switches between legacy SPORT modules. See "SPORT Control 2 Registers (SPCTLNx)" on page A-165.

Serial Port Error (SPERRxx). Two error registers (SPERRCTLx/SPERRSTAT) are used to observe and control error handling during transfers. Detected errors can be frame sync violation or buffer over/underflow conditions. For more information, see "Sources" on page 11-52.

Clocking

The fundamental timing clock of the SPORT modules is peripheral clock/4 (PCLK/4). Each serial port has a clock signal (SPORTx_CLK) for transmitting and receiving data on the two associated data signals. The clock and frame sync signals are configured by the ICLK/IFS and CLKDIV/FSDIV bits of the SPCTLx/DIVx control registers. One clock signal clocks both A and B data signals (either configured as inputs or outputs) to receive or transmit data at the same rate. The clock to this module may be shut off for power savings. For more information, see “Disabling Peripheral Clocks” on page 23-11.

Master Clock

The CLKDIV bit field specifies how many times the processor’s internal clock (PCLK) is divided to generate the transmit and receive clocks. The frame sync (SPORTx_FS) is considered a receive frame sync if the data signals are configured as receivers. Likewise, the frame sync SPORTx_FS is considered a transmit frame sync if the data signals are configured as transmitters. The divisor is a 15-bit value, (bit 0 in divisor register is reserved) allowing a wide range of serial clock rates. Use the following equation to calculate the serial clock frequency:

TX master: $SCLK = PCLK \div (4 \times (CLKDIV + 1))$ for $CLKDIV[1–32767]$

RX master: $SCLK = PCLK \div (4 \times (CLKDIV + 1))$ for $CLKDIV[2–32767]$

The maximum serial clock frequency is equal to one-fourth (0.25) the processor’s internal peripheral clock (PCLK) frequency, which occurs when CLKDIV is set to a minimum of 1. Use the following equation to determine the value of CLKDIV, given the PCLK frequency and desired serial clock frequency:

$$CLKDIV = (PCLK \div 4 \times SCLK) - 1$$

Master Frame Sync

The bit field FSDIV specifies how many transmit or receive clock cycles are counted before a frame sync pulse is generated. In this way, a frame sync can initiate periodic transfers. The counting of serial clock cycles applies to internally- or externally-generated serial clocks. The formula for the number of cycles between frame sync pulses is:

Number of serial clocks between frame syncs = $FSDIV + 1$

Use the following equation to determine the value of FSDIV, given the serial clock frequency and desired frame sync frequency:

$FSDIV = (SCLK \div FSCLK) - 1$

The frame sync is continuously active when FSDIV = 0. The value of FSDIV should not be less than the serial word length (the value of the SLEN field in the serial port control register), as this may cause an external device to abort the current operation or cause other unpredictable results.

Programming SLEN > FSDIV–1 field causes a FS error exception if error logic is enabled. For more information, see “Interrupts” on page 11-52.

Programs should not use the master clock/frame sync on the SPORTs to drive ADCs/DACs in high fidelity audio systems since jitter may be introduced from the on-chip PLL. To alleviate this problem use the precision clock generator (PCGx) routed by low jitter external master clocks (CLKIN or DAI pin inputs). For more information, see Chapter 15, Precision Clock Generator.

General-Purpose Pulse Generator

If the serial port is not being used, the FSDIV divisor can be used as a counter for dividing an external clock or for generating a periodic pulse or periodic interrupt. The serial port must be enabled for this mode of operation to work properly.

If the SPORT serial clock (SCLK) is required as general-purpose clock in a system, only the ICLK/MSTR bit and the serial clock divider register DIVx must be programmed.

If the frame sync of SPORT (FS) is required as general-purpose clock in a system, the ICLK/MSTR bit and the serial clock divider register DIVx must be programmed. Additionally the SPEN_x/SPTRAN/DIFS bits must be set.

Slave Mode

Exercise caution when operating with externally-generated transmit clocks near the frequency of PCLK/4 of the processor's internal clock. There is a delay between when the clock arrives at the SPORTx_CLK node and when data is output. This delay may limit the receiver's speed of operation. Refer to the appropriate product data sheet for exact timing specifications.

Externally-generated late transmit frame syncs also experience a delay from when they arrive to when data is output. This can also limit the maximum serial clock speed. Refer to the appropriate product data sheet for exact timing specifications.

Mixed Mode

This mode allows combinations of serial clock as master and frame sync as slave or vice versa. This mode is only supported by the standard serial and multichannel or packed protocol modes.

Maximum Clock Rate Restrictions

Caution should be exercised when operating with externally generated transmit clocks near the maximum operating frequency (PCLK/4). There is a delay between when the clock arrives at the DAI pin and when data is output which may limit the receiver's operating speed. For reliable operation, it is recommended that full-speed serial clocks only be used when receiving with an externally generated clock and externally generated frame sync (ICLK = 0, IFS = 0).

Externally-generated late transmit frame syncs (LAFS = 1) also experience a delay from when they arrive to when data is output which can also limit the maximum serial clock speed. Refer to the product-specific data sheet for exact timing specifications.

Clock Power Savings

For information on managing power when the SPORT is paused or not used, see Chapter 23, Power Management.

Functional Description

The following sections provides general information on the function of the SPORTs.

- "Architecture" below
- "Data Types Format" on page 11-33
- "Frame Sync Modes" on page 11-34

Architecture

A serial port receives serial data on one of its bidirectional serial data signals configured as inputs, or transmits serial data on the bidirectional serial data signals configured as outputs. It can receive or transmit on both channels simultaneously and unidirectionally, where the pair of data signals can both be configured as either transmitters or receivers.

The SPORTx_DA and SPORTx_DB channel data signals on each SPORT cannot transmit and receive data simultaneously for full-duplex operation. Two SPORTs must be combined to achieve full-duplex operation. The

SPTRAN bit in the SPCTLx register controls the direction for both the A and B channel signals.

> The data direction of channel A and channel B on a particular SPORT must be the same.

Serial communications are synchronized to a clock signal. Every data bit must be accompanied by a clock pulse. Each serial port can generate or receive its own clock signal (SPORTx_CLK). Internally-generated serial clock frequencies are configured in the DIVx registers. The A and B channel data signals shift data based on the rate of SPORTx_CLK.

Note that if the SPORT is enabled in master mode, the serial clock starts running unless the SPORT is disabled.

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 signals depends upon the type of serial device connected to the processor. Each serial port can generate or receive its own frame sync signal (SPORTx_FS) for transmitting or receiving data. Internally-generated frame sync frequencies are configured in the DIVx registers. Both the A and B channel data signals shift data based on their corresponding SPORTx_FS signal.

Figure 11-2 shows a block diagram of a serial port. Setting the SPTRAN bit enables the data buffer path, which, once activated, responds by shifting data in response to a frame sync at the rate of SPORTx_CLK. An application program must use the correct serial port data buffers, according to the value of the SPTRAN bit. The SPTRAN bit enables either the transmit data buffers for the transmission of A and B channel data, or it enables the receive data buffers for the reception of A and B channel data. Inactive data buffers are not used.

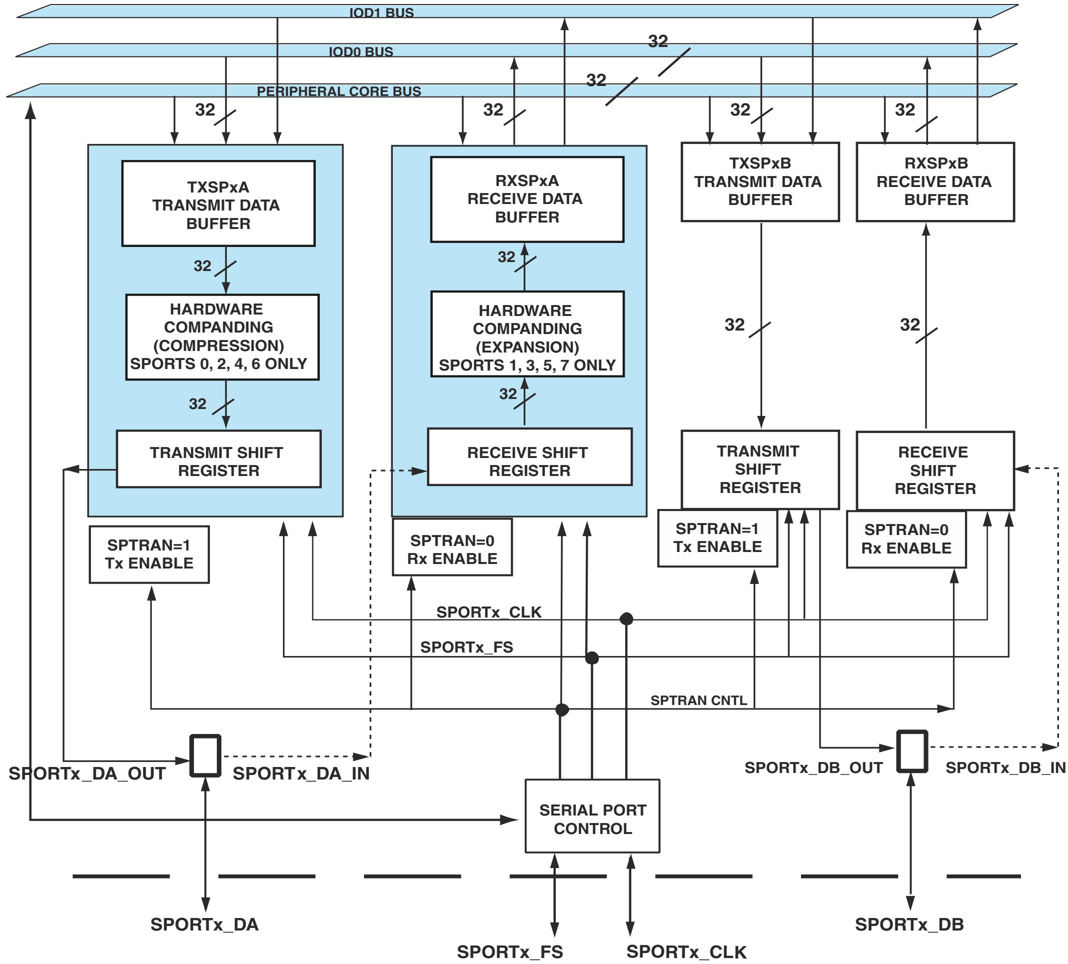

Figure 11-2. Serial Port Block Diagram

Companding

Companding (compressing/expanding) is the process of logarithmically encoding and decoding data to minimize the number of bits that must be sent. The processor's serial ports support the two most widely used companding algorithms, A-law and μ-law, performed according to the CCITT G.711 specification. Note that companding is not supported for I^2S and left-justified protocols.

The type of companding can be selected independently for each SPORT. Companding is selected by the `DTYPE` field of the `SPCTLx` control register.

ⓘ Companding is supported on the A channel only. SPORT0, 2, 4 and 6 primary channels are capable of compression, while SPORTs 1, 3, 5 and 7 primary channels are capable of expansion.

The processor's SPORTs 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.

Frame Sync and Data Sampling

The information contained in this section is generic to the SPORTs in any operating mode. Additional information about frame syncs and data sampling that applies to a specific operating mode can be found in "Operating Modes" on page 11-29.

As shown in Figure 11-3 the SPORT uses two control signals to sample data.

1. Serial clock (`SCLK`) applies the bit clock for each serial data.
2. Frame sync (`FS`) divides the incoming data stream into frames.

Frames define the required data length (after the serial to parallel conversion) necessary to store the data in memory for further processing as shown in Figure 11-3. For example, the transmitter drives the master clock and frame sync while the receiver slave is sampling the data.

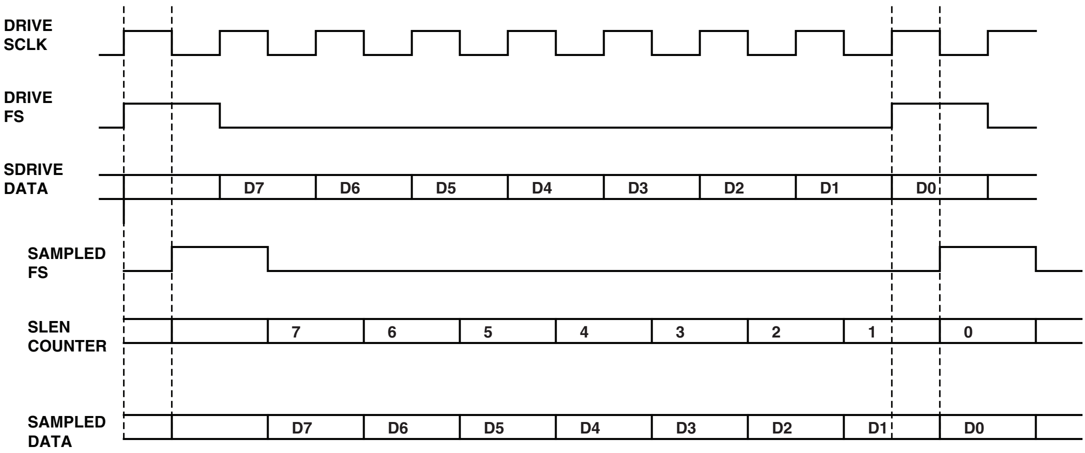

Figure 11-3. Frame Sync and Data Driven on Rising Edge

After the slave has sampled the FS the SLEN word counter is reloaded to its maximum setting. Each SCLK period decrements the SLEN counter until the full frame is received. If the transmitter drives the frame sync and data on the rising edge, the falling edge is used to sample the frame sync and data, and vice versa.

Continuous Framed Data Transfers

If data transmission is continuous in framing mode (for example, the last bit of each word is immediately followed by the first bit of the next word), the following settings should be used.

- For non-multichannel protocol mode, load the FSDIV register with SLEN–1. For example, for 8-bit data words set SLEN = 0x7 and FSDIV = 0x7.
- For multichannel/packed mode the FS period = serial word length × number of channels. In multichannel mode if NCH = 0x7 channels, set SLEN = 0x7 and FSDIV = 0x3F.

SPORT Protocols

The following sections provide brief descriptions of the serial port supported protocols. For more information, see Appendix C, Audio Frame Formats.

Standard Serial Protocol

The standard serial mode lets the serial ports interface to 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.

Protocol Configuration Options

- Data (direction, linear or companding)
- Little or big endian
- Serial word length (3-32 bits)
- Data buffer (16 or 32-bit packing)
- Serial clock (internal or external edge select)

- Frame sync (polarity, required vs continuous, internal or external, early or late, data or channel de-pendent)
- DMA (standard or chained)
- Debug (loopback test)

Left-Justified Protocol

The left-justified protocol transmits and or receives two samples of data in each frame sync cycle—one sample on the high segment (left channel) of the frame sync, the other on the low segment (right channel) of the frame sync.

Protocol Configuration Options

- Data (direction)
- Serial word length (8-32 bits)
- Data buffer (16 or 32-bit packing)
- Serial clock and FS (Internal or External)
- Frame sync (polarity, data dependent, channel first)
- DMA (standard or chained)
- Debug (loopback test)

I^{2}S Protocol

The I^{2}S protocol transmits and or receives two samples of data in each frame sync cycle–one sample on the high segment (right channel) of the frame sync, the other on the low segment (left channel) of the frame sync. Note that in I^{2}S mode, the data is delayed by one `SCLK` cycle.

Protocol Configuration Options

- Data (direction)
- Serial word length (8-32 bits)
- Data buffer (16 or 32-bit packing)
- Serial clock and FS (Internal or External)
- Frame sync (polarity, data dependent, channel first)
- DMA (standard or chained)
- Debug (loopback test)

I^{2}S Compatibility

In previous generations of SHARC processors, the serial ports did not generate a FS edge (word select signal) after the transmission of the last word in the DMA channel. This differed from standard I^{2}S receivers which look for the edge to latch and read data. Therefore, I^{2}S slave receivers connected to the SHARC SPORTs were unable to latch the last word of a transmit DMA. The ADSP-214xx SHARC processors are able to generate the last FS edge (WS) if configured as an I^{2}S master (valid only for DMA), if the extra frame edge (`I2SEFE` bit) in the `SPCTLNx` register is set. If this bit is cleared, legacy behavior is provided.

Setting the `I2SEFE` bit generates the compatible (extra) frame edge only for continuous data streams (`FSDIV = SLEN` programmed in `DIVx` register). In the cases where `FSDIV` > `SLEN` value programmed, the extra last edge is not generated.

Channel Order First

The LFS bit (renamed to L_FIRST) is used for the I^2S/left-justified or the packed protocol to determine which frame transmits or receives first. The left and right channels are time-duplex data channels.

Table 11-5. Channel Order First

OPMODE	L_FIRST = 0	L_FIRST = 1
Left-Justified	Left channel first	Right channel first
I^2S/Packed	Right channel first	Left channel first

In I^2S/left justified or packed protocols the SPORT starts to drive the FS signal high (=1) for the first valid frame.

Multichannel Protocol

The processor's serial ports offer a multichannel mode of operation (Figure 11-4), 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 and 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.

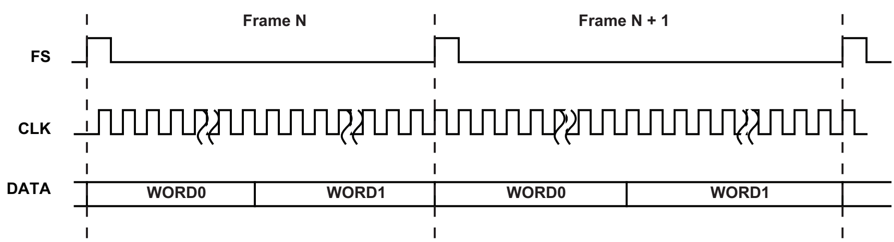

Figure 11-4. TDM and Multichannel Protocol

Protocol Configuration Options

- Data (direction, linear or companding)
- Little or big endian
- Serial word length (3-32 bits)
- Data buffer (16 or 32-bit packing)
- Serial clock (internal or external, edge select)
- Frame sync (polarity, internal or external, delay)
- DMA (standard or chained)
- TDM channel (number, activation)

Multiple Channels

The serial port can automatically select some words for particular channels while ignoring others. Up to 128 channels are available for transmitting or receiving or both. Each SPORT can receive or transmit data selectively from any of the 128 channels.

To operate in full-duplex operation, the SPORT can be optionally routed in pairs together, each SPORT configured as transmitter/ receiver. All receiving and transmitting devices in a multichannel system must have the same timing reference.

Data companding and DMA transfers can also be used in multichannel mode on channel A. Channel B can also be used in multichannel mode, but companding is not available on this channel.

Although the eight SPORTs are programmable for data direction in the standard mode of operation, their programmability is restricted for multichannel operations. The following points summarize the following limitations.

- The primary A channels of SPORT1, 3, 5, and 7 are capable of expansion only, and the primary A channels of SPORT0, 2, 4, and 6 are capable of compression only.
- Receive comparison is not supported.

Multichannel Frame Sync Delay

The 4-bit MFD field (bits 4–1) in the multichannel control registers (SPMCTLx) specifies a delay between the frame sync pulse and the first data bit in multichannel mode. The value of MFD is the number of serial clock cycles of the delay. Multichannel frame delay allows the processor to work with different types of telephony interface devices.

A value of zero for MFD causes the frame sync to be concurrent with the first data bit. The maximum value allowed for MFD is 15. A new frame sync may occur before data from the last frame has been received, because blocks of data occur back to back.

Figure 11-5 shows an example of timing for a multichannel transfer with SPORT pairing using SPORT0 and 1. The transfer has the following characteristics.

- SPORT1–0 have the same SCLK and frame sync as input.
- Multichannel is configured as 8 channels.
- SPORT0A drives data to DAC1 during slot 1–0 which asserts TDV for two slots.
- SPORT1A drives data to DAC2 during slot 3–2 which asserts TDV for two slots.
- SPORT1B receives data from ADC during slot 3–0.

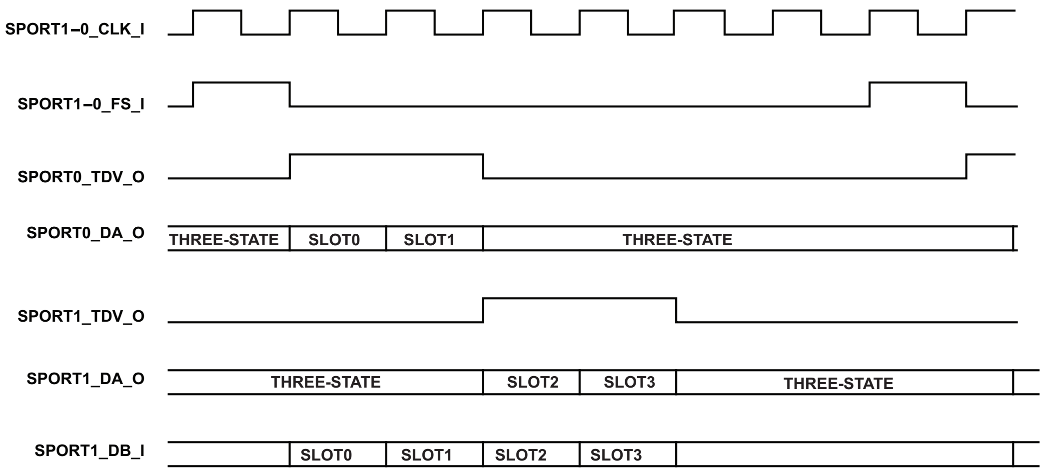

Figure 11-5. Multichannel Operation

Number of Channels (NCH)

Select the number of channels used in multichannel operation by using the 7-bit `NCH` field in the multichannel control register. Set `NCH` to the actual number of channels minus one (`NCH` = Number of channels – 1).

The 7-bit `CHNL` field in the multichannel control registers indicates the channel that is currently selected during multichannel operation. This field is a read-only status indicator. The `CHNL(6:0)` bits increment modulo `NCH(6:0)` as each channel is serviced.

Active Channel Selection Registers

Specific channels can be individually enabled or disabled to select the words that are received and transmitted during multichannel communications. Data words from the enabled channels are received or transmitted, while disabled channel words are ignored. Up to 128 channels are available for transmitting and receiving.

The multichannel selection registers enable and disable individual channels. The registers for each serial port are shown in "Serial Port Registers" on page A-155.

Each of the four multichannel enable and compand select registers are 32 bits in length. These registers provide channel selection for 128 (32 bits × 4 channels = 128) 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). For example, setting bit 0 in the SP0CS0 register (SPORT0) or SP7CS0 register (SPORT7) selects channel 0, setting bit 12 selects channel 12, and so on. Setting bit 0 in SP0CS1 register (SPORT0) or SP7CS1 register (SPORT7) selects channel 32, setting bit 12 selects channel 44, and so on.

Active Channel Companding Selection Registers

Companding may be selected on a per-channel basis as shown in Table 11-6. Setting a bit to 1 in any of the multichannel registers (SPx-CCSy specifies that the data be companded for that channel. A-law or μ-law companding can be selected using the DTYPE bit in the SPCTLx control registers. SPORTA1, 3, 5 and 7 expand selected incoming time slot data, while SPORTA0, 2, 4 and 6 can compress the data.

ⓘ Active channel selection registers must be enabled according to the application, otherwise a core or DMA hang can occur.

Table 11-6. Active Channel Selection Register Settings

Active Channel Selection Registers	Active Channel Companding Selection Registers	Activated Channel Slots
SP7–0CS0	SP7–0CCS0	0–31
SP7–0CS1	SP7–0CCS1	32–63
SP7–0CS2	SP7–0CCS2	64–95
SP7–0CS3	SP7–0CCS3	96–127

Companding Limitations (ADSP-2146x)

In multichannel mode, there is an option to enable companding for any active channel. If the first active channel is NOT channel 0 and companding is enabled for the first active channel (for example, channel 2), then from the second frame onward companding for channel 2 does not occur.

In Table 11-7 channel 0 and 1 are not active and channel 2 is active and companding is enabled. For the ADSP-2146x processors, in the first frame companding occurs for the first active channel (for example, channel 2) but the second frame onward companding for channel 2 does not occur. However, for other channels, companding occurs correctly. In Table 11-7, 1 = Active, 0 = Not Active, x = Don't care.

Table 11-7. Companding

Channel Number	0	1	2	3	4	5
Active Channel Number	0	0	1	x	x	x
Companding Enable	0	0	1	x	x	x

For the ADSP-2147x and ADSP-2148x processors, setting the COMPANDEN bit in the SPCTLNx register overcomes this limitation.

Transmit Data Valid Output Enable

Each SPORT has its own transmit data valid signal (SPORTx_TDV_O) which is active during the transmission of an enabled word. Because the serial port's receiver signals are three-stated when the time slot is not active, the SPORTx_TDV_O signal specifies if the SPORT data is being driven by the processor.

For polarity change of the SPORTx_TDV_O output signal use any of the DAI_PB20-19_I inputs of the routing unit. For more information, see "DAI Pin Buffer Polarity" on page 10-30.

Multichannel Protocol Backward Compatibility

In previous ADSP-2136x SHARC processors, multichannel protocol mode required a selected SPORT pair (SPORT01, 23 or 45), even if only half-duplex operation was required. This pair needs to route the SCLK and FS (regardless of master/slave) to the odd SPORT (receive) and the TDV output enable signal is multiplexed with the FS output of even SPORT (transmit). The pair itself interconnects the SCLK and FS signals internally.

With the ADSP-21367/8/9 processors and later, multichannel mode operates completely independently and no pair is required for half-duplex operation. Each SPORT uses its own SCLK, FS and TDV signal programmed using the SRU. The FS signal synchronizes the channels and restarts each multichannel sequence. The SPORTx_FS signal initiates the start of the first channel data word. The frame sync can be configured in master or slave mode based on the setting of the IFS bit and the FS polarity can be changed using the LFS bit.

Packed Protocol

A packed mode is available in the SPORT and can be 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^2S mode. Packed mode is implemented using standard multichannel mode (and is therefore programmed similarly to multichannel mode). Packed mode also supports the maximum of 128 channels as does multichannel mode as well as the maximum of (128 x 32) bits per left or right channel.

Protocol Configuration Options

- Data (direction, linear or companding)
- Little or big endian
- Serial word length (3-32 bits)

- Data buffer (16 or 32-bit packing)
- Serial clock (internal or external, edge select)
- Frame sync (internal or external, channel order, delay)
- DMA (standard or chained)
- TDM channel (number, activation)

Packed Words

As shown in Figure 11-6, 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^2S mode. So it is a hybrid between multichannel and I^2S mode.

Note that every polarity change of FS restarts a new TDM frame, therefore the frame sync frequency is one-half of that in the TDM protocol.

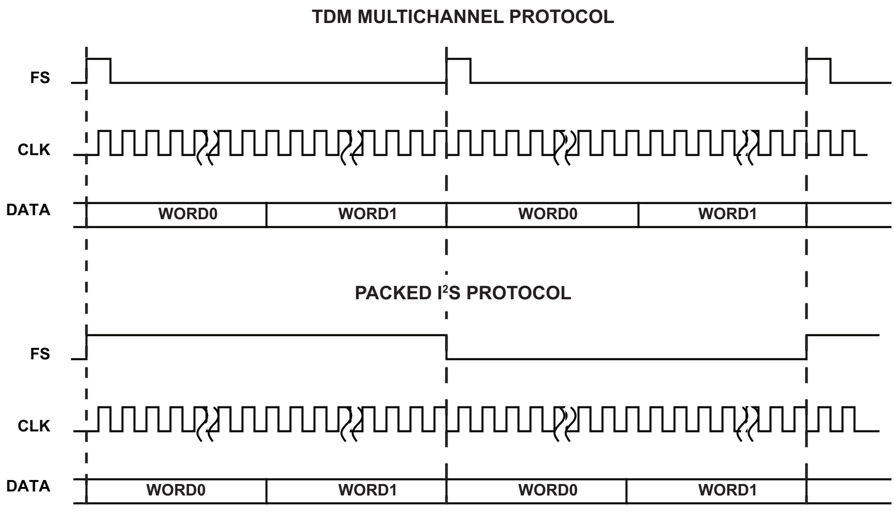

Figure 11-6. Packed I^2S Versus TDM Multichannel Protocols

Operating Modes

The serial port protocol modes are selected via bits in the SPCTLx and the SPMCTLx registers as shown in Table 11-8 and the following list.

1. Bits 0/24 (SPEN_A/SPEN_B) of SPCTLx register enables I^2S, left-justified, and standard serial protocols for channel A/B.
2. Bit 11 (OPMODE) of the SPCTLx register selects between I^2S, left-justified, and standard serial/multichannel protocols for channel A/B.
3. Bit 17 (LAFS) of the SPCTLx register selects between I^2S and left-justified protocol only if bit 11(OPMODE) set.
4. Bits 0/23 (MCEA/MCEB) of SPMCTLx register enable multichannel and packed protocols for channel A/B.
5. Bit 11 (OPMODE) of the SPCTLx register selects between the multichannel and packed protocol for channel A/B only if bits 0/24 (SPEN_A/SPEN_B) are cleared.

Table 11-8. SPORT Protocol Enable Bit Settings

OPERATING MODES (x = A or B or A and B SPORT Channels)	SPCTLx Bits			SPMCTLx Bits
	OPMODE (Bit 11)	OPMODE (Bit 17)	SPEN_x (Bit 0/24)	MCEx
Standard Serial Mode	0	Valid	1	0
Left-justified Mode	1	1	1	0
I^2S Mode	1	0	1	0
Packed Mode	1	0	0	1
Multichannel Mode	0	0	0	1

When changing protocol mode, clear the serial port control registers before the new protocol mode is written to the register.

The SPORTs operate in five protocols which are listed in the next tables. In each protocol a bit can have the same meaning or a different meaning or is reserved. All modes depending on protocol are described in this section.

The SPCTLx and SPMCTL control registers are unique in that the name and function of their bits change depending on the protocol selected. In the following sections, the bit names associated with the protocol are described. Table 11-8 provides values for each of the bits in the SPORT serial control registers that must be set in order to configure each specific protocol. The shaded regions indicated bits responsible for protocol mode setting.

The main control register for each serial port is the serial port control register, SPCTLx. These registers are described in“Serial Port Registers” on page A-155.

When changing operating modes, clear the serial port control register before the new mode is written to the register.

The SPCTLx registers control the operating modes of the serial ports. Table 11-9 lists all the bits in the SPCTLx register.

Table 11-9. SPCTLx Control Bit Comparison

[Bit] Name	Standard Serial Mode	Multichannel Mode	Packed I^2S Mode	I^2S and Left-justified Mode
Control				
[0] SPEN_A	Used	Reserved		Used
[1–2] DTYPE	Used			Reserved
[3] LSBF	Used			Reserved (=1)
[4–8] SLEN	Used			
[9] PACK	Used			
[10]	Used			Used (MSTR)

Table 11-9. SPCTLx Control Bit Comparison (Cont'd)

[Bit] Name	Standard Serial Mode	Multichannel Mode	Packed I^{2}S Mode	I^{2}S and Left-justified Mode
[11] OPMODE	Used			
[12] CKRE	Used			Reserved (=1)
[13] FSR	Used	Reserved (=1)		
[14]	Used			Reserved
[15] DIFS	Used	Reserved (=1)		Used
[16] LFS	Used		Used (L_FIRST)	
[17] LAFS	Used	Reserved		Used (OPMODE)
[18] SDEN_A	Used			
[19] SCHEN_A	Used			
[20] SDEN_B	Used			
[21] SCHEN_B	Used			
[22] FS_BOTH	Used	Reserved (=0)		Reserved (=1) if both channels
[23] BHD	Used			
[24] SPEN_B	Used	Reserved		Used
[25] SPTRAN	Used			
Status				
[26] DERR_B	Used			
[27–28] DXS_B	Used			
[29] DERR_A	Used			
[30–31] DXS_A	Used			

Table 11-10. SPMCTLx Control Bit Comparison

[Bit] Name	Standard Serial Mode	I^{2}S and Left-justified Mode	Multichannel Mode	Packed I^{2}S Mode
Control				
[0] MCEA	Reserved		Used	
[4–1] MFD	Reserved		Used	
[11–5] NCH	Reserved		Used	
[12] SPL	Used		Reserved	
[22–16] CHNL	Reserved		Used	
[23] MCEB	Reserved		Used	
Status				
[24] DMASxA	Standard DMA Channel A Status			
[25] DMASxB	Standard DMA Channel B Status			
[26–27]	Compatible to legacy programming model			
[28] DMACHSxA	Chain Loading DMA Channel A Status			
[29] DMACHSxB	Chain Loading DMA Channel B Status			
[30–31]	Compatible to legacy programming model			

Mode Selection

The following sections provide detailed information on operating modes available in some or all protocols.

Data Direction

The SPTRAN bit enables the channel A or B as a transmitter or receiver. Since one SPORT operates in half-duplex mode, both channels must be either transmit or receive. Otherwise one other SPORT is required for full-duplex mode (see “Packed Protocol” on page 11-27).

The SPORT output data signal is always driven if the serial port is enabled as transmitter (SPTRAN = 1 and SPEN_x= 1) unless it is in multichannel/packed mode and an inactive channel slot occurs.

Serial Word Length

The serial word length is not unique and is based on the operation mode. Moreover the companding feature limits the word length settings.

Words smaller than 32 bits are right-justified in the receive and transmit buffers, residing in the least significant bit (LSB) positions (Table 11-11).

Table 11-11. Serial Word Length Versus Modes

Mode	Word Length (SLEN) Bits
Standard Serial Mode	3–32
Multichannel	3–32
Packed	3–32
Left justified	8–32
I^2S	8–32

Data Types Format

Linear transfers occur in the A channel if the A channel is active and companding is disabled (bit 1 of DTYPE) for that A channel. Companded transfers occur if the A channel is active and companding is enabled for that A channel. The multichannel compand select registers (SPxCCSy) specify the transmit and receive channels that are companded when multichannel mode is enabled. Companding is not supported for the B channel.

For A and B channels transmit or receive sign extension is selected by bit 0 of DTYPE in the SPCTLx register and is common to all transmit or receive channels. If bit 0 of DTYPE is set, sign extension occurs on selected A channels that do not have companding selected. If this bit is not set, the word contains zeros in the MSB positions.

> ⓘ The compression for transmission requires a minimum word length of 8 (SLEN = 7) for proper function. If SLEN < 7 the expansion may not work correctly.

Sampling Edge

Data and frame syncs can be sampled on the rising or falling edges of the serial port clock signals. The CKRE bit of the SPCTLx control registers selects the sampling edge. For sampling receive data and frame syncs, setting CKRE to 1 in the SPCTLx register selects the rising edge of SPORTx_CLK. When CKRE is cleared (=0), the processor selects the falling edge of SPORTx_CLK for sampling receive data and frame syncs.

Note that transmit data and frame sync signals change their state on the clock edge that is not selected. For example, the transmit and receive functions of any two serial ports connected together should always select the same value for CKRE so internally-generated signals are driven on one edge and received signals are sampled on the opposite edge.

Frame Sync Modes

This section describes the different operating modes for the frame sync signal.

Framed Versus Unframed Frame Syncs

The use of frame sync signals is optional in serial port communications. The FSR (transmit frame sync required) bit determines whether frame sync signals are required. Active low or active high frame syncs are selected using the LFS bit. This bit is located in the SPCTLx control registers.

When FSR 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 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.

Figure 11-7 illustrates framed serial transfers.

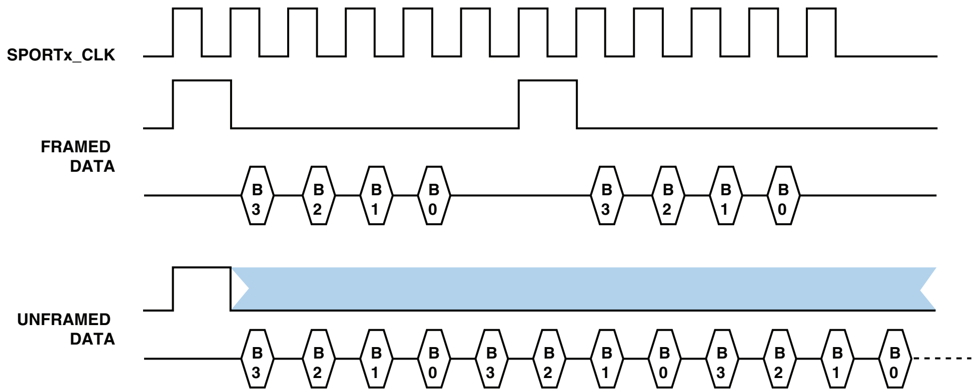

Figure 11-7. Framed Versus Unframed Data

Early Versus Late Frame Syncs

Frame sync signals can be early or late. Frame sync signals can occur during the first bit of each data word or during the serial clock cycle immediately preceding the first bit. The LAFS bit of the SPCTLx control register configures this option.

When LAFS is cleared (=0), early frame syncs are configured. This is the default 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 (for example, the last bit of each word is immediately followed by the first bit of the next word), the 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.

When `LAFS` is set (=1), late frame syncs are configured. 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.

Figure 11-8 illustrates the two modes of frame signal timing.

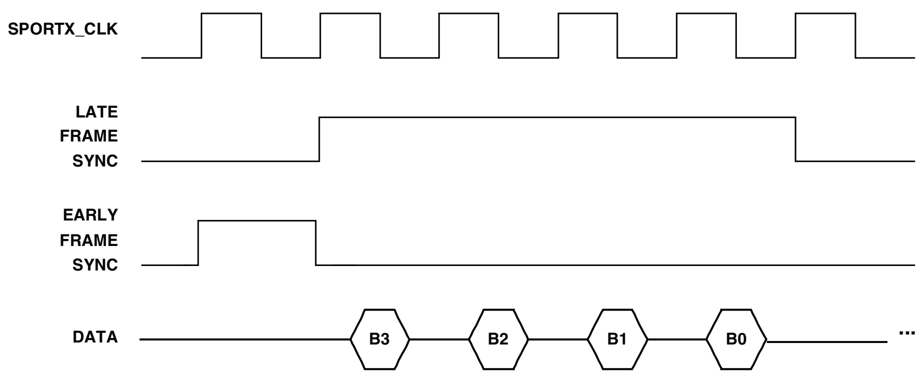

Figure 11-8. Normal Versus Alternate Framing

Internal Versus External Frame Syncs

Both transmit and receive frame syncs can be generated internally or input from an external source. The IFS bit of the SPCTLx control register determines the frame sync source.

When IFS is set (=1), the corresponding frame sync signal is generated internally by the processor, and the SPORTx_FS signal is an output. The frequency of the frame sync signal is determined by the value of the frame sync divisor (FSDIV) in the DIVx register.

When IFS is cleared (=0), the corresponding frame sync signal is accepted as an input on the SPORTx_FS signals, and the frame sync divisors in the DIVx registers are ignored.

All frame sync options are available whether the signal is generated internally or externally.

Note that for I^2S, left-justified, and packed protocols, the MSTR bit selects the clock and frame sync to be together configured as master or slave.

Polarity Frame Sync Level

Frame sync signals may be active high or active low (for example, inverted). The LFS/LMFS bit in the SPCTLx registers selects the logic level of the frame sync signals as active low (inverted) if set (=1) or active high if cleared (=0). Active high (=0) is the default.

Frame Sync Generation

A frame sync pulse marks the beginning of the data word. There are some conditions related to this signal which are discussed in the following sections.

Data-Independent Frame Sync

When DIFS = 0 and SPTRAN = 1, the internally-generated transmit frame sync is only output when a new data word has been loaded into the SPORT channel's transmit buffer. Once data is loaded into the transmit buffer, it is not transmitted until the next frame sync is generated. This mode of operation allows data to be transmitted only at specific times. When DIFS = 0 and SPTRAN = 0, a receive SPORTx_FS signal is generated only when receive data buffer status is not full.

If the internally-generated frame sync is used with DIFS = 0, any core write to the transmit buffer starts the FS and data transfer.

The data-independent frame sync mode allows the continuous generation of the SPORTx_FS signal, with or without new data in the buffers. The DIFS bit of the SPCTLx control register configures this option. When DIFS = 1 and SPTRAN = 1, a transmit SPORTx_FS signal is generated regardless of the transmit data buffer status. When DIFS = 1 and SPTRAN = 0, a receive SPORTx_FS signal is generated regardless of the receive data buffer status.

Note that the frame sync pulse marks the beginning of the data word. If DIFS is set, the frame sync pulse is issued on time, whether the transmit buffer has been loaded or not. If DIFS is cleared, the frame sync pulse is only generated if the transmit buffer has been loaded. If the receiver demands regular frame sync pulses, DIFS should be set, and the processor should keep loading the transmit buffer on time. For DIFS = 1, the core or DMA controller is responsible for streaming data to/from the buffers. If the real time requirements are not meet accordingly, the DERR_x error channel bit is set.

Channel Dependency

In addition to the DIFS bit, FS generation may be dependent on the buffer status of both channels. In standard protocol the setting of the FS_BOTH bit defines the logical conditions as follows.

- 0 = A OR B buffer update required for FS generation.
- 1 = A AND B buffer updates required for FS generation.

For multichannel/packed modes the FS_BOTH bit is internally cleared. For I^2S/left-justified protocol mode control is done internally as follows.

- A OR B if one channel enabled.
- A AND B if both channels enabled.

Frame Sync Error Detection

The SPORTs are capable of detecting underflow and overflow buffer errors as well as frame sync errors. They can detect frame syncs that occur before the last transmission or reception completes.

When a serial port is receiving or transmitting, its bit count is set to a word length (for example SLEN = 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.

Internal Frame Sync Errors

Internal FS errors occur due to programming faults:
SLEN > Frame

External Frame Sync Errors

Unexpected external FS errors are detected if: SLEN > Frame
Error FS pulse only during a data transmission

As shown in Figure 11-9, the frame sync error (which sets the error bit) is triggered when an early frame sync occurs during data transmission or reception or for late frame sync if the period of the frame sync is smaller then the serial word length (SLEN). However, the current transmit/receive operation continues without interruption.

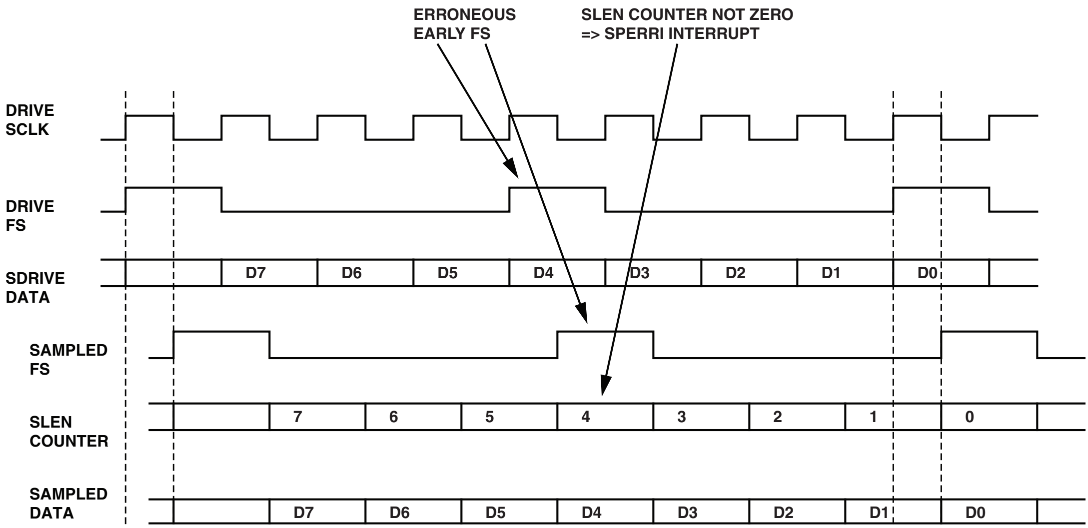

Figure 11-9. Frame Sync Error Detection

Note that a frame sync error is not detected in following cases.

- When there is no active data transmit/receive (SLEN counter is 0) and the frame sync pulse occurs due to noise in the input signal. It will be considered as a valid frame sync.

- If there is already a buffer underflow error. The SPORT error logic does not operate (the bit count is not set and decremented) if there is a buffer error.
- When the frame sync pulse < SCLK period.
- If the SPORT is operating in TDM slave mode, the frame sync must be at the start of new frame for one SCLK cycle active then inactive. If using duty cycles of for example 50%, the FS error bits (SPERRSTAT) get automatically set.

Data Transfers

Serial port data can be transferred for use by the processor in two different methods:

- Core-driven word transfers
- DMA transfers between SPORTs and internal or external memory

DMA transfers can be set up to transfer a configurable number of serial words between the serial port buffers (TXSPxA, TXSPxB, RXSPxA, and RXSPxB) 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 buffers (TXSPxA, TXSPxB, RXSPxA, and RXSPxB).

Serial Shift Registers

The following sections describe the SPORT shift registers.

Output Shift Register

The transmit shift register receives from 3 to 32-bit data and serially shifts its data out externally off chip. The transmit shift register is clocked with the driving edge.

Input Shift Register

The receive shift register receives its data serially from off chip. Internally the receive shift register is byte wide and data received can either be transferred to the FIFO buffer or used in an address comparison. The receive shift register is clocked with the sampling edge.

Buffers

When programming the serial port channel (A or B) as a transmitter (`SPTRAN` = 1), only the corresponding `TXSPxA` and `TXSPxB` buffers become active while the receive buffers `RXSPxA` and `RXSPxB` remain inactive. Similarly, when the SPORT channel A and B are programmed as receive-only (`SPTRAN` = 0) the corresponding `RXSPxA` and `RXSPxB` are activated. Do not attempt to read or write to inactive data buffers. If the processor operates on the inactive transmit or receive buffers while the SPORT is enabled, unpredictable results may occur.

Word lengths of less than 32 bits are automatically right-justified in the receive and transmit buffers.

Transmit Buffers

The transmit buffers (`TXSP7-0A`, `TXSP7-0B`) are the 32-bit transmit data buffers for SPORT7–0 respectively. These buffers must be loaded with the data to be transmitted if the SPORT is configured to transmit on the A and B channels. The data is loaded automatically by the DMA controller or loaded manually by the program running on the processor core.

The transmit buffers act like a two-location buffer because they have a data register plus an output shift register. Two 32-bit words may both be stored in the transmit queue at any one time. When the transmit register is loaded and any previous word has been transmitted, the register contents are automatically loaded into the output shifter. An interrupt occurs when the output transmit shifter has been loaded, signifying that the transmit buffer is ready to accept the next word (for example, the transmit buffer is not full). This interrupt does not occur when serial port DMA is enabled or when the corresponding mask bit in the LIRPTL/IRPTL register is set.

Receive Buffers

The receive buffers (RXSP7-0A, RXSP7-0B) are the 32-bit receive data buffers SPORT7–0 respectively. These 32-bit buffers become active when the SPORT is configured to receive data on the A and B channels. When a SPORT is configured as a receiver, the RXSPxA and RXSPxB registers are automatically loaded from the receive shifter when a complete word has been received. The data is then loaded to internal memory by the DMA controller or read directly by the program running on the processor core.

Buffer Packing

Received data words of 16 bits or less may be packed into 32-bit words, and transmitted 32-bit words may be unpacked into 16-bit words. Word packing and unpacking is selected by the PACK bit in the SPCTLx control registers.

When PACK = 1, two successive received words are packed into a single 32-bit word, and 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. Companding can be used with word packing or unpacking.

When serial port data packing is enabled, the transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.

When 16-bit received data is packed into 32-bit words and stored in normal word space in the processor's internal memory, the 16-bit words can be read or written with short word space addresses.

Companding

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. This operation requires one peripheral clock cycle of overhead, as described below. For companding to execute properly, program the SPORT registers prior to loading data values into the SPORT buffers.

Note that companding is hard coded for the A channels only and is directional relative to the SPORT number (0, 2, 4, 6 transmit and 1, 3, 5, 7 receive).

Buffer Status

Serial ports provide status information about data buffers via the `DXS_A` and `DXS_B` status bits and error status via `DERR_x` bits in the `SPCTL` register. Depending on the `SPTRAN` setting, these bits reflect the status of either the `TXSPxy` or `RXSPxy` data buffers.

If your program causes the core processor to attempt to read from an empty receive buffer or to write to a full transmit buffer, the access is delayed until the buffer is accessed by the external I/O device. This delay is called a core processor hang. If you do not know if the core processor can access the receive or transmit buffer without a hang, the buffer's status should be read first (in `SPCTLx`) to determine if the access can be made.

The status bits in `SPCTLx` are updated during reads and writes from the core processor even when the serial port is disabled.

If the SPORTs are configured as transmitters, programs should not write to the inactive TXSPxA and TXSPxB buffers. If the core keeps writing to the inactive buffer, the transmit buffer status becomes full. This causes the core to hang indefinitely since data is never transmitted to the output shift register.

If the SPORTs are configured as receivers, programs should not read from the inactive RXSPxA and RXSPxB buffers. If the core keeps reading from to the inactive buffer, the receive buffer status becomes empty. This causes the core to hang indefinitely since new data is never received via the input shift register.

The status bits in SPCTLx are updated during reads and writes from the core processor even when the serial port is disabled.

Buffer Errors

The following sections describe error conditions in the buffers. For more information see also "Interrupts" on page 11-52.

Reception Error

Two complete 32-bit words can be stored in the receive buffer while a third word is being shifted in. The third word overwrites the second if the first word has not been read out (by the processor core or the DMA controller). When this happens, the receive overflow status bit is set in the serial port control register. Almost three complete words can be received without the receive buffer being read before an overflow occurs. The overflow status is generated on the last bit of the third word. The DERR_x status bits are sticky and are cleared only by disabling the serial port.

Transmission Error

Whenever the SPORT is required to transmit and the transmit buffer is empty. the underflow status bit is set (DERR_x). The DERR_x status bits are sticky and are cleared only by disabling the SPORT or by writing to the corresponding RW1C bits in the SPERRCTL register.

Flushing Buffers

The SPORT buffers are flushed by disabling the serial port or by writing to the RW1C error bits in the SPERRCTL register.

Core Transfers

The following sections provide information on core driven data transfers.

Individual data words may also be transmitted and received by the serial ports, with interrupts occurring as each 32-bit word is transmitted or received. When a serial port is enabled and DMA is disabled, the SPORT interrupts are generated whenever a complete 32-bit word has been received in the receive buffer, or whenever the transmit buffer is not full.

When performing core-interrupt driven access, (FS master DIFS=1 or external FS), the FS is generated (DIVx register) regardless of buffer status. Therefore the buffer access is in the responsibility of the application which must meet real time requirements. The enabled interrupt is triggered if the transmit buffer has vacancy or the receive buffer new data. Any real time violation can be reported with the DERR-x bits to trigger an SPERRI exception.

If interrupts are disabled, to avoid hanging the processor core, check the buffer's full/empty status when the processor core's program reads a word from a serial port's receive buffer or writes a word to its transmit buffer. The full/empty status can be read in the DXS bits of the SPCTLx register. Reading from an empty receive buffer or writing to a full transmit buffer causes the processor to hang, while it waits for the status to change.

When performing core-driven transfers with DIFS=0, the first buffer access starts FS generation. This mode of operation allows data to be transmitted only at specific times.

If using multichannel/packed mode active channel section registers should be enabled to prevent any core hang situations.

DMA Transfers

SPORT DMA provides a mechanism for receiving or transmitting an entire block of serial data before the interrupt is generated. When serial port 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.

Each transmitter and receiver has its own DMA registers. The same DMA channel drives the left and right I^2S channels for the transmitter or the receiver. The software application must stop multiplexing the left and right channel data received by the receive buffer, because the left and right data are interleaved in the DMA buffers.

The SPORT DMA channels are assigned by default higher priority (fixed DMA channel priority enabled by default `DCPR` bit in `SYSCTL` register) 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 as shown in Table 3-29 on page 3-39.

Due to the possible priority of other DMA channels if the DMA controller is not able to load the transmit buffer with the actual value from memory or read the actual value from receive buffer, then the previous value is transmitted/received. The error status DERR_x bit will report any exception by using the SPORT error interrupt (SPERRI). Note if the DMA transfers have completed, the FS continues to drive.

For standard serial, I^2S and left-justified modes, the frame sync generation is optional.

SPORT DMA Group Priority

Each SPORT module has 2 DMA channels (A and B). Two SPORT modules represent a SPORT group (SPORT10, SPORT32, SPORT54 and SPORT76) for DMA access.

When a SPORT group (for example 4AB and 5AB channels) have data ready, the channel arbitrates by fixed priority method odd over even SPORT and A over B channel (which is the first arbitration stage). The winning channel requests the DMA bus arbiter to get control of the peripheral DMA bus (2nd stage of arbitration) or to the SPEP bus arbiter if access to external memory is required.

For the I/O processor, only the SPORT groups are requesting for the peripheral bus. For more information, see "Peripheral DMA Arbitration" on page 3-36.

Standard DMA

To set up a serial port DMA channel, write a set of memory buffer parameters to the SPORT DMA parameter registers as shown in Table 3-15 on page 3-15.

Load the IISPxy, IMSPxy, and CSPxy registers with a starting address for the buffer, an address modifier, and a word count, respectively. The register contains the internal memory address for transfers to internal memory and the external memory address for transfers to external memory. For DMA-driven transfers, the serial port logic performs the data transfer from internal memory to/from the appropriate buffer depending on the SPTRAN bit setting.

When both SPORT A and B channels are used in I^{2}S/left-justified mode with standard DMA enabled, then the DMA count should be the same for both channels.

Each SPORT DMA channel has a DMA enable bit (SDEN_A and SDEN_B) in its SPCTLx register. When DMA is disabled for a particular channel, the SPORT generates an interrupt every time it receives a data word or whenever there is a vacancy in the transmit buffer. For more information, see Table 3-15 on page 3-15.

Once serial port DMA is enabled, the processor's DMA controller automatically transfers received data words in the receive buffer to the buffer in internal or external memory, depending on the transfer type. Likewise, when the serial port is ready to transmit data, the DMA controller automatically transfers a word from internal or external memory to the transmit buffer. The controller continues these transfers until the entire data buffer is received or transmitted.

Therefore, set the direction bit, the serial port enable bit, and DMA Enable bits before initiating any operations on the SPORT data buffers. If the processor operates on the inactive transmit or receive buffers while the SPORT is enabled, it can cause unpredictable results.

Although the word lengths can be 3 to 32 bits, transmitting or receiving words smaller than 7 bits at the full clock rate of the serial port may cause incorrect operation when DMA chaining is enabled. Chaining locks the processor's internal I/O bus for several cycles while the new transfer control block (TCB) parameters are being loaded. Receive data may be lost (for example, overwritten) during this period. Moreover, transmitting or receiving words smaller than five bits may cause incorrect operation when all the DMA channels are enabled in standard DMA mode.

DMA Chaining

Each channel also has a DMA chaining enable bit (SCHEN_A and SCHEN_B) in its SPCTLx control register.

Each SPORT DMA channel also has a chain pointer register (CPSPxy). The CPSPxy register functions are used in chained DMA operations.

In chained DMA operations, the processor's DMA controller automatically sets up another DMA transfer when the contents of the current buffer have been transmitted (or received). The chain pointer register (CPSPxy) functions as a pointer to the next set of buffer parameters stored in external or internal memory. The DMA controller automatically downloads these buffer parameters to set up the next DMA sequence. For more information on SPORT DMA chaining, see "Chained DMA" on page 3-32.

DMA chaining occurs independently for the transmit and receive channels of each serial port. Each SPORT DMA channel has a chaining enable bit (SCHEN_A or SCHEN_B) that when set (= 1), enables DMA chaining and when cleared (= 0), disables DMA chaining. Writing all zeros to the address field of the chain pointer register (CPSPxy) also disables chaining.

The chain pointer register should be cleared first before chaining is enabled.

The I/O processor responds by auto-initializing the first DMA parameter registers with the values from the first TCB, and then starts the first data transfer.

Note that in chained mode for DIFS = 0, setting the SPEN bit starts first loading the first TCB which fills up the transmit buffer. Since the buffer is non empty the first FS is driven.

DMA Chain Insertion Mode

It is possible to insert a single SPORT DMA operation or another DMA chain within an active SPORT DMA chain. Programs may need to perform insertion when a high priority DMA requires service and cannot wait for the current chain to finish.

When DMA on a channel is disabled and chaining on the channel is enabled, the DMA channel is in chain insertion mode. This mode allows a program to insert a new DMA or DMA chain within the current chain without effecting the current DMA transfer.

Chain insertion mode operates the same as non-chained DMA mode. When the current DMA transfer ends, an interrupt request occurs and no TCBs are loaded. This interrupt request is independent of the `PCI` bit state.

Chain insertion should not be set up as an initial mode of operation. This mode should only be used to insert one or more TCBs into an active DMA chaining sequence. For more information, see “Enter DMA Chain Insertion Mode” on page 11-59.

SPORT DMA to External Memory

In previous SHARC processors, transferring data from a SPORT to external memory required placing that data temporarily in internal memory and then transferring it to external memory using DMA. The ADSP-214xx processors (Figure 11-2 on page 11-15) allow direct DMA transfers between SPORTs and external memory which removes this overhead, freeing up the core and internal memory for other peripherals. The SPORT DMA index and chain pointer registers have been expanded to be able to hold the external memory address.

SPORT SPEP Bus Priority

The SPORT groups which have external memory DMA access must arbitrate first for the SPEP bus which connects the SPORT to the external port interface. The priority for the SPEP bus is optional, the DCPR bit (SYSCTL register) defines if the priority is fixed or rotating.

For the I/O processor, the 4 DMA channels are considered as a group and one arbitration request. For more information, see "Peripheral DMA Arbitration" on page 3-36.

Interrupts

Table 11-12 provides an overview of SPORT interrupts.

Table 11-12. SPORT Interrupt Overview

Default Programmable Interrupt	Sources	Masking	Service
SP1I = P3I SP3I = P4I SP5I = P5I SP0I = P7I SP2I = P8I SP4I = P9I SP7I = P11I SP6I = P16I	DMA complete Core buffer service request Internal transfer completion Access completion	N/A	RTI instruction
	Buffer underflow Buffer overflow Unexpected frame sync		RW1C to SPERRCTLx + RTI instruction

Sources

The SPORT module generates three local interrupt signals—one for each data channel (A and B) signal and a third used for error detection.

The data channel interrupts are both logically ORed into one SPORT interrupt signal and the error detection interrupt is logically ORed with all SPORTs into one signal, SPERRI. The serial ports generate interrupts as described in the following sections.

Core Buffer Service Request

When DMA is disabled the processor core may read from the RXSPx buffer or write to the TXSPx buffer. An interrupt is generated when the receive buffer is not empty or the transmit buffer is not full.

An interrupt occurs when the output transmit shifter has been loaded, signifying that the transmit buffer is ready to accept the next word (for example, the transmit buffer is not full). This interrupt does not occur when serial port DMA is enabled. An interrupt is generated when the receive buffer has been loaded with a received word (for example, the receive buffer is not empty).

Data Buffer Packing

When serial port data packing is enabled (PACK = 1 in the SPCTLx registers), the transmit and receive interrupts are generated for 32-bit packed words, not for each 16-bit word.

DMA Complete

When serial port data packing is enabled (PACK = 1 in the SPCTLx registers), the transmit and receive interrupts are generated for 32-bit packed words, not for each 16-bit word.

Internal Transfer Complete

Interrupts can be used to indicate the completion of the transfer of a block of serial data when the serial ports are configured for DMA. Each DMA channel has a count register (CSPxA/CSPxB), which 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, the SPORT generates an interrupt, then automatically stops the DMA channel.

Access Complete

The SPORT DMA interrupt can be programmed to be generated either when the transmit DMA count is expired (ETDINTEN = 0) or when the last bit of the last word is shifted out (ETDINTEN = 1).

For chained DMA where ETDINTEN = 1:

- If PCI = 0, the interrupt is generated after that last word of last DMA block in the chain is shifted out.
- If PCI = 1, the interrupt is generated when the DMA counter expires for the initial DMA blocks in the chain and the last bit of the last word is shifted out for the last DMA block in the chain (CP is nonzero).
- For receive DMA, the interrupt behaves in the same way, independent of the value of ETDINTEN bit.

Chained DMA

For chained DMA, if the PCI bit is cleared (= 0), the DMA complete interrupt is generated only after the entire chained DMA access is complete. If the PCI bit is set (= 1), then a DMA interrupt is generated for each TCB.

Buffer Over/Underflow

Each SPORT can generate an interrupt if a channel buffer (DERRA_STAT, DERRB_STAT bits) or frame sync error (FSERR_STAT bit) occurs. These bits are located in the SPERRCTLx register.

Similar to previous SHARC processors, the SPORTs can return the status of data buffer underflow and overflow conditions. Additionally, the SPORTs can also detect unexpected frame syncs occurring early, even before the last transmit or receive completes. An error interrupt is triggered on a data underflow/overflow, or frame sync error in their respective channels.

Unexpected Frame Sync Errors

Additionally, the SPORTs can also detect frame syncs that are occurring early, even before the last transmit or receive completes.

Masking

The SPORTxI signals are routed by default to programmable interrupts as described in the list below.

- To service SPORTs 1, 3, 5, 6, unmask (set = 1) the P3I, P4I, P5I, and P16I bits in the IMASK register.
- To service SPORTs 0, 2, 4, 7, unmask (set = 1) the P7IMSK, P8IMSK, P9IMSK, P11IMSK bits in the LIRPTL register.
- To service the SPERRI interrupt, unmask (set = 1) the SPERRI bit in the IMASK register.

For example:

```
bit set IMASK P3I;          /* unmasks P3I interrupt */
bit set LIRPTL P7IMSK;      /* unmasks P7I interrupt */
bit set IMASK SPERRI;       /* unmasks SPERRI interrupt */
```

Similar to previous SHARC processors, the SPORTs report the status of the data buffers (transmit and receive). The DERRx_EN bits (SPERRCTLx register) enable the specific A or B channels.

Additionally, the SPORTs can detect frame syncs that are occurring early, even before the last transmit or receive completes. To detect these errors, enable the FSERR_EN bit in the SPERRCTLx) register.

The error interrupts must be unmasked by setting the SPERRI bit in the IMASK register.

SPORT interrupts occur on the second peripheral clock (PCLK) after the last bit of the serial word is latched in or driven out.

Service

Both the A and B channels share a common interrupt vector in the interrupt-driven data transfer mode, regardless of whether they are configured as a transmitter or receiver.

The SPORT generates an interrupt when the transmit buffer has a vacancy or the receive buffer has data. To determine the source of an interrupt, programs must check the transmit or receive data buffer status bits (DXS_A, DXS_B) in the SPCTLx registers. For DMA programs the corresponding status bits in the SPMCTLx registers must be checked. Note that in most cases, if both channels are enabled with the same DMA count, there is no need to check the status since both channel interrupts are generated close to each other.

Standard DMA does not function properly in I^{2}S/left-justified mode when two channels (A and B) are enabled with different DMA count values. In this case, the interrupt is generated for the least count only. If both the A and B channels of the SPORTs are used in I^{2}S/left-justified mode with DMA enabled, then the DMA count value should be the same for both channels. This does not apply to chained DMA.

One error interrupt is connected for all SPORTs together. Therefore, when an error occurs, programs should first read the global error status interrupt register, SPERRSTAT, to identify which SPORT caused the error condition. The next step requires a RW1C (write 1 to clear) operation to the corresponding error bit in the local SPERRCTLx register. This operation also clears the SPERRSTAT bits.

For frame sync errors, clear the FSERR_STAT bit (SPERRCTL register) which resets both channel bits SPENx or MCEx. For buffer channel errors, clear the DERRx_STAT bit (SPERRCTL register) which resets the channel bits (SPENx or MCEx) and DERRx. For example:

```
ISR_SPERRI:
ustat1 = dm(SPERRSTAT);      /* read error status from SP7-0 */
bit TST ustat1 SP2_DERRA;    /* identify SPORT and cause */
If TF jump SP2_ERROR;
…
SP2_ERROR:
ustat3=dm(SPERRCTL2);
bit set ustat3 DERRA_STAT;
dm(SPERRCTL2)=ustat3;       /* RW1C buffer status error bit */
r5=dm(SPCTL2);              /* dummy read for latency */
rti;
```

Throughput

When the SPORT is operating in half-duplex mode, and both channels (A and B) are active, each SPORT has 112 M bit/s maximum total throughput.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

SPORT Effect Latency

After a write to a SPORT control register, control and mode bit changes take effect in the second serial clock cycle (SCLK).

The SPORT is ready to start transmitting or receiving three serial clock cycles after it is enabled in the SPCTLx control register. No serial clocks are lost from this point on. This delay also applies in slave mode (external clock/frame sync) for synchronization.

Multichannel and packed operation is activated three serial clock cycles (SCLK) after the MCEA or MCEB bits are set. Internally generated frame sync signals activate four serial clock cycles after the MCEA or MCEB bits are set.

Programming Model

The section describes some programming procedures that are used to enable and operate the SPORTs.

Setting Up and Starting DMA Master Mode

To set up and initiate a master DMA operation, use the following procedure.

1. Clear the SPORT control registers (SPCTLx/SPMCTLx) which flushes the buffers.
2. Write to the appropriate DIVx register, setting the master clock and frame sync ratios.
3. Configure all DMA parameter registers (index, modify and count).
4. Configure the SPORT protocol mode and enable DMA operation (SPCTLx).

Setting Up and Starting Chained DMA

To set up and initiate a chain of DMA operations, use the following procedure.

1. Clear the chain pointer register.
2. Clear the SPORT control registers (SPCTLx/SPMCTLx) which flushes the buffers.
3. For internal memory transfers, set up all TCBs in internal memory.
4. Write the address containing the index register value of the first TCB to the chain pointer register, which starts the chain.
5. Write to the SPCTLx register by setting the DMA enable bit to one and the chaining enable bit to one. Setting these bits loads the DMA parameter registers.

Enter DMA Chain Insertion Mode

Chain insertion lets the SPORTs insert a single SPORT DMA operation or another DMA chain within an active SPORT DMA chain.

1. Enter chain insertion mode by setting SDENx = 0 and SCHENx = 1 in the channel's DMA control register. The DMA interrupt indicates when the current DMA sequence is complete.
2. Copy the address currently held in the chain pointer register to the chain pointer position of the last TCB in the chain that is being inserted.
3. Write the start address of the first TCB of the new chain into the chain pointer register.
4. Resume chained DMA mode by setting SDENx = 1 and SCHENx = 1.

Setting Up and Starting Multichannel Mode

Unlike standard, left justified and I^2S protocols, configuring the SPORTs for TDM/packed protocol requires programming two control registers.

The SPCTL register is responsible for chained DMA control settings while the SPMCTL register is responsible for TDM control settings. As soon as the SPCTL register (SPENx bits cleared, DEN/CHEN bits set) and the chain pointer register are programmed, TCB loading (refer to IOP effect latency) starts which fills up the transmit buffer. However these samples are not transmitted until the MCEx bit (SPMCTL register) is set. Note that the DIFS bit is hard wired to 1 for TDM/packed protocols, setting the MCE bit in master mode automatically generates the FS.

Use the control registers (SPCTLx/SPMCTLx) and active channel selection registers (SPxCSy and SPxCCSy) to configure the serial ports to run in multichannel mode as follows. For proper data alignment on sports in multichannel mode, the multichannel enable bit must be set last since the DIFS bit is hard wired to 1.

1. Clear all control registers (SPCTLx/y and SPMTCLxy) and chain pointer registers.
2. For DMA operation, configure the DMA parameter registers (Index, Modify and Count). For DMA chaining setup the TCBs according to the chain.
3. Configure the receiver SPORTx control register of SPORT pair (SPCTLx) and enable the standard/chaining bits. Do not enable the SPENx bits.
4. Configure the transmitter SPORTx control register of a SPORTxy pair (SPCTLx) and enable the DMA/DMA chaining.
5. Configure the transmitter SPORTx control register of a SPORT pair (SPCTLx) and enable the standard/chaining bits. Do not enable the SPENx bits.

6. For DMA chaining, initialize the chain pointer register with the index register for the first chain to start chaining, filling up the transmit buffer.

7. Poll DMA chain loading complete status DMACHSxA bits OR the transmit buffer status DXS bit to be full.

8. Configure the number of channels, frame delay and enable MCEx bits for multichannel/packed mode for the SPORT pair (SPMCTLx).

Multichannel Mode Backward Compatibility

In previous SHARC models, the serial port pair used the same control register (SPMCTL01) to program multichannel mode. In the ADSP-214xx processors, this register is simply renamed to SPMCTL0 and a new identical register, SPMCTL1 has been added. Programs using the older code simply need to change from the SPMCTL01 register to the SPMCTL0 register or the SPMCTL1 register.

The following steps should be taken to port the code to the ADSP-214xx products.

1. Instead of programming SPMCTLxy only, program both SPMCTLx and SPMCTLy registers.

2. In previous ADSP-2136x processors the data direction bit in the SPCTL register is hard coded in multichannel mode (where the even port is always the transmitter and the odd port is always the receiver). From the ADSP-21367/8/9 processors onward, the direction (SPTRAN bit) is honored and therefore should be set as required.

3. Routing models for hard coded multichannel pairs use the odd RX SPORT for the clock and frame sync. The TDV signal was multiplexed with the even TX SPORT frame sync output. From the ADSP-21367/8/9 processors onward, these limitations no longer

apply. All SPORTs operate completely independently. Every SPORT requires the clock and frame sync to be routed. The TDV output signal is an independent output signal in the SRU unit.

Programming Packed Mode

Since packed mode is implemented on top of multichannel mode, programming this mode is the same as programming multichannel mode. Use the serial port control (SPCTLx) and multichannel selection registers (SPMCTLx) to configure the serial ports to run in packed mode as follows.

1. Clear all control registers (SPCTLx and SPMCTLx)
2. Configure the multichannel channel select registers (SPxCSy or SPxCCSy).
3. Set the OPMODE, CKRE bits in the SPCTLx register and (ICLK, IFS bits for master mode) to operate packed mode. The L_FIRST bit allows swapping left and right channels. Note the CKRE bit must be set in both receiver and transmitter. Clear the LSBF bit to run in packed mode.
4. Clear the LSBF bit to run in packed mode.
5. To emulate I^2S in packed mode, set the MFD bit field to one and the NCH bit field according to the channels in the SPMCTLx register.

The MFD bit field and the L_FIRST bit allow programs to manipulate the timing as follows.

- The MFD bit field selects the data delay in SCLK cycles after the frame sync occurred.
- The L_FIRST bit allows to swap the left and right channels.

External Frame Sync Operation

There are two procedures which allow programs to save SPORT initialization during an inactive frame sync:

- Read the DAI_PIN_STAT register of the frame sync to get the level prior to starting SPORT configuration.
- Route a MISCA register input to the external frame signal (rising or falling edge) as an interrupt trigger to generate an interrupt to start SPORT configuration.

In the ADSP-214xx processors the FSED bit in the SPCTLN register allows SPORT initialization regardless of the state of the external frame sync. The SPORT starts the transfer on the next valid rising or falling edge. This makes it easy to release the state machine and frame sync generators from reset. For example, if you configure the SPORT for a rising edge frame sync, there is no need to wait for a low/high level on the frame sync pin before releasing the SPORT state machine from reset.

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. This operation requires one peripheral clock cycle of overhead, as described below. 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 SPTRAN bit to 1 in the SPCTLx register. The SPEN_A and SPEN_B bits should be = 0.

2. Enable companding in the DTYPE field of the SPCTLx transmit control register.
3. Write a 32-bit data word to the transmit buffer. Companding is calculated in this cycle.
4. Wait two 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 (SLEN) in the SPCTLx register.

With companding enabled, interfacing the serial port to a codec requires little additional programming effort. If companding is not selected, two formats are available for received data words of fewer than 32 bits—one that fills unused MSBs with zeros, and another that sign-extends the MSB into the unused bits.

Debug Features

The following sections provide information on debugging features available with the serial ports.

SPORT Loopback

When the SPORT loopback bit, SPL (bit 12), is set in the SPMCTLx register, the serial port is configured in an internal loopback connection as follows: SPORT0/SPORT1 work as a pair, SPORT2/SPORT3 work as a pair, SPORT4/SPORT5 work as a pair and SPORT6/SPORT7 work as a pair.

Pairings of SPORTs (01, 23, 45 and 67) is only required for loopback mode which is valid for all non multichannel modes.

The SPL bit applies to all non multichannel modes.

The loopback mode enables programs to test the serial ports internally and to debug applications. In loopback mode, either of the two paired SPORTS can be transmitters or receivers. One SPORT in the loopback pair must be configured as a transmitter; the other must be configured as a receiver. For example, SPORT0 can be a transmitter and SPORT1 can be a receiver for internal loopback. Or, SPORT0 can be a receiver and SPORT1 can be the transmitter when setting up internal loopback.

LoopBack Routing

The SPORTs support an internal loopback mode by using the SRU. For more information, see "Loopback Routing" on page 10-40.

Buffer Hang Disable (BHD)

To support debugging buffer transfers, the processors have a buffer hang disable (BHD) bit. When set (= 1), this bit prevents the processor core from detecting a buffer-related stall condition, permitting debugging of this type of stall condition.

12 INPUT DATA PORT (SIP, PDAP)

The Input Data Port (IDP) compromises two units: the serial input port (SIP) and the parallel data acquisition port (PDAP). Located inside the DAI of the SHARC processor the IDP provides an efficient way of transferring data from DAI pin buffers, the external port, the asynchronous sample rate converters (ASRC) and the S/PDIF transceiver to the internal memory of SHARC. The IDP specifications are shown in Table 12-1.

Table 12-1. IDP Port Specifications

Feature	SIP	PDAP
Connectivity		
Multiplexed Pinout	No	Yes (External Port)
SRU DAI Required	Yes	Yes
SRU DAI Default Routing	No	No
SRU2 DPI Required	No	No
SRU2 DPI Default Routing	N/A	N/A
Interrupt Control	Yes	Yes
Protocol		
Master Capable	No	No
Slave Capable	Yes	Yes
Transmission Simplex	Yes	Yes
Transmission Half Duplex	No	No
Transmission Full Duplex	No	No

Table 12-1. IDP Port Specifications (Cont'd)

Feature	SIP	PDAP
Access Type		
Data Buffer	Yes	Yes
Core Data Access	Yes	Yes
DMA Data Access	Yes	Yes
DMA Channels	8	1
DMA Chaining	No	No
Boot Capable	No	No
Local Memory	No	No
Max Clock Operation	$f_{PCLK}/4$	$f_{PCLK}/4$

Features

The following list describes the IDP features.

- The IDP provides a mechanism for a large number of asynchronous channels (up to eight).
- The IDP supports industry standard data formats, I^2S, Left-justified and Right-justified for serial input ports.
- The PDAP supports four data packing modes for parallel data.
- The PDAP supports a maximum of 20-bits.
- Provides two data transfer types, through DMA or interrupt driven transfer by core.

Pin Descriptions

Table 12-2 provides descriptions of the IDP pins used for the serial interface port.

Table 12-2. SIP Pin Descriptions

Internal Node	I/O	Description
IDP7–0_CLK_I	I	**Serial Input Port Receive Clock Input.** This signal must be generated externally and comply to the supported input formats.
IDP7–0_FS_I	I	**Serial Input Port Frame Sync Input.** The frame sync pulse initiates shifting of serial data. This signal must be generated externally and comply to the supported input formats.
IDP7–0_DAT_I	I	**Serial Input Port Data Input.** Unidirectional data pin. Data signal must comply to the supported data formats.

Table 12-3 provides descriptions of the IDP pins used for the parallel interface port.

Table 12-3. PDAP Pin Descriptions

Internal Nodes	Type	Description
PDAP_CLK_I	I	**Parallel Data Acquisition Port Clock Input.** Positive or negative edge of the PDAP clock input is used for data latching depending on the IDP_PDAP_CLKEDGE bit (29) of the IDP_PP_CTL register. Note that input has multiplexed.
PDAP_HOLD_I	I	**Parallel Data Acquisition Port Frame Sync Input.** The PDAP hold signal determines whether the data is to be latched at an active clock edge or not. When the PDAP hold signal is HIGH, all latching clock edges are ignored and no new data is read from the input pins. The packing unit operates as normal, but it pauses and waits for the PDAP hold signal to be deasserted and waits for the correct number of distinct input samples before passing the packed data to the IDP FIFO. Note that the input has multiplexed control.
PDAP_DATA	I	**Parallel Data Acquisition Port Data Input.** The PDAP latches 20-bit parallel data which where packed into 32-bits by using different packing. Note that input has multiplexed control.

Table 12-3. PDAP Pin Descriptions (Cont'd)

Internal Nodes	Type	Description
PDAP_STRB_O	O	**Parallel Data Acquisition Port Clock input.** The PDAP packing unit asserts the output strobe whenever there is 32-bit data available for transfer to the IDP FIFO. The width of this pulse is equal to 2 x PCLK cycles. This signal can be used to synchronize external requests for new PDAP data. Note that input has multiplexed control.

Table 12-4 provides descriptions of the pin multiplexing between DAI and external port. For more information, see "Pin Multiplexing" on page 24-28.

Table 12-4. Pin Multiplexing between DAI and External Port

Signal	DAI	External Port
Serial Clock	IDP0_CLK_I	ADDR[2]
Frame Sync	IDP0_FS_I	ADDR[3]
Data	DAI_PB20–1	ADDR[23–4]
Strobe Out	PDAP_STRB_O	ADDR[0]

SRU Programming

The SRU (signal routing unit) needs to be programmed in order to connect the IDP to the SIP/PDAP as shown in Table 12-5.

Table 12-5. SIP SRU Signal Connections

SIP Source	DAI Connection	SIP Destination
No Sources	Group A	IDP7–0_CLK_I
	Group B	IDP7–0_DAT_I
	Group C	IDP7–0_FS_I

Table 12-6 shows the signal connections when using the PDAP on the DAI pins.

Table 12-6. PDAP SRU Signal Connections

PDAP Source	DAI Connection	PDAP Destination
	Group A	PDAP_CLK_I
	Group B	IDP0_DAT_I
	Group C	PDAP_HOLD_I
PDAP_STRB_O	Group D	

Register Overview

This section provides brief descriptions of the major registers. For complete information see "Input Data Port Registers" on page A-174.

IDP Control Registers (IDP_CTLx). The ADSP-2136x and ADSP-2137x SHARC processors have two IDP control registers. The IDP_CTL1-0 registers are used to control the SIP operations.

PDAP Control Register (IDP_PP_CTL). The register (shown in Figure 12-1) is used to control all PDAP operations.

IDP Status Register (DAI_STAT). Returns different types of status for SIP/PDAP core and DMA operations.

Clocking

The fundamental timing clock of the IDP module is peripheral clock/4 (PCLK/4). The IDP SIP/PDAP operates in slave mode only. The clock to this module may be shut off for power savings. For more information, see "Disabling Peripheral Clocks" on page 23-11.

Functional Description

The IDP provides up to eight serial input channels—each with its own clock, frame sync, and data inputs. The eight channels are automatically multiplexed into a single 32-bit by eight-deep FIFO. Data is always formatted as a 64-bit frame and divided into two 32-bit words. The serial protocol is designed to receive audio channels in I^2S, left-justified, or right-justified mode. One frame sync cycle indicates one 64-bit left-right pair, but data is sent to the FIFO as 32-bit words (that is, one-half of a frame at a time). Data transfer occurs on all channels from the SIP/PDAP to the IDP FIFO with fixed priority, from channel 0 (highest priority) to channel 7 (lowest priority). Transfers from this FIFO to internal memory can be performed either via DMA or by core interrupts.

IDP channel 0 is shared by SIP0 and PDAP. All other 7 SIPs are connected to corresponding IDP channel of FIFO.

The DMA engine of the IDP implements DMA for all the 8 channels. It has eight sets of DMA parameter registers for 8 channels. Data from channel 0 is directed to internal memory location controlled by set of registers for channel 0 and so on.

The parallel data is acquired through the parallel data acquisition port (PDAP) which provides a means of moving high bandwidth data to the core's memory space. The data may be sent to memory as one 32-bit word per input clock cycle or packed together (for up to four clock cycles worth of data).

Figure 12-1 provides a graphical overview of the input data port architecture. Notice that each channel is independent and contains a separate clock and frame sync input.

The IDP provides an easy way to pump serial data into on-chip memory since it is less complex than the traditional SPORT module, limited to unidirectional slave transfers only.

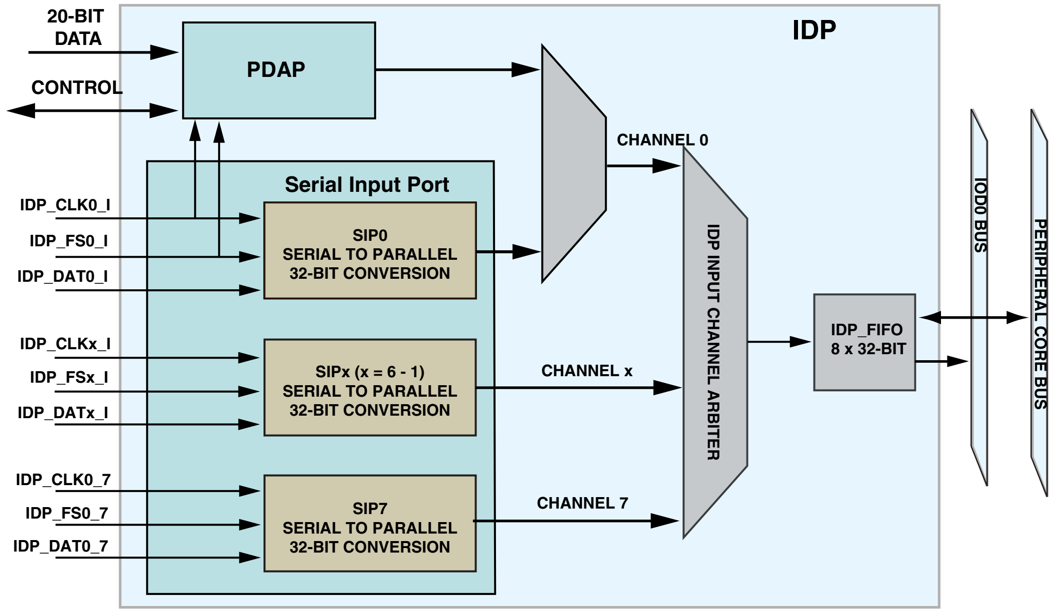

Figure 12-1. Input Data Port

Operating Modes

The following sections provide information on the various operation modes used by the PDAP module. The IDP has access to the IDP FIFO in the three modes listed below. The bit settings that configure these modes are shown in Table 12-7.

- Core mode (SIP/PDAP)
- DMA mode (SIP/PDAP)
- DMA ping-pong mode (SIP/PDAP)

Table 12-7. IDP Operation Modes

IDP Operation Modes	IDP_CTL0 Global Control		IDP_CTL1 Channel Control			IDP_PP_CTL PDAP Control	
	IDP_EN	IDP_ DMA_EN	IDP_ENx	IDP_ DMA_ENx	IDP_ PINGx	IDP_ PDAP_ EN	PDAP_PP _SELECT
Core SIP7–0	1	0	1	0	0	0	0
Core PDAP DAI	1	0	1	0	0	1	0
Core PDAP EP	1	0	1	0	0	1	1
DMA SIP7–0	1	1	1	1	0	0	0
DMA PDAP DAI	1	1	1	1	0	1	0
DMA PDAP EP	1	1	1	1	0	1	1
DMA Ping-pong SIP7–0	1	1	1	1	1	0	0
DMA Ping-pong PDAP DAI	1	1	1	1	1	1	0
DMA Ping-pong PDAP EP	1	1	1	1	1	1	1

PDAP Port Selection

The input to channel 0 of the IDP is multiplexed, and may be used either in the serial mode or in a direct parallel input mode. Setting the PDAP_EN bit high disables the connection of SIP0 to channel 0 of the FIFO. The data inputs can come either from the DAI pins or the external port ADDR pins. This is selected by the PDAP_PP_SELECT bit in the PDAP_CTL register.

Figure 12-2 illustrates the data flow for IDP channel 0, where either the PDAP or serial input can be selected.

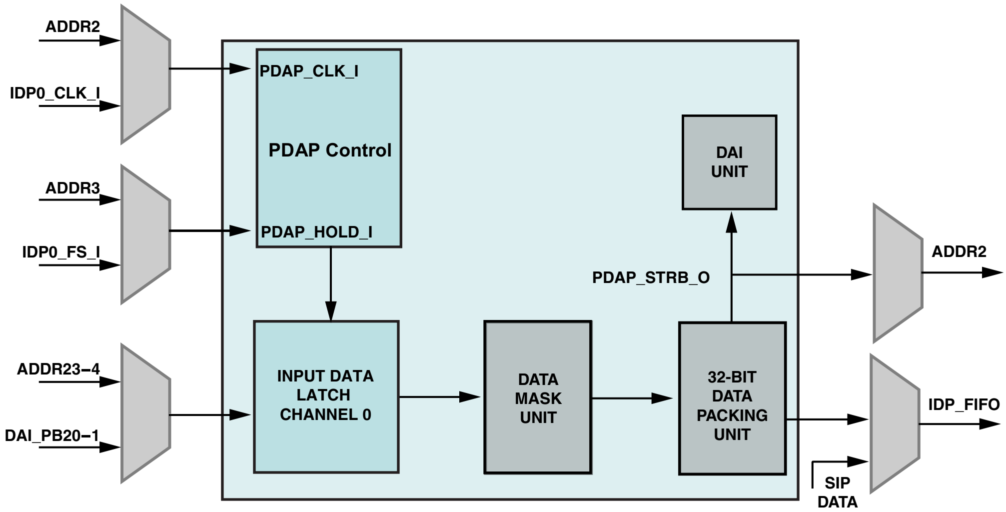

Figure 12-2. PDAP Port (Detail of IDP Channel 0)

Data Hold

When the PDAP_HOLD signal is high, all latching clock edges are ignored and no new data is read from the input pins. The packing unit operates as normal, but it pauses and waits for the PDAP_HOLD signal to be deasserted and waits for the correct number of distinct input samples before passing the packed data to the FIFO.

Figure 12-3 on page 12-11 through Figure 12-5 on page 12-13 show different packing modes including valid data hold inputs.

As shown in the figures, PDAP_DATA and PDAP_HOLD are driven by the inactive edges of the clock (falling edge in the above figures) and these signals are sampled by the active edge of the clock (rising edge in the figures).

PDAP Data Masking

For input data widths less than 20, inputs are aligned to the MSB pins. Additionally all PDAP inputs can be masked (IDP_PDAP_CTL register) to form user specific data streams from any input pins. Clearing the MASK bits (=0) disables data from the corresponding DAI or external port pin.

PDAP Data Packing

Multiple latched parallel sub word samples may be packed into 32-bit words for efficiency. The frame sync input is used to hold off latching of the next sample (that is, ignore the clock edges). The data then flows through the FIFO and is transferred by a dedicated DMA channel into the core's memory as with any IDP channel. As shown in Figure 12-2, the PDAP can accept input words up to 20 bits wide, or can accept input words that are packed as densely as four input words up to eight bits wide.

The IDP_PDAP_PACKING bits define the packing format. Based on the PDAP packing the data buffer format changes as shown in Figure 12-9.

No Packing

No packing provides for 20 bits coming into the packing unit and 32 bits going out to the FIFO in a single cycle. On every clock edge, 20 bits of data are moved and placed in a 32-bit register, left-aligned. That is, bit 19 maps to bit 31. The lower bits, 11–0, are always set to zero.

This mode sends one 32-bit word to FIFO for each input clock cycle—the DMA transfer rate matches the PDAP input clock rate.

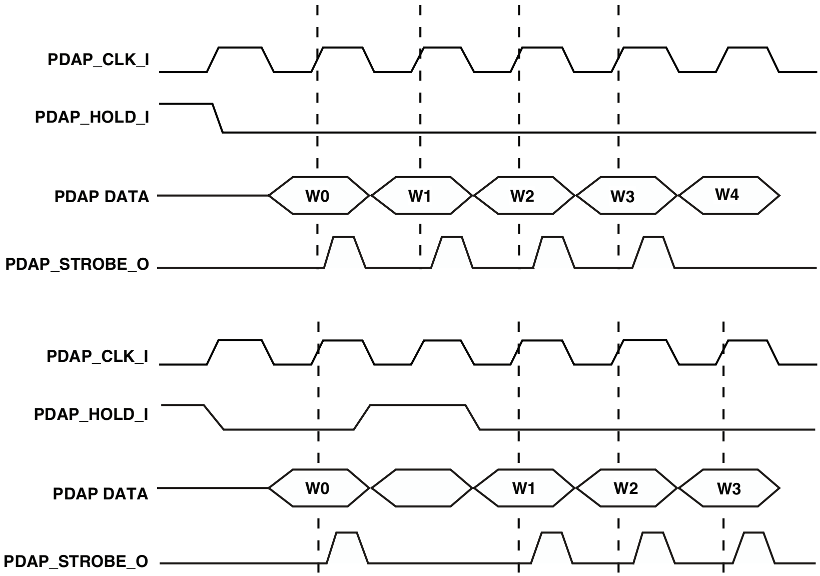

Figure 12-3. PDAP Hold Input (No Packing)

Packing by 2

Packing by 2 moves data in two cycles. Each input word can be up to 16 bits wide.

- On clock edge 1, bits 19–4 are moved to bits 15–0 (16 bits)
- On clock edge 2, bits 19–4 are moved to bits 31–16 (16 bits)

This mode sends one packed 32-bit word to FIFO for every two input clock cycles—the DMA transfer rate is one-half the PDAP input clock rate.

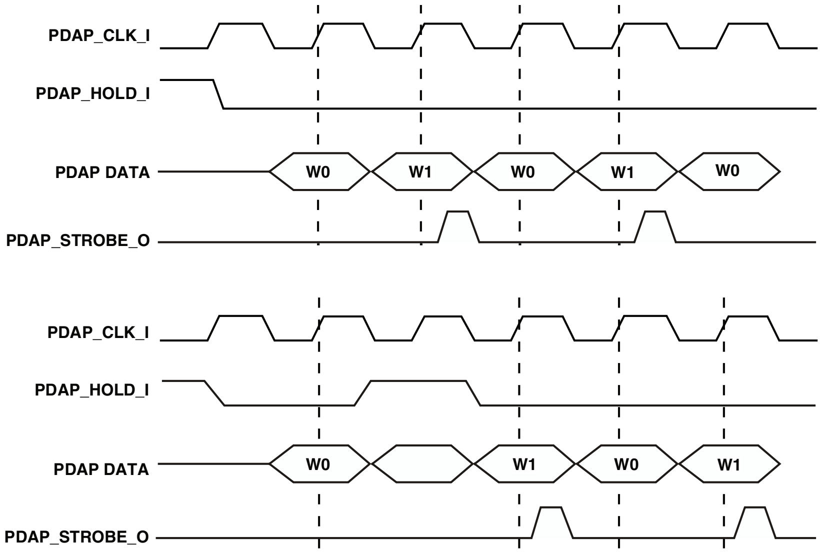

Figure 12-4. PDAP Hold Input (Packing by 2)

Packing by 3

Packing by 3 packs three acquired samples together. Since the resulting 32-bit word is not divisible by three, up to ten bits are acquired on the first clock edge and up to eleven bits are acquired on each of the second and third clock edges:

- On clock edge 1, bits 19–10 are moved to bits 9–0 (10 bits)
- On clock edge 2, bits 19–9 are moved to bits 20–10 (11 bits)
- On clock edge 3, bits 19–9 are moved to bits 31–21 (11 bits)

This mode sends one packed 32-bit word to FIFO for every three input clock cycles—the DMA transfer rate is one-third the PDAP input clock rate.

Packing by 4

Packing by 4 moves data in four cycles. Each input word can be up to 8 bits wide.

- On clock edge 1, bits 19–12 are moved to bits 7–0
- On clock edge 2, bits 19–12 are moved to bits 15–8
- On clock edge 3, bits 19–12 are moved to bits 23–16
- On clock edge 4, bits 19–12 are moved to bits 31–24

This mode sends one packed 32-bit word to FIFO for every four input clock cycles—the DMA transfer rate is one-quarter the PDAP input clock rate.

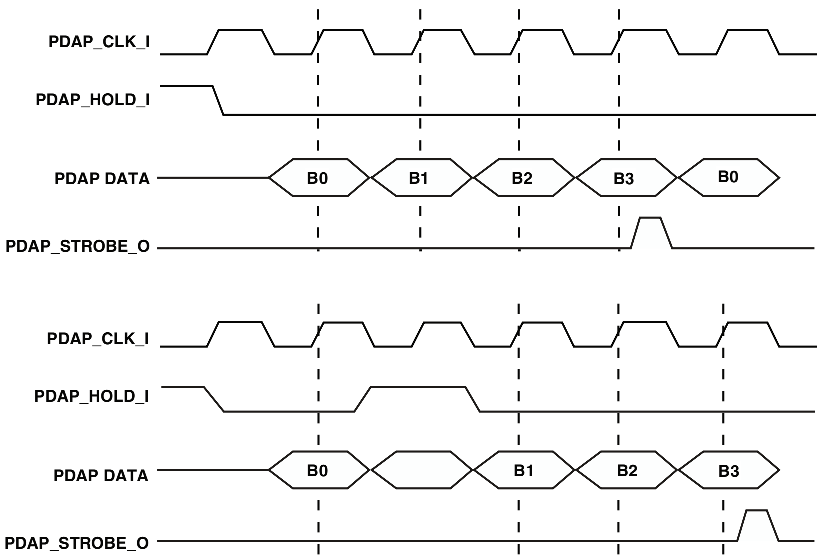

Figure 12-5. PDAP Hold Input (Packing by 4)

Data Transfer

The data from each of the eight IDP channels is inserted into an eight register deep FIFO, which can only be transferred to the core's memory space sequentially. Data is moved into the FIFO as soon as it is fully received. One of two methods can be used to move data from the IDP FIFO to internal memory:

- The core can remove data from the FIFO manually. This method of moving data from the IDP FIFO is described in the next section, "Core Transfers" on page 12-16.
- Eight dedicated DMA channels can sort and transfer data. This method of moving data from the IDP FIFO is described in "DMA Transfers" on page 12-19.

Buffers

The following sections provide information about the IDP buffers.

Buffer Threshold Depth

The IDP_FIFO register provides information about the output of the 8-deep IDP FIFO which have been filled by the SIP or the PDAP units. Normally, this register is used only to read and remove the top sample from the FIFO. Channel encoding provides for eight serial input types that correspond to the IDP_SMODEx bits in the IDP control registers. When using channels 0–7 in serial mode, this register format applies. When using channel 0 in parallel mode, refer to the description of the packing bits for PDAP mode.

The information in Table 12-8 is not valid when data comes from the PDAP channel.

Table 12-8. IDP_FIFO Register Bit Descriptions

Bit	Name	Description
2–0	CHAN_ENC	**IDP Channel Encoding.** These bits indicate the serial input port channel number that provided this serial input data. Note: This information is not valid when data comes from the PDAP.
3	LR_STAT	**Left/Right Channel Status.** Indicates whether the data in bits 31-4 is the left or the right audio channel as dictated by the frame sync signal. The polarity of the encoding depends on the serial mode selected in IDP_SMODE for that channel. See Table A-88 on page A-175.
31–4	SDATA	**Input Data (Serial).** Some LSBs can be zero, depending on the mode.

Buffer Status

The status of the IDP buffer at any time is reflected in the IDP_FIFOSZ bit field in the DAI_STAT0 register.

Buffer Error Status

The error status of the IDP buffer is reflected in the SRU_OVFx bit field in the DAI_STAT0 register. The error status can be cleared by setting the IDP_-CLROVF bit or by disabling the IDP port

Flushing the Buffer

The IDP buffers are flushed by disabling the IDP Port or by setting the IDP_FFCLR bit.

Buffer Hang Disable

For more information, see “Buffer Hang Disable” on page 12-33.

Core Transfers

The core transfers require that the serial peripheral at the SIP writes data to the IDP_DATAx_I pin (DATA or DAI pins for PDAP) according to the selected input format used. These data are automatically moved to the IDP_FIFO register without DMA intervention.

The output of the FIFO can be directly fetched by reading from the IDP_FIFO buffer. The IDP_FIFO buffer is used only to read and remove the top sample from the FIFO, which is a maximum of eight locations deep. When this register is read, the corresponding element is removed from the IDP FIFO, and the next element is moved into the IDP_FIFO register. A mechanism is provided to generate an interrupt when more than a specified number of words are in the FIFO. This interrupt signals the core to read the IDP_FIFO register.

The number of data samples in the FIFO at any time is reflected in the IDP_FIFOSZ bit field (bits 31-28 in the DAI_STAT0 register), which tracks the number of samples in FIFO.

The three LSBs of FIFO data are the encoded channel number. These are transferred "as is" for this mode. These bits can be used by software to decode the source of data.

The maximum data transfer width to internal memory is 32-bits, as in the case of PDAP data or I^2S and left-justified modes in single channel mode using 32 bits of data. Therefore, PDAP or I^2S and left-justified 32-bit modes cannot be used with other channels in the core/interrupt driven mode since no channel information is available in the data stream.

SIP Data Buffer Format

An audio signal that is normally 24 bits wide is contained within the 32-bit word. Four bits are available for status and formatting data (compliant with the IEC 90958, S/PDIF, and AES3 standards). An additional bit identifies the left-right one-half of the frame. If the data is not in IEC standard format, the serial data can be any data word up to 28 bits wide. Unlike DMA, the core requires a status information about which channel triggered the interrupt. It does this by reading the data buffer. The remaining three bits are used to encode one of the eight channels being passed through the FIFO to the core. The FIFO output may feed eight DMA channels, where the appropriate DMA channel (corresponding to the channel number) is selected automatically.

Regardless of mode, the L/R channel status bit (Bit 3) always specifies whether the data is received in the left channel or the right channel of the corresponding input frame, as shown in Figure 12-6.

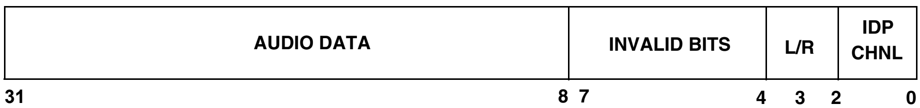

Figure 12-6. Principle Data Format for the SIP

Note that each input channel has its own clock and frame sync input, so unused IDP channels do not produce data and therefore have no impact on FIFO throughput. The clock and frame sync of any unused input should be routed by the SRU to low to avoid unintentional acquisition.

The framing format is selected by using the `IDP_SMODEx` bits (three bits per channel) in the `IDP_CTL0` register. Bits 31–8 of the `IDP_CTL0` register control the input format modes for each of the eight channels. The eight groups of three bits indicate the mode of the serial input for each of the eight IDP channels.

Figure 12-8 and Figure 12-7 shows the IDP data buffer input format for the SIP (depending on `SMODEx` bits) for core access.

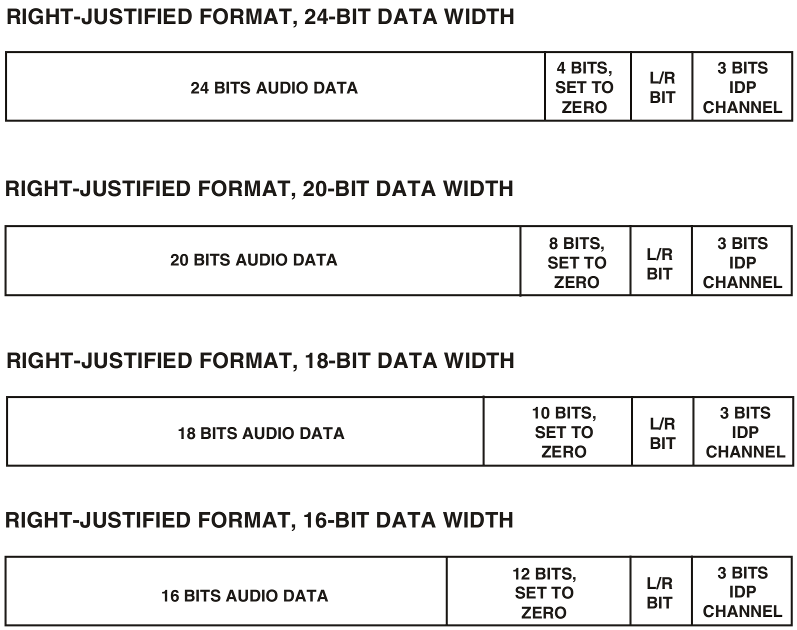

Figure 12-7. IDP Data Buffer Format SIP – Right-Justified

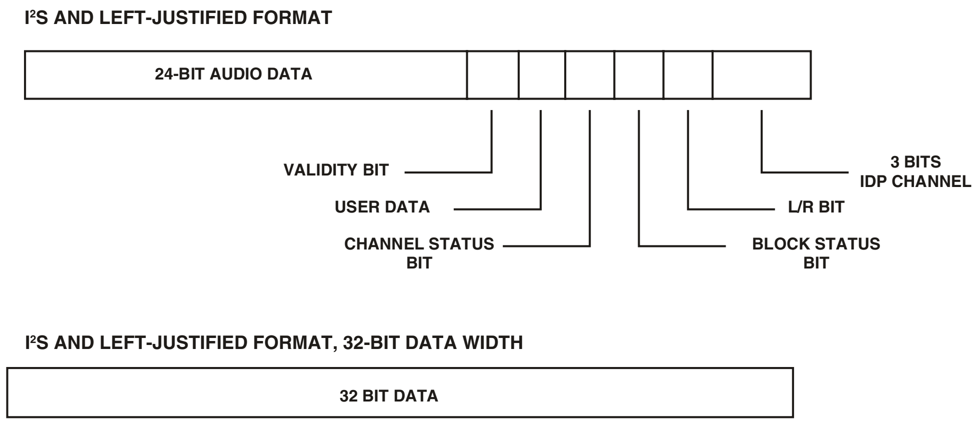

Figure 12-8. IDP Data Buffer Format SIP – I2S/Left-Justified (32 Bits)

The polarity of left-right encoding is independent of the serial mode frame sync polarity selected in IDP_SMODE for that channel (Table 12-3 on page 12-3). Note that I^{2}S mode uses a LOW frame sync (left-right) signal to dictate the first (left) channel, and left-justified mode uses a HIGH frame sync (left-right) signal to dictate the first (left) channel of each frame. In either mode, the left channel has bit 3 set (= 1) and the right channel has bit 3 cleared (= 0).

PDAP Data Buffer Format

If the PDAP module is enabled the IDP data buffer format will change according to the PDAP packing bits (IDP_PDAP_CTL register) as shown in Figure 12-9.

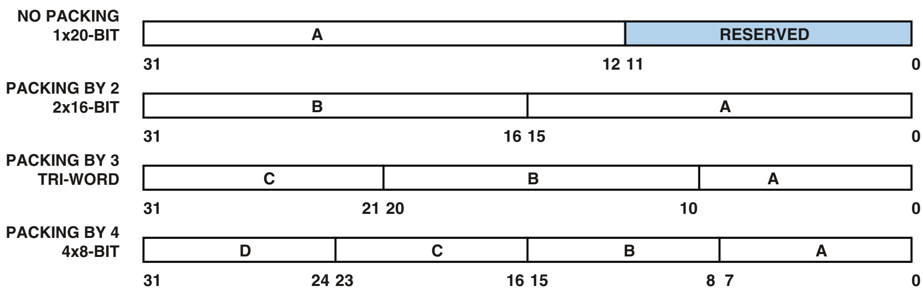

Figure 12-9. IDP Data Buffer Formats for the PDAP

DMA Transfers

The processors support two types of DMA transfers, standard and ping-pong. Eight dedicated DMA channels can sort and transfer the data into one buffer per source channel. When the memory buffer is full, the DMA channel raises an interrupt in the DAI interrupt controller.

Data Buffer Format for DMA

The LSB bits 2–0 of the data format from the serial inputs are channel encoding bits. Since the data is placed into a separate buffer for each DMA channel (defined by parameter index registers), these bits are not required and are cleared (=0) when transferring data to internal memory through the DMA. However, bit 3 still contains the left/right status information. In the case of PDAP data or 32-bit I^2S and left-justified modes, these three bits are a part of the 32-bit data.

For serial input channels, data is received in an alternating fashion from left and right channels. Data is not pushed into the FIFO as a full left/right frame. Rather, data is transferred as alternating left/right words as it is received. For the PDAP and 32-bit (non-audio) serial input, data is transferred as packed 32-bit words.

IDP DMA Group Priority

The IDP module can be configured with up to eight DMA channels. When multiple channels have data ready, the channel arbitrates using the fixed arbitration method (which is the first arbitration stage). The winning channel requests the DMA bus arbiter to get control of the peripheral DMA bus (2nd stage of arbitration).

The I/O processor considers the eight DMA channels as a single group and therefore one arbitration request. For more information, see “Peripheral DMA Arbitration” on page 3-36.

Standard DMA

The eight DMA channels each have a set of registers for standard DMA: an I (index register), an M (modify register), and a C (count register).

The IDP DMA parameter registers have these functions:

- **Internal index registers** (IDP_DMA_Ix). Index registers provide an internal memory address, acting as a pointer to the next internal memory location where data is to be written.
- **Internal modify registers** (IDP_DMA_Mx). Modify registers provide the signed increment by which the DMA controller post-modifies the corresponding internal memory Index register after each DMA write.
- **Count registers** (IDP_DMA_Cx). Count registers indicate the number of words remaining to be transferred to internal memory on the corresponding DMA channel.

This DMA access is enabled when the IDP_EN bit and IDP_DMA_EN bit and the IDP_DMA_ENx bits register are set to select a particular channel. The DMA is performed according to the parameters set in the various DMA registers and IDP control registers. An interrupt is generated after end of DMA transfer (when the count = 0).

Ping-Pong DMA

In ping-pong DMA, the parameters have two memory index values (index A and index B), one count value and one modifier value. The DMA starts the transfer with the memory indexed by A. When the transfer is completed as per the value in the count register, the DMA restarts with the memory location indexed by B. The DMA restarts with index A after the transfer to memory with index B is completed as per the count value.

The IDP DMA parameter registers have these functions:

- **Internal index registers** (IDP_DMA_IxA, IDP_DMA_IxB). Index A/B registers provide an internal memory address, acting as a pointer to the next internal memory location where data is to be written.

- **Internal modify registers** (IDP_DMA_Mx). Modify registers provide the signed increment by which the DMA controller post-modifies the corresponding internal memory Index register after each DMA write.
- **Ping-Pong Count registers** (IDP_DMA_PCx). Count registers indicate the number of words remaining to be transferred to internal memory on the corresponding DMA channel.

This mode is activated when the IDP_EN bit, the IDP_DMA_EN bit, the IDP_DMA_ENx bits, and the IDP_PINGx bits are set for a particular channel. An interrupt is generated after every ping and pong DMA transfer (when the count = 0).

Note that ping-pong DMA is repeated until stopped by resetting the IDP_DMA_ENx bits (OR global IDP_DMA_EN bit).

Multichannel DMA Operation

The SIP/PDAP can run both standard and ping-pong DMAs in different channels. When running standard DMA, initialize the corresponding IDP_DMA_Ix, IDP_DMA_Mx and IDP_DMA_Cx registers. When running ping-pong DMA, initialize the corresponding IDP_DMA_IxA, IDP_DMA_IxB, IDP_DMA_Mx and IDP_DMA_PCx registers.

DMA transfers for all 8 channels can be interrupted by changing the IDP_DMA_EN bit in the IDP_CTL0 register. None of the other control settings (except for the IDP_EN bit) should be changed. Clearing the IDP_DMA_EN bit (= 0) does not affect the data in the FIFO, it only stops DMA transfers. If the IDP remains enabled, an interrupted DMA can be resumed by setting the IDP_DMA_EN bit again. But resetting the IDP_EN bit flushes the data in the FIFO. If the bit is set again, the FIFO starts accepting new data.

Programs can drop DMA requests from the FIFO if needed. If one channel has finished its DMA, and the global IDP_DMA_EN bit is still set (=1), any data corresponding to that channel is ignored by the DMA machine. This feature is provided to avoid stalling the DMA of other channels, which are still in an active DMA state. To avoid data loss in the finished channel, programs can clear (=0) IDP_DMA_EN bit as discussed in previously.

Multichannel FIFO Status

The state of all eight DMA channels is reflected in the IDP_DMAx_STAT bits (bits 24–17 of DAI_STAT register). These bits are set once the IDP_DMA_EN and IDP_DMA_ENx bits are set, and remain set until the last data from that channel is transferred. Even if IDP_DMA_EN and IDP_DMA_ENx bits remain set, the IDP_DMAx_STAT bits clear once the required number of data transfers takes place.

Note that when a DMA channel is not used (that is, parameter registers are at their default values), the DMA channel's corresponding IDP_DMAx_STAT bit is cleared (= 0).

If the combined data rate from the channels is more than the DMA can service, a FIFO overflow occurs. This condition is reflected for each channel by the individual overflow bits (SRU_OVFx) in the DAI_STAT0 register. These are sticky bits that must be cleared by writing to the IDP_CLROVR bit (bit 6 of the IDP_CTL0 register). When an overflow occurs, incoming data from IDP channels is not accepted into the FIFO, and data values are lost. New data is only accepted once space is again created in the FIFO.

Interrupts

Table 12-9 provides an overview of IDP interrupts.

Table 12-9. IDP Interrupt Overview

Default Programmable Interrupt	Sources	Masking	Service
DAIHI = P0I DAILI = P12	DMA complete Core buffer service Buffer threshold Buffer overflow	DAI_IMASK_RE	ROC from DAI_IRPTL_x + RTI instruction

Sources

The IDP module drives in a total of 10 interrupt signals. Eight signals drive the DMA channel status and two are responsible for FIFO status (overflow and threshold buffer). These interrupts are connected to the DAI_IRPTL latch register.

The IDP port interface generates interrupts as described in the following sections.

Core Buffer Service Request

When DMA is disabled the processor core may read from the IDP_FIFO buffer. An interrupt is generated when the receive buffer is not empty.

Interrupt Acknowledge

The correct handling of the IDP interrupt requires that the ISR must read the DAI_IMASK_x register to clear the interrupt latch appropriately. Note that many interrupts are combined in the DAI interrupt. Refer to "Interrupts" on page 10-31.

Buffer Threshold

When using the interrupt scheme, the IDP_NSET bits (bits 3–0 of the IDP_CTL0 register) can be set to N, so N + 1 data can be read from the FIFO in the interrupt service routine (ISR). The IDP_FIFO_GTN_INT bit in DAI_IMASK_X register allows the IDP to configure interrupts to respond with the core under different system conditions.

DMA Complete

Using DMA transfers overrides the mechanism used for interrupt-driven core reads from the FIFO. When the IDP_DMA_EN bit and at least one IDP_DMA_ENx of the IDP_CTL1 register are set, the eighth interrupt (IDP_FIFO_GTN_INT) in the DAI_IMASK_x registers is NOT generated.

At the end of the DMA transfer for individual channels, interrupts are generated. These interrupts are generated after the last DMA data from a particular channel has been transferred to memory. These interrupts (IDP_DMAx_INT) are mapped from bits 17–10 in the DAI_IMASK_x registers and generate interrupts when they are set (= 1). These bits are ORed and reflected in high level interrupts that are sent to the DAI interrupt controller.

An interrupt is generated at the end of a DMA, which is cleared by reading the DAI_IMASK_x registers.

Buffer Overflow

If the data out of the FIFO (either through DMA or core reads) is not sufficient to transfer at the combined data rate of all the channels, then a FIFO overflow can occur. When this happens, new data is not accepted. Additionally, data coming from the serial input channels (except for 32-bit I^2S and left-justified modes) are not accepted in pairs, so that alternate data from a channel is always from left and right channels. If overflow occurs, an interrupt is generated if the IDP_FIFO_OVR_INT bit in the

DAI_IMASK_x register is set (sticky bits in DAI_STAT0 register are also set). Data is accepted again when space has been created in the FIFO.

Note that the total FIFO depth per channel is 9 locations: 1 location for SIP to parallel data conversion + 8 locations for the IDP_FIFO.

In case for DMA overflow error handling the DAISTAT0 sets overflow bits on the respective channel. A RW1C operation to the FIFO flush bit in the IDP_CTL0 register also clears the DAI_STAT0 bits.

Masking

The DAIHI and DAILI signals are routed by default to programmable interrupt. To service the DAIHI, unmask (set = 1) the P0I bit in the IMASK register. To service the secondary DAILI, unmask (set = 1) the P12IMSK bit in the LIRPTL register. For DAI system interrupt controllers the DAI_IMASK_RE register must be unmasked. For example:

```
bit set IMASK P0I;        /* unmasks P0I interrupt */
bit set LIRPTL P12IMSK;   /* unmasks P12I interrupt */
```

Service

The correct handling of the IDP interrupt requires that the ISR read the DAI_IRPTL_x register to clear the interrupt latch appropriately. Note that all IDP interrupts are combined in the DAI interrupt.

Note that the IDP_FIFO_GTN_INT interrupt is not cleared when the DAI_IRPTL_H/L registers are read. This interrupt is cleared automatically when the situation that caused the interrupt goes away.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

IDP Effect Latency

The IDP is ready to start receiving data one serial clock cycle (SCLK) after it is enabled by setting IDP_EN bit. No LRCLK edges are lost from this point on.

Disabling IDP DMA by resetting the IDP_DMA_EN bit requires 1 PCLK cycle. Disabling an individual DMA channel by resetting the IDP_DMA_ENx bit requires 2 PCLK cycles.

Programming Model

The following sections provide procedures that are helpful when programming the input data port.

Setting Miscellaneous Bits

This sequence is used in most following programming models as intermediate step.

Set the required values for:

- IDP_SMODEx bits in the IDP_CTLx register to specify the frame sync format for the serial inputs (left-justified I^2S, or right-justified mode).
- IDP_Pxx_PDAPMASK bits in the IDP_PP_CTL register to specify the input mask, if the PDAP is used.

- IDP_PP_SELECT bits in the IDP_PP_CTL register to specify input from the DAI pins or the DATA pins, if the PDAP is used.
- IDP_PDAP_CLKEDGE bit (bit 29) in the IDP_PP_CTL register to specify if data is latched on the rising or falling clock edge, if the PDAP is used.

Starting Core Interrupt-Driven Transfer

To start a core interrupt-driven data transfer:

1. Clear the IDP_EN bit which automatically clears the FIFO.
2. Keep the SCLK and frame sync inputs of the SIP and PDAP connected to low, by setting the proper values in the SRU registers.
3. Refer to "Setting Miscellaneous Bits" above.
4. Program the SRU registers to establish the proper connection to the SIP and/or PDAP being used. Keep the unused clock and frame sync signals connected to low.
5. Set the desired values for the *N_SET* variable using the IDP_NSET bits in the IDP_CTL0 register.
6. Set the IDP_FIFO_GTN_INT bit (bit 8 of the DAI_IMASK_RE register) to high and set the corresponding bit in the DAI_IMASK_FE register to low to unmask the interrupt. Set bit 8 of the DAI_IMASK_PRI register (IDP_FIFO_GTN_INT) as needed to generate a high priority or low priority core interrupt when the number of words in the FIFO is greater than the value of N set.
7. Enable the PDAP by setting IDP_PDAP_EN (bit 31 in the IDP_PP_CTL register), if required.

8. Enable the IDP by setting the IDP_EN bit (bit 7 in the IDP_CTL0 register) and the IDP_ENx bits in the IDP_CTL1 register.

In older SHARC processors, the IDP starts shifting data before the IDP is enabled. However, the shifted data is latched at the next frame sync edge only if the IDP is enabled. Therefore, whether the first channel received by the IDP is left/right depends on the instant when the IDP is enabled— which may lead to channel swapping.

Additional Notes

When IDPs are used to receive data from external devices, there is a sequence to be followed to enable the IDP ports when configured to receive data in I^2S mode. Failing to follow this sequence can give rise to channel shift or swap.

1. Connect the frame sync internally using the SRU (Signal Routing Unit) to the DAI interrupt.
2. Configure the DAI interrupt (MISCA) for the inactive edge of the frame sync.
3. Wait for the DAI interrupt, and enable the IDP port inside the DAI interrupt service routine.
4. Clear the DAI interrupt by reading the DAI interrupt latch register. This procedure ensures that the IDP ports are enabled at the correct time, avoiding issues like channel shift or swap in the received data.

Starting A Standard DMA Transfer

To start a DMA transfer from the FIFO to memory:

1. Clear the IDP_EN bit which automatically clears the FIFO.
2. While the global IDP_DMA_EN and the IDP_EN bits are cleared (= 0), set the values for the DMA parameter registers that correspond to channels 7–0.
3. Keep the clock and the frame sync input of the serial inputs and/or the PDAP connected to low, by setting proper values in the SRU registers.
4. Refer to "Setting Miscellaneous Bits" on page 12-27 above.
5. Route all of the required inputs to the IDP by writing to the SRU registers
6. Enable the channel's IDP_ENx and IDP_DMA_ENx bit settings.
7. Start the DMA by setting
   - The IDP_PDAP_EN bit (bit 31 in IDP_PP_CTL register if the PDAP is required).
   - The global IDP_DMA_EN bit of the IDP_CTL0 register to enable standard DMA on the selected channel.
   - The global IDP_EN bit (bit 7 in the IDP_CTL0 register).

Starting a Ping-Pong DMA Transfer

To start a ping-pong DMA transfer from the FIFO to memory:

1. Clear the IDP_EN bit which automatically clears the FIFO.
2. While the global IDP_DMA_EN and IDP_EN bits are cleared (=0), set the values for the following DMA parameter registers that correspond to channels 7–0.
3. Keep the clock and the frame sync input of the serial inputs and/or the PDAP connected to LOW, by setting proper values in the SRU registers.
4. Refer to “Setting Miscellaneous Bits” on page 12-27 above.
5. Connect all of the required inputs to the IDP by writing to the SRU registers.
6. Enable the channel’s IDP_ENx, IDP_DMA_ENx and IDP_PINGx bit settings.
7. Start DMA by setting:
    - The IDP_PDAP_EN bit (bit 31 in IDP_PP_CTL register if the PDAP is required).
    - The global IDP_DMA_EN bit of the IDP_CTL0 register to enable the standard DMA of the selected channel.
    - The global IDP_EN bit (bit 7 in the IDP_CTL0 register).

Servicing Interrupts for DMA

The following steps describe how to handle an IDP ISR for DMA.

1. An interrupt is generated and program control jumps to the ISR when the DMA for a channel completes.

2. The program clears the IDP_DMA_EN bit in the IDP_CTL0 register.

   a. To ensure that the DMA of a particular IDP channel is complete, (all data is transferred into internal memory) wait until the IDP_DMAx_STAT bit of that channel becomes zero in the DAI_STAT register. This is required if a high priority DMA (for example a SPORT DMA) is occurring at the same time as the IDP DMA.

   b. As each DMA channel completes, a corresponding bit in either the DAI_IRPTL_L or DAI_IRPTL_H register for each DMA channel is set (IDP_DMAx_INT).

3. The program clears (= 0) the channel's IDP_DMA_ENx bit in the IDP_CTL1 register which has finished.

4. Reprogram the DMA registers for finished DMA channels.

   More than one DMA channel may have completed during this time period. For each, a bit is latched in the DAI_IRPTL_L or DAI_IRPTL_H registers. Ensure that the DMA registers are reprogrammed. If any of the channels are not used, then its clock and frame sync should be held LOW.

5. Read the DAI_IRPTL_L or DAI_IRPTL_H registers to see if more interrupts have been generated.

   - If the value(s) are not zero, repeat step 4.

   - If the value(s) are zero, continue to step 6.

6. Re-enable the IDP_DMA_EN bit in the IDP_CTL0 register (set to 1).

7. Exit the ISR.

If a zero is read in step 5 (no more interrupts are latched), then all of the interrupts needed for that ISR have been serviced. If another DMA completes after step 5 (that is, during steps 6 or 7), as soon as the ISR completes, the ISR is called again because the OR of the latched bits will not be nonzero again. DMAs in process run to completion.

If step 5 is not performed, and a DMA channel expires during step 4, then, when IDP DMA is re-enabled, (step 6) the completed DMA is *not* reprogrammed and its buffer overruns.

This unit is multiplexed with SIP0. The PDAP provides one clock input, one clock hold input and a maximum of 20 parallel data input pins. The positive or negative edge of the clock input is used for data latching. The clock hold input (PDAP_HLD_I) validates a clock edge—if this input is high then clock edge is masked for data latching. It supports four types of data packing mode selected by MODE bits in the IDP_PP_CTL register.

Debug Features

The following sections describe the features available for debugging the IDP.

Status Register Debug

The core may also write to the FIFO. When it does, the audio data word is pushed into the input side of the FIFO (as if it had come from the SRU on the channel encoded in the three LSBs). This can be useful for verifying the operation of the FIFO, the DMA channels, and the status portions of the IDP. The IDP_STAT1 register returns the current state of the read/write index pointers from FIFO.

Buffer Hang Disable

The IDP_BHD bit in IDP_CTL0 is used for buffer hang disable control. When there is no data in the FIFO, reading the IDP_FIFO register causes the core to hang. This condition continues until the FIFO contains valid data.

Setting the IDP_BHD bit (= 1) prevents the core from hanging on reads from an empty IDP_FIFO register. Clearing this bit (= 0) causes the core to hang under the conditions described previously.

If the IDP_BHD bit (bit 4 in the IDP_CTL0 register) is not set, attempts to read more data than is available in the FIFO results in a core hang.

Shadow Interrupt Registers

For more information, see "Debug Features" on page 2-15.

Core FIFO Write

The core may also write to the FIFO. When it does, the audio data word is pushed into the input side of the FIFO (as if it had come from the SRU on the channel encoded in the three LSBs). This can be useful for verifying the operation of the FIFO, the DMA channels, and the status portions of the IDP. The IDP_STAT1 register returns the current state of the read/write index pointers from FIFO. Note that if both the SIP/PDAP and the core try to write to the FIFO, the core has higher priority.

13 ASYNCHRONOUS SAMPLE RATE CONVERTER

Sample rate converters (SRC) are frequently used in digital signal processing audio applications. In the ADSP-214xx processors, the most frequently used sample rate conversions are off-loaded into hardware modules that are dedicated for filter processing and reduce the instruction processing load on the core, freeing it up for other tasks. The specifications for the module are listed in Table 13-1.

Table 13-1. ASRC Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	Yes
SRU DAI Default Routing	No
SRU2 DPI Required	No
SRU2 DPI Default Routing	N/A
Interrupt Control	Yes
Protocol	
Master Capable	No
Slave Capable	Yes
Transmission Simplex	Yes
Transmission Half Duplex	No
Transmission Full Duplex	No

Table 13-1. ASRC Specifications (Cont'd)

Feature	Availability
Access Type	
Data Buffer	No
Core Data Access	N/A
DMA Data Access	N/A
DMA Channels	N/A
DMA Chaining	N/A
Boot Capable	N/A
Local Memory	Yes (RAM, ROM)
Max Clock Operation	$f_{PCLK}/4$

Features

The ASRC for the SHARC processors has the features shown in the list below.

- 4 Asynchronous stereo SRCs operating in slave mode.
- Simple programming model.
- Controllable muting options (hardware, software and automatic).
- Automatically senses input and output sample frequencies.
- Supports left-justified, I^2S, right-justified (16-,18-, 20-, 24-bits), and TDM serial port modes.
- Daisy-chain configuration in TDM modes for input and output ports to create a serial frame.
- Different protocols on input/output port allow format conversions.
- De-emphasis filter for 32, 44.1 and 48 KHz sampling frequencies.

- Up to 192 kHz sample rate input/output continuous sample ratios from 7.5:1 to 1:8.
- Group delay (latency of interpolation filter) is 16 samples.
- SNR from 128 to 140 dB (depending on processor model).
- Matched phase mode available (ADSP-21488 model only) to compensate for group delays.
- Can be used to de-jitter clocks in systems.

Pin Descriptions

The ASRC has two interfaces: an input port and an output port. Table 13-2 describes the six inputs and two outputs for the IP (input port) and OP (output port).

Table 13-2. ASRC Pin Descriptions

ADSP-214xx Internal Node	I/O	Description
ASRC3–0_CLK_IP_I	Input	ASRC input port clock input
ASRC3–0_FS_IP_I	Input	ASRC input port frame sync input
ASRC3–0_DAT_IP_I	Input	ASRC input port data input
ASRC3–0_CLK_OP_I	Input	ASRC output port clock input
ASRC3–0_FS_OP_I	Input	ASRC output port frame sync input
ASRC3–0_TDM_OP_I	Input	ASRC output port TDM daisy chain data input
ASRC3–0_TDM_IP_O	Input	ASRC output port TDM daisy chain data output
ASRC3–0_DAT_OP_O	Output	ASRC output port data output

SRU Programming

The SRU (signal routing unit) needs to be programmed in order to connect the ASRCs to the output pins or any other peripherals. For normal operation, the data, clock, and frame sync signals need to be routed as shown in Table 13-3.

Table 13-3. ASRC DAI/SRU Signal Routing

ASRC Source	DAI Connection	ASRC Destination
	Group A	ASRC3–0_CLK_IP_I ASRC3–0_CLK_OP_I
ASRC3–0_DAT_IP_O ASRC3–0_TDM_OP_O	Group B	ASRC3–0_DAT_IP_I ASRC3–0_TDM_OP_I
	Group C	ASRC3–0_FS_IP_I ASRC3–0_FS_OP_I
ASRC3–0_DAT_OP_O	Group D	

For information on using the SRU, see "Rules for SRU Connections" on page 10-20.

Register Overview

The ASRC uses five registers to configure and operate the ASRC module. For complete register and bit descriptions, see "Asynchronous Sample Rate Converter Registers" on page A-183.

Control Registers (ASRCCTLx). Enable or disable the sample rate converters. They also specify the input and output data format.

Mute Register (ASRCMUTE). Controls the connection of the mute in and mute out signal.

Ratio Registers (ASRCRATx). Return the sample ratio between the input and out data stream and mute information (mute out).

Clocking

The fundamental timing clock of the ASRC module is peripheral clock/4 (PCLK/4) and is operating in slave mode only. The clock to this module may be shut off for power savings. For more information, see "Disabling Peripheral Clocks" on page 23-11.

Functional Description

Figure 13-1 on page 13-6 shows a top level block diagram of the ASRC module and Figure 13-2 on page 13-8 shows architecture details. Conceptually, the sample rate converter interpolates the serial input data at a rate of 220 and samples the interpolated data stream by the output sample rate. In practice, a 64-tap FIR filter with 220 polyphases, a FIFO, a digital servo loop that measures the time difference between the input and output samples within 5 ps, and a digital circuit to track the sample rate ratio are used to perform the interpolation and output sampling.

I/O Ports

The I/O ports provide the interface through which data is transferred asynchronously into and out of the SRC modules. The SRC has a 3-wire interface for the serial input and output ports that supports left-justified, I^2S, and right-justified (16-, 18-, 20-, 24-bit) modes. Additionally, the serial interfaces support TDM mode for daisy-chaining multiple SRCs to form a frame. The serial output data is dithered down to 20, 18, or 16 bits when 20-, 18-, or 16-bit output data is selected.

The SRC converts the data from the serial input port to the sample rate of the serial output port. The sample rate at the serial input port can be asynchronous with respect to the output sample rate of the output serial port.

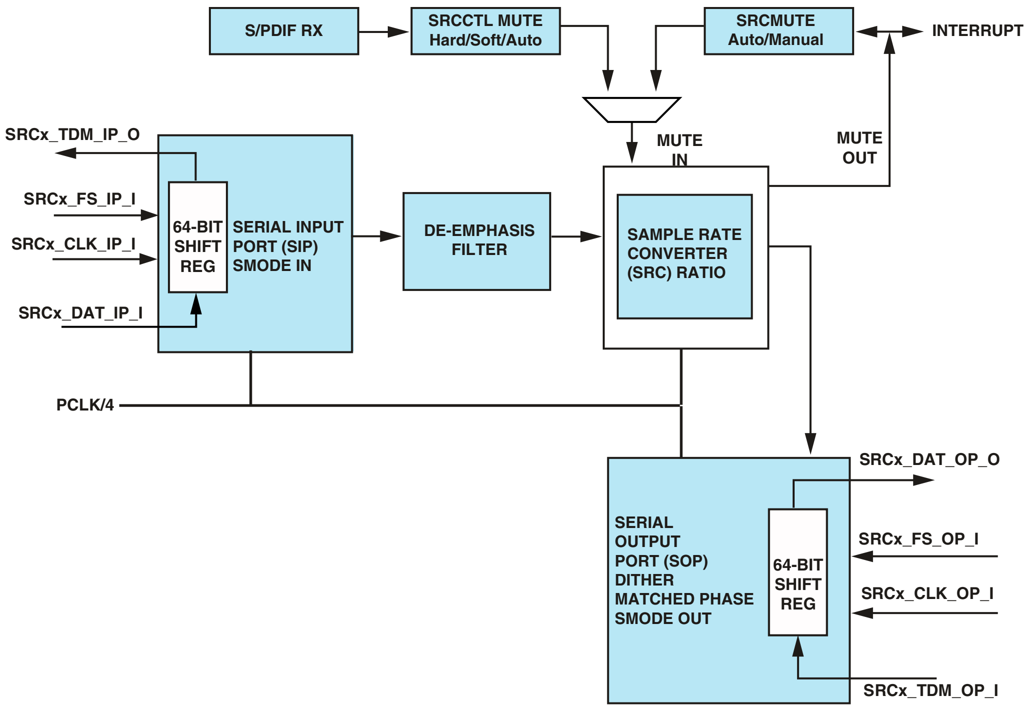

Figure 13-1. Top Level ASRC Block Diagram

De-Emphasis Filter

The de-emphasis filter is used to de-emphasize audio data that has been emphasized.

Mute Control

When either the SRC starts up (or there is a change in sample ratio), the mute out signal (SRCx_MUTEOUT) is asserted (=1). The mute out signal stays high until the SRC settles on the new sample rate(s). While mute out is asserted high, the mute in signal should be asserted high as well. The mute in signal performs a soft mute of the audio input data when asserted and un mutes the input audio data softly when de-asserted.

Note that it takes 4096 input port FS samples until the audio input data is completely muted and 4096 FS samples until the audio input data is completely un muted.

SRC Core

The sample rate converter's RAM FIFO block adjusts the left and right input samples and stores them for the FIR filter's convolution cycle. The ASRCx_FS_IP counter provides the write address (for scaling) to the FIFO block and the ramp input to the digital-servo loop. The ROM stores the coefficients for the FIR filter convolution and performs a high-order interpolation between the stored coefficients. The sample rate ratio block measures the sample rate by dynamically altering the ROM coefficients and scaling the FIR filter length and input data. The digital-servo loop automatically tracks the SRCx_FS_IP and SRCx_FS_OP sample rates and provides the RAM and ROM start addresses for the start of the FIR filter convolution.

Unlike other peripherals, the sample rate converters own local memories (RAM and ROM) which are dedicated for the purpose of sample rate conversion only.
The sample rate converter only operates asynchronously and is always a slave to the input and output ports.

RAM FIFO

The RAM FIFO receives the left and right input data and adjusts the amplitude of the data for both the soft muting of the SRC and the scaling of the input data by the sample rate ratio before storing the samples in RAM. The input data is scaled by the sample rate ratio because as the FIR filter length of the convolution increases, so does the amplitude of the convolution output. To keep the output of the FIR filter from saturating, the input data is scaled down by multiplying it by (SRCx_FS_OP)/(SRCx-_FS_IP) when SRCx_FS_OP<SRCx_FS_IP. The FIFO also scales the input data to mute and stop muting the SRC.

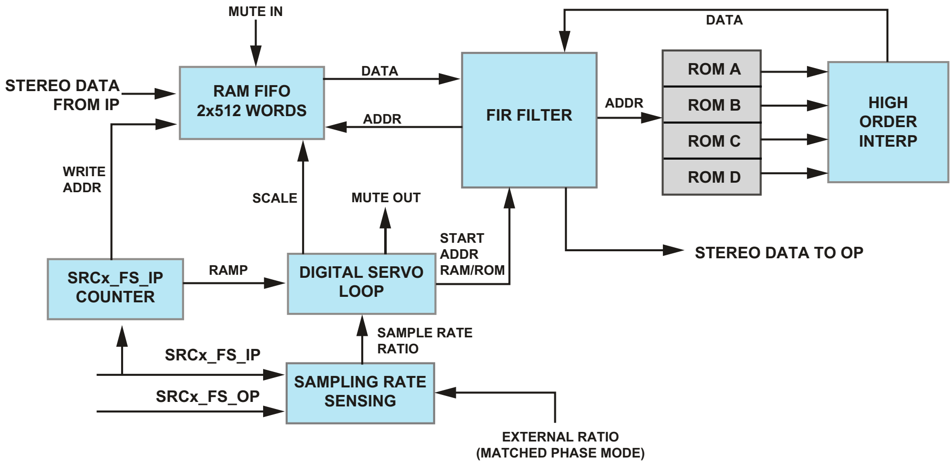

Figure 13-2. Core Architecture

Digital Servo Loop

The digital-servo loop is essentially a ramp filter that provides the initial pointer to the address in RAM and ROM for the start of the FIR convolution. The RAM pointer is the integer output of the ramp filter while the ROM pointer is the fractional part. The digital-servo loop must be able to provide excellent rejection of jitter on the ASRCx_FS_IP and ASRCx_FS_OP clocks as well as measure the arrival of the ASRCx_FS_OP clock within 5 ps.

The digital-servo loop also divides the fractional part of the ramp output by the ratio of (ASRCx_FS_IP)/(ASRCx_FS_OP) for the case when ASRCx-_FS_IP > ASRCx_FS_OP, to dynamically alter the ROM coefficients.

The digital-servo loop is implemented with a multi-rate filter. To settle the digital-servo loop filter quickly at startup or at a change in the sample rate, a fast mode has been added to the filter. When the digital-servo loop starts up or the sample rate is changed, the digital-servo loop kicks into fast mode to adjust and settle on the new sample rate. Upon sensing the digital-servo loop settling down to some reasonable value, the digital-servo loop kicks into normal or slow mode. During fast mode, the ASRCx-_MUTE_OUT bit of the ASRC is asserted to mute the ASRC input which avoids clicks and pops.

FIR Filter

The FIR filter is a 64-tap filter in the case of ASRCx_FS_OP < ASRCx_FS_IP and is (ASRCx_FS_IP)/(ASRCx_FS_OP) × 64 taps for the case when ASRCx-_FS_IP > ASRCx_FS_OP. The FIR filter performs its convolution by loading in the starting address of the RAM address pointer and the ROM address pointer from the digital-servo loop at the start of the ASRCx_FS_OP period. The FIR filter then steps through the RAM by decrementing its address by 1 for each tap, and the ROM pointer increments its address by the (ASRCx_FS_OP/ASRCx_FS_IP) × 2^{20} ratio for ASRCx_FS_IP > ASRCx_FS_OP or 2^{20} for ASRCx_FS_OP < ASRCx_FS_IP. Once the ROM address rolls over, the convolution is complete. The convolution is performed for both the left and right channels, and the multiply/accumulate circuit used for the convolution is shared between the channels.

Sample Rate Sensing

The (SRCx_FS_IP)/(SRCx_FS_OP) sample rate ratio circuit is used to dynamically alter the coefficients in the ROM for the case when SRCx_FS_IP > SRCx_FS_OP. The ratio is calculated by comparing the output of an SRCx-_FS_OP counter to the output of an SRCx_FS_IP counter. If ASRCx_FS_OP >

SRCx_FS_IP, the ratio is held at one. If SRCx_FS_IP > SRCx_FS_OP, the sample rate ratio is updated if it is different by more than two SRCx_FS_OP periods from the previous SRCx_FS_OP to SRCx_FS_IP comparison. This is done to provide some hysteresis to prevent the filter length from oscillating and causing distortion.

Digital Filter Group Delay[1]

The RAM in the FIFO is 512 words deep for both left and right channels. An offset of 16 samples to the write address, provided by the SRCx_FS_IP counter, is added to prevent the RAM read pointer from overlapping the write address. The maximum decimation rate can be calculated from the RAM word is: depth = (512 – 16) ÷ 64 taps = 7.5:1.

The 64 samples effect latency in the interpolation filter. This latency (group delay) depends on interpolation or decimation ratio and is determined as follows:

$$GDL = \frac{16}{f_{S\_IN}} + \frac{32}{f_{S\_IN}} \textit{ seconds for } SRC\_FS\_OP > SRC\_FS\_IP$$

$$GDL = \frac{16}{f_{S\_IN}} + \left(\frac{32}{f_{S\_IN}}\right) \times \left(\frac{f_{S\_IN}}{f_{S\_OUT}}\right) \textit{ seconds for } SRC\_FS\_OP < SRC\_FS\_IP$$

Data Format

Figure 13-3 shows the data input format for a frame (stereo data). The frame format is valid for all protocols. For models which do not support matched phase mode the 8-bit data field is ignored.

[1] Intuitively, the time interval required for a full-level input pulse to appear at the ASRC's output, at full level, is typically expressed in milliseconds (ms). More precisely, the derivative of the radian phase with respect to the radian frequency at a given frequency.

AUDIO DATA LEFT CHANNEL, 24 BITS	MATCHED-PHASE DATA, 8 BITS	AUDIO DATA RIGHT CHANNEL, 24 BITS	MATCHED-PHASE DATA, 8 BITS

MSB (BIT 63) Left-Justified, I^2S, and TDM Mode LSB (BIT 0)

MATCHED-PHASE DATA, 8 BITS	AUDIO DATA LEFT CHANNEL, 16 BITS - 24 BITS	MATCHED-PHASE DATA, 8 BITS	AUDIO DATA RIGHT CHANNEL, 16 BITS - 24 BITS

Right-Justified Mode

Figure 13-3. ASRC Data Frame Format by Protocol

Operating Modes

The ASRC can operate in TDM, I^2S, left-justified, right-justified, and bypass modes. The serial ports of the processor can be used for moving the ASRC data to/from the internal memory.

In I^2S, left-justified and right-justified modes, the ASRCs operate individually. The serial data provided in the input port is converted to the sample rate of the output port.

TDM Input Mode

In TDM input port, several ASRCs can be daisy-chained together and connected to the serial input port of a SHARC processor or other processor (Figure 13-4). The ASRC IP contains a 64-bit parallel load shift register. When the `SRCx_FS_IP_I` pulse arrives, each ASRC parallel loads its left and right data into the 64-bit shift register. The input to the shift register is connected to `SRCx_DATA_IP_I`, while the output is connected to `SRCx_TDM_IP_O`. By connecting the `SRCx_TDM_IP_O` to the `SRCx_DATA_IP_I` of the next ASRC, a large shift register is created, which is clocked by `ASRCx_CLK_IP_I`.

The number of ASRCs that can be daisy-chained together is limited by the maximum frequency of `SRCx_CLK_xx_I`, refer to the data sheet for exact values. For example, if the maximum frequency of

SRCx_CLK_xx_I is *x* MHz, and the output sample rate is *fS*, then number of ASRCs (n) that can be connected in daisy chained fashion is: n × 64 × FS <= *x* MHz.

TDM Output Mode

In TDM output port, several ASRCs can be daisy-chained together and connected to the SPORT of an ADSP-214xx or other processor (Figure 13-4). The ASRC OP contains a 64-bit parallel load shift register. When the ASRCx_FS_OP_I pulse arrives, each ASRC loads its left and right data into the 64-bit shift register. The input to the shift register is connected to ASRCx_TDM_OP_I, and the output is connected to SRCx_DAT_OP_O. By connecting the SRCx_DAT_OP_O to the ASRCx_TDM_OP_I of the next ASRC, a large shift register is created, which is clocked by SRCx_CLK_OP_I.

As shown in Figure 13-4, with three ASRCs in a daisy-chain connection, the serial clock for input/or output port is defined as:
SCLK = 3 × 64 × FS = 192 × FS

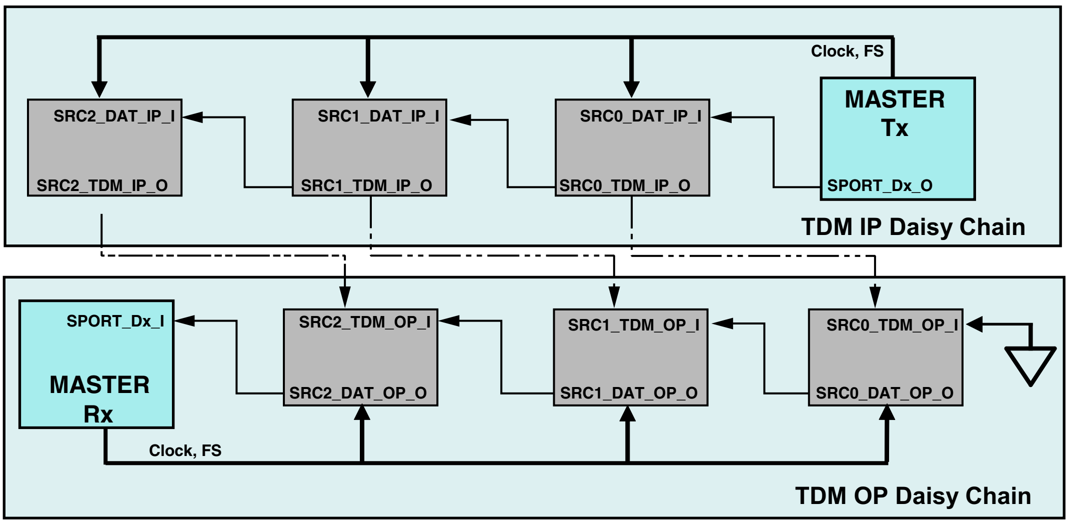

Figure 13-4. TDM Input/Output Modes

Matched-Phase Mode (ADSP-21488)

The matched phase mode of the sample rate converter, shown in Figure 13-5, is enabled by the SRCx_MPHASE bit. This mode is used to match the phase (group delay) between two or more adjacent sample rate converters that are operating with the same input and output clocks.

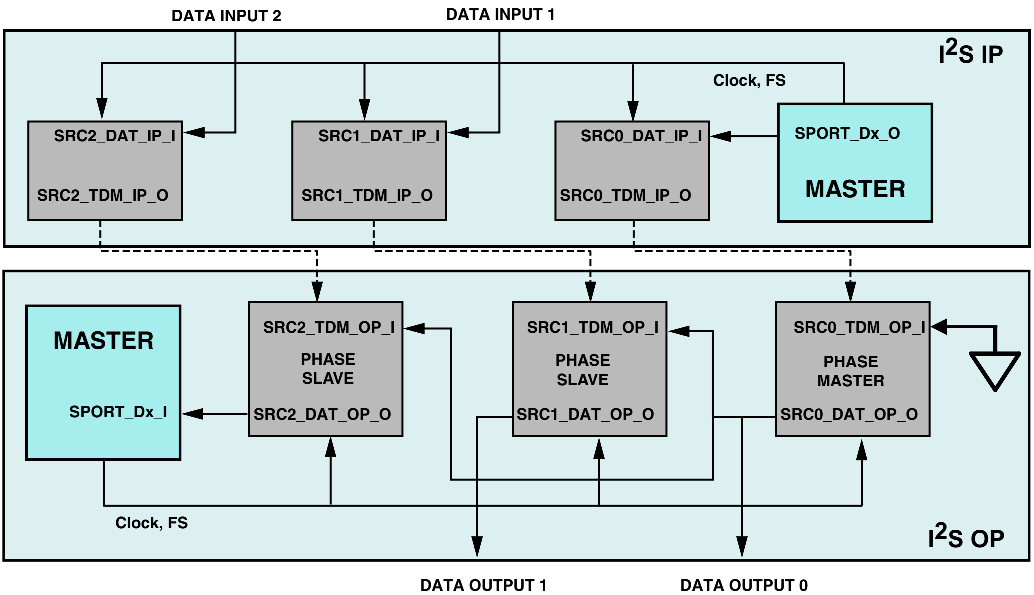

Figure 13-5. Typical Configuration for Matched-Phase Mode Operation

Hysteresis of the (SRCx_FS_OP)/(SRCx_FS_IP) ratio circuit can cause phase mismatching between two ASRCs operating with the same input and output clocks. Since the hysteresis requires a difference of more than two ASRCx_FS_OP periods to update the SRCx_FS_OP and SRCx_FS_IP ratios, two ASRCs may have differences in their ratios from 0 to 4 SRCx_FS_OP period counts. The (SRCx_FS_OP)/(SRCx_FS_IP) ratio adjusts the filter length of the ASRC, which corresponds directly with the group delay. Thus, the magnitude in the phase difference depends upon the resolution of the

ASRCx_FS_OP and ASRCx_FS_IP counters. The greater the resolution of the counters, the smaller the phase difference error.

When the slave SRC SRCx_MPHASE bit is set (=1), it accepts the sample rate ratio transmitted by another SRC, (the matched phase master) which has its SRCx_MPHASE bit cleared (=0), through its serial output.

The phase master ASRC device transmits its SRCx_FS_OP/SRCx_FS_IP ratio through the data output pin (SRCx_DAT_OP_O) to the slave's ASRC's data input pins (SRCx_TDM_OP_I). The transmitted data (32-bit subframe) contains 24-bit data and 8-bits matched phase (see Figure 13-3 on page 13-11).

The slave SRCs receive the 8-bit matched phase bits (instead of their own internally-derived ratio) if their SRCx_MPHASE bits set to 1, respectively. The SRCx_FS_IP and SRCx_FS_OP signals may be asynchronous with respect to each other in this mode. Note that there must be 64 SRCx_CLK_OP cycles per frame in matched-phase mode (2 × 24-bits data and 2 × 8-bits phase match).

> By default, the ADSP-21488 sends matched phased data on its SRCx_DAT_OP_O pin, but only if the SRCx_TDM_OP_I pin is tied low. The slaves simply ignore the matched phased data if their SRCx_MPHASE bits are cleared (= 0).

Bypass Mode

When the BYPASS bit is set (=1), the input data bypasses the sample rate converter and is sent directly to the serial output port. Dithering is disabled. This mode is ideal when the input and output sample rates are the same and ASRCx_FS_IP_I and ASRCx_FS_OP_I are synchronous with respect to each other. In matched phase bypass mode, the ASRCx_FS_OP_I should come at least one SRCx_CLK_xx_I period before ASRCx_FS_IP_I. Cases where this is not met could result in data loss. For example, if internal SPORTS are used then ASRCx_FS_OP_I and ASRCx_FS_IP_I could be driven by different SPORTS so that the timing of these signals could be controlled by enabling them at different times. This mode can also be used for passing through non-audio data since no processing is performed on the input data.

De-Emphasis Mode

The DEEMPHASIS bits choose the type of de-emphasis filter based on the input sample rate for 32, 44.1 or 48 kHz sampling rates.

Dithering Mode[1]

Serial output data is dithered down to 20, 18, or 16 bits when 20-, 18-, or 16-bit output data is selected. In the case of 20, 18 and 16 bit word lengths, the least significant bits of the 24-bit word coming from the SRC into the serial output port are truncated. The DITHER_EN signal (not user configurable) automatically adds dithering to the 24-bit word before truncating to the appropriate output word length. The 21BIT_DITHER signal is used for the consumer version of the SRC to reduce the dynamic range performance to approximately 128 dB.

Muting Modes

The mute feature of the ASRC can be controlled automatically in hardware using the MUTE_IN signal by connecting it to the MUTE_OUT signal. Automatic muting can be disabled by setting (=1) the ASRCx_MUTE_EN bits in the ASRCMUTE register.

Note that by default, the ASRCMUTE register connects the MUTE_IN signal to the MUTE_OUT signal, but not vice versa.

[1] The ASRC can be programmed to add triangular Probability Distribution Function (PDF) dither to the digital audio samples. It is advisable to add dither when the input word width exceeds the output word width, for example the input word is 20 bits and the output word is 16 bits. Triangular PDF is generally considered to create the most favorable noise shaping of the residual quantization noise.

Soft Mute

When the ASRCx_SOFTMUTE bit in the ASRCCTL register is set, the MUTE_IN signal is asserted, and the ASRC performs a soft mute by linearly decreasing the input data to the ASRC FIFO to zero, (–144 dB) attenuation as described for automatic hardware muting.

A 12-bit counter, clocked by ASRCx_FS_IP_I, is used to control the mute attenuation. Therefore, the time it takes from the assertion of MUTE_IN to –144 dB, full mute attenuation is 4096 FS cycles.

Likewise, the time it takes to reach 0 dB mute attenuation from the deassertion of MUTE_IN is 4096 FS cycles.

Hard Mute

When the ASRCx_HARD_MUTE bit in the ASRCCTL register is set, the ASRC immediately mutes the input data to the ASRC FIFO to zero, (–144 dB) attenuation.

Auto Mute

When the ASRCx_AUTO_MUTE bit in the ASRCCTLx register is set, the ASRC communicates with the S/PDIF receiver peripheral to determine when the input should mute. Each ASRC is connected to the S/PDIF receiver to read the DIR_NOAUDIO bits. When the DIR_NOAUDIO bit is set (=1), the ASRC immediately mutes the input data to the ASRC FIFO to zero, (–144 dB) attenuation.

This mode is useful for automatic detection of non-PCM audio data received from the S/PDIF receiver.

Interrupts

Table 13-4 provides an overview of ASRC interrupts

Table 13-4. Overview of ASRC Interrupts

Default Programmable Interrupt	Sources	Masking	Service
DAIHI = P0I DAILI = P12I	ASRC initialization ASRC sample rate change	DAI_IMASK_x	ROC from DAI_IRPTL_x + RTI instruction

Sources

Each ASRC module drives one interrupt signal (mute out asserted). All these signals are connected into the DAI_IRPTL latch register. The S/PDIF ports generate interrupts as described below.

SRC Mute Out

The SRC mute-out signal can be used to generate interrupts on their rising edge, falling edge, or both, depending on how the DAI interrupt mask registers (DAI_IMASK_RE/DAI_IMASK_FE) are programmed. This allows the generation of DAIHI/DAILI interrupts either entering mute, exiting muting or both. The SRCx_MUTE_OUT interrupt is generated only once when the SRC is locked (after 4096 FS input samples) and after changes to the sample ratio. Hard mute, soft mute, and auto mute only control the muting of the input data to the SRC.

Masking

The DAI_IMASK_x register must be unmasked accordingly. The DAIHI and DAILI signals are routed by default to programmable interrupt. To service the DAIHI, unmask (set = 1) the P0I bit in the IMASK register. To service the secondary DAILI, unmask (set = 1) the P12IMSK bit in the LIRPTL register. For DAI system interrupt controllers the DAI_IMASK_RE or DAI_IMASK_FE register must be unmasked. For example:

```
bit set IMASK P0I;            /* unmasks P0I interrupt */
bit set LIRPTL P12IMSK;       /* unmasks P12I interrupt */
ustat1=dm(DAI_IMASK_RE);      /* set SRC0 INT on RE */
bit set ustat1 SRC0_MUTE_INT;
dm(DAI_IMASK_RE)=ustat1;
```

Service

The ISR reads the DAI_IRPTL_x register to clear the interrupt request.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

ASRC Effect Latency

After the ASRC registers are configured the effect latency is 1.5 PCLK cycles minimum and 3 PCLK cycles maximum.

Programming Model

The following is basic information on programming the ASRC module.

1. Program the SRCTLx register and keep the SRCx_ENABLE bit cleared.
2. Set the SRCx_ENABLE bit in SRCCTLx register. After 4096 input port FS cycles the ASRC has un muted.

Debug Features

The asynchronous sample rate converter allow the bypass mode. When the BYPASS bit is set (=1), the input data bypasses the sample rate converter and is sent directly to the serial output port. This mode can be used for testing both ports when the input and output sample rates are at the same frequency, therefore both in- and output ports can be routed to the same serial clock and frame sync.

Shadow Interrupt Registers

For more information, see "Debug Features" on page 2-15.

14 SONY/PHILIPS DIGITAL INTERFACE

The Sony/Philips Digital Interface (S/PDIF) is a standard audio data transfer format that allows the transfer of digital audio signals from one device to another without having to convert them to an analog signal. The digital audio interface carries three types of information; audio data, non audio data (compressed data) and timing information. Its specifications are listed in Table 14-1.

Table 14-1. S/PDIF Specifications

Feature	Transmitter	Receiver
Connectivity		
Multiplexed Pinout	No	No
SRU DAI Required	Yes	Yes
SRU DAI Default Routing	No	No
SRU2 DPI Required	No	No
SRU2 DPI Default Routing	N/A	N/A
Interrupt Control	Yes	Yes
Protocol		
Master Capable	No	Yes
Slave Capable	Yes	No
Transmission Simplex	Yes	Yes
Transmission Half Duplex	No	No
Transmission Full Duplex	No	No

Table 14-1. S/PDIF Specifications (Cont'd)

Feature	Transmitter	Receiver
Access Type		
Data Buffer	No	No
Core Data Access	N/A	N/A
DMA Data Access	N/A	N/A
DMA Channels	N/A	N/A
DMA Chaining	N/A	N/A
Boot Capable	N/A	N/A
Local Memory	No	No
Max Clock Operation	$f_{PCLK}/4$	$f_{PCLK}/4$

Features

The S/PDIF interface has the following features.

- Supports one stereo channel or compressed audio streams.
- AES3-compliant S/PDIF transmitter and receiver.
- Transmitting a biphase mark encoded signal that may contain any number of audio channels (compressed or linear pulse code modulation) or non-audio data.
- S/PDIF receiver managing clock recovery with separate S/PDIF on-chip PLL.
- S/PDIF receiver direct supports DTS frames of 256, 512, 1024, 2048, and 4096 (4096 frames are not supported for the ADSP-2146x processors).
- Manage user status information and provide error-handling capabilities in both the transmitter and receiver.

- DAI allows interactions over DAI by serial ports, IDP and/or the external DAI pins to interface to other S/PDIF devices. This includes using the receiver to decode incoming biphase encoded audio streams and passing them via the SPORTs to internal memory for processing-or using the transmitter to encode audio or digital data and transfer it to another S/PDIF receiver in the audio system.

It is important to be familiar with serial digital audio interface standards IEC-60958, EIAJ CP-340, AES3 and AES11.

Pin Descriptions

Table 14-2 provides descriptions of the pins used for the S/PDIF transmitter.

Table 14-2. S/PDIF Transmitter Pin Descriptions

Internal Node	I/O	Description
DIT_CLK_I	Input	**Serial clock.** Controls the rate at which serial data enters the S/PDIF module (64 × FS).
DIT_DAT_I	Input	**Serial Data.** The format of the serial data can be I^2S, and right- or left-justified.
DIT_FS_I	Input	**Serial Frame Sync.**
DIT_HFCLK_I	Input	**Input sampling clock.** The over sampling clock (which is divided down according to the FREQMULT bit in the transmitter control register to generate the biphase clock)
DIT_EXTSYNC_I	Input	**External Synchronization.** Used for synchronizing the frame counter. If External synchronization is enabled (bit 15 of DITCTL is set), Frame counter resets at rising edge of LRCLK next to the rising edge of EXT_SYNC_I.

Table 14-2. S/PDIF Transmitter Pin Descriptions (Cont'd)

Internal Node	I/O	Description
DIT_O	Output	**Transmit Biphase Mark Encoded Data Stream (AES3 format).**
DIT_BLKSTART_O	Output	**Transmit Block Start.** Indicates the last frame of the current block. This is high for the entire duration of the last frame. This can also be connected to the DAI interrupts 31–22 using SRU_MISCx registers.

Table 14-3 provides descriptions of the pins used for the S/PDIF receiver.

Table 14-3. S/PDIF Receiver Pin Descriptions

Internal Node	I/O	Description
DIR_I	Input	Biphase mark encoded data receiver input stream.
DIR_CLK_O	Output	Extracted receiver sample clock output. This clock is 64 × DIR_FS_O.
DIR_TDMCLK_O	Output	Extracted receiver TDM clock out. This clock is 256 × DIR_FS_O.
DIR_FS_O	Output	Extracted receiver frame sync out.
DIR_DAT_O	Output	Extracted audio data output.

SRU Programming

The SRU (signal routing unit) is used to connect the S/PDIF transmitter biphase data out to the output pins or to the S/PDIF receiver. The serial clock, frame sync, data, and `EXT_SYNC` (if external synchronization is required) inputs also need to be routed through SRU (see Table 14-4).

Table 14-4. S/PDIF DAI/SRU Signal Connections

S/PDIF TX Source	DAI Connection	S/PDIF TX Destination
	Group A	DIT_CLK_I DIT_HFCLK_I DIT_EXTSYNC_I
DIT_O	Group B	DIT_DAT_I
	Group C	DIT_FS_I
DIT_O DIT_BLKSTART_O	Group D	
DIT_BLKSTART_O	Group E	

The SRU (signal routing unit) needs to be programmed in order to connect the S/PDIF receiver to the output pins or any other peripherals and also for the connection to the input biphase stream.

Program the corresponding SRU registers to connect the outputs to the required destinations (Table 14-5). The biphase encoded data and the external PLL clock inputs to the receiver are routed through the signal routing unit (SRU). The extracted clock, frame sync, and data are also routed through the SRU.

Table 14-5. S/PDIF SRU Receiver Signal Connections

S/PDIF RX Source	DAI Connection	S/PDIF RX Destination
DIR_CLK_O DIR_TDMCLK_O	Group A	
DIR_DAT_O	Group B	DIR_I
DIR_FS_O	Group C	
DIR_CLK_O DIR_TDMCLK_O DIR_DAT_O DIR_FS_O	Group D	

Register Overview

This section provides brief descriptions of the major registers. For complete information see "Sony/Philips Digital Interface Registers" on page A-197.

Transmit Control Register (DITCTL). Contains control parameters for the S/PDIF transmitter. The control parameters include transmitter enable, mute information, over sampling clock division ratio, SCDF mode select and enable, serial data input format select and validity and channel status buffer selects.

Transmit Channel Status Registers (DITCHANAx/Bx). Provide status bit information for transmitter subframe A and B in standalone mode.

Transmit User Bit Registers (DITUSRBITAx/Bx). Provide user bit information for transmitter subframe A and B in standalone mode.

Receive Control Register (DIRCTL). Contains control parameters for the S/PDIF receiver. The control parameters include mute information, error controls, SCDF mode select and enable, and S/PDIF PLL disable.

Receive Status Register (DIRSTAT). The receiver also detects errors in the S/PDIF stream. These error bits are stored in the status register, which can be read by the core. Optionally, an interrupt may be generated to notify the core on error conditions.

Receive Channel Status Registers (DIRCHANAx/Bx). Provide status information for receiver subframe A and B.

Clocking

The fundamental timing clock of the S/PDIF is peripheral clock/4 (PCLK/4). The clock to this module may be shut off for power savings.

S/PDIF Transmitter

The following sections provide information on the S/PDIF transmitter.

Functional Description

The S/PDIF transmitter, shown in Figure 14-1, resides within the DAI, and its inputs and outputs can be routed via the SRU. It receives audio data in serial format, encloses the specified user status information, and converts it into the biphase encoded signal. The serial data input to the transmitter can be formatted as left-justified, I^2S, or right-justified with word widths of 16, 18, 20 or 24 bits. Figure 14-2 shows the detail of the AES block.

The serial data, clock, and frame sync inputs to the S/PDIF transmitter are routed through the signal routing unit (SRU).

The S/PDIF transmitter output may be routed to an output pin via the SRU and then routed to another S/PDIF receiver or to components for off-board connections to other S/PDIF receivers. The output is also available to the S/PDIF receiver for loop-back testing through SRU.

In addition to encoding the audio data in the bi-phase format, the transmitter also provides a way to easily add the channel status information to the outgoing bi-phase stream. There are status/user registers for a frame (192-bits/24 bytes) in the transmitter that correspond to each channel or subframe. For more information, see "Transmitter Registers" on page A-197.

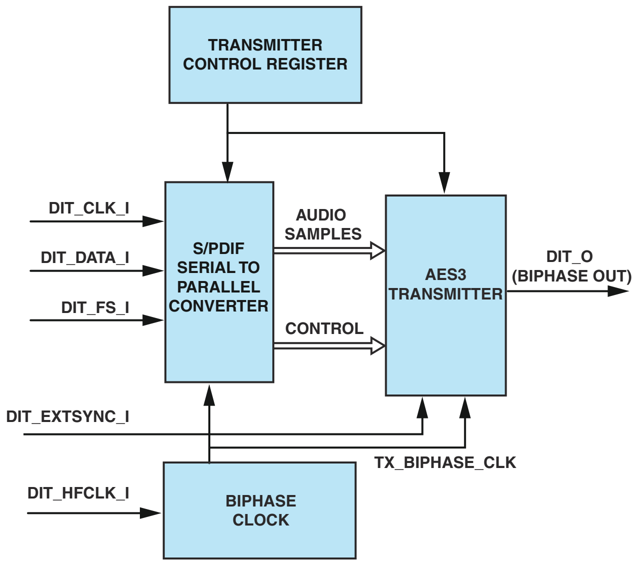

Figure 14-1. S/PDIF Transmitter Block Diagram

Validity bits for both channels may also be controlled by the transmitter control register. Optionally, the user bit, validity bit, and channel status bit are sent to the transmitter with each left/right sample. For each subframe the parity bit is automatically generated and inserted into the bi-phase encoded data.

A mute control and support for double-frequency single-channel mode are also provided. The serial data input format may be selected as left-justified, I^{2}S, or right-justified with 16-, 18-, 20- or 24-bit word widths. The over sampling clock is also selected by the transmitter control register.

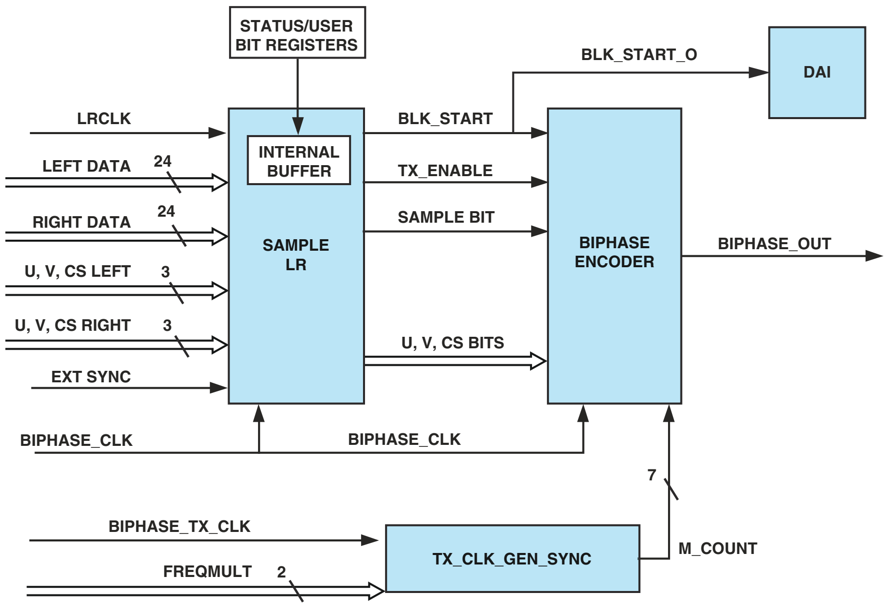

Figure 14-2. AES3 Output Block

Input Data Formats

The Figure 14-3 and Figure 14-4 show the format of data that is sent to the S/PDIF transmitter using a variety of protocol standards.

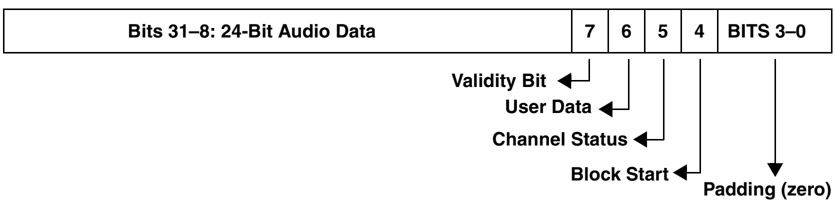

Figure 14-3. I^{2}S and Left-Justified Formats

When I²S format is used with 20-bit or 16-bit data, the audio data should be placed from the MSB of the 24-bit audio data.

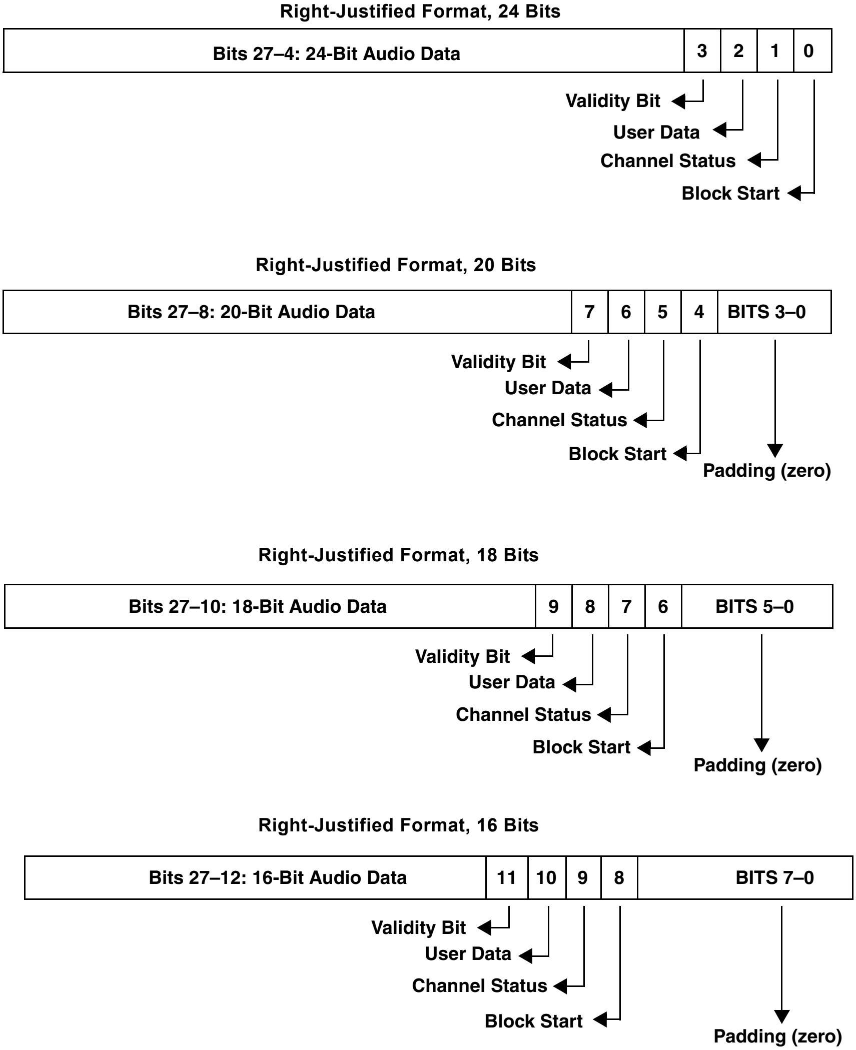

Figure 14-4. Right-Justified Formats

Operating Modes

The S/PDIF transmitter can operate in standalone and full serial modes. The following sections describe these modes in detail.

Full Serial Mode

This mode is selected by clearing bit 9 in the DITCTL register. In this mode all the status bits, audio data and the block start bit (indicating start of a frame), come through the serial data stream (DIT_DATA_I) pin. The transmitter should be enabled after or at the same time as all of the other control bits.

Standalone Mode

This mode is selected by setting bit 9 in the DITCTL register. In this mode, the block start bit (indicating the start of a frame) is generated internally. The channel status bits come from the channel status buffer registers (DITCHANAx and DITCHANBx). The user status bits come from the user bits buffers (DITUSRBITAx and DITUSRBITBx) as shown in Figure 14-2 on page 14-9.

The channel status buffer must be programmed before the S/PDIF transmitter is enabled and used for all the successive blocks of data.

The validity bit for channel A and B are taken from bit 10 and bit 11 of the DITCTL register. In this mode only audio data comes from the DIT_DATA_I pin. All other data, including the status bit and block start bit is either generated internally or taken from the internal register.

Once the user bits buffer registers (DITUSRBITA0-5 and DITUSRBITB0-5) are programmed, they are used only for the next block of data. This allows programs to change the user bit information in every block of data.

To allow user bit updates, write a 0x1 to the DIT_USRUPD register that is used for further processing. If the DIT_AUTO bit in the DITCTL register is set:

- and if DITUSRUPD = 1, at every 192nd frame end, the user status bits are taken from user bits buffers and transmitted. Simultaneously, the DIT_USRUPD register is cleared automatically by hardware.
- and if DITUSRUPD = 0, at every 192nd frame end, then the user status bits are updated as zeros and transmitted. The DIT_USRUPD register remains low.

For the first block of transfer, write a one (1) to the DITUSRUPD register and then enable the S/PDIF transmitter.

In general, for the next block, programs can update user bits buffers at any time during the transfer of the current block (1 block = 192 frames). There are internal buffers to store the user status bits of the current block of transfer. In other words, at the beginning of every new block, the user status bit (DIT_USRPEND in the DITCTL register) from user bits buffers are copied to internal buffers and transmitted in each frame during the transfer.

Note that since a frame contains 192 bits/8 = 24 bytes, six status/user registers are required to store each four bytes.

Data Output Mode

Two output data formats are supported by the transmitter; *two channel mode* and *single-channel double-frequency* (SCDF) mode. The output format is determined by the transmitter control register (DITCTL).

In two channel mode, the left channel (channel A) is transmitted when the DIT_FS_I is high and the right channel (channel B) is transmitted when the DIT_FS_I is low.

In SCDF mode, the transmitter sends successive audio samples of the same signal across both sub frames, instead of channel A and B. The transmitter will transmit at half the sample rate of the input bit stream. The DIT_SCDF bit (bit 4 in the DITCTL register) selects SCDF mode. When in SCDF mode, the DIT_SCDF_LR bit (bit 5 in the DITCTL register) register decides whether left or right channel data is transmitted.

S/PDIF Receiver

The S/PDIF receiver (Figure 14-5) is compliant with all common serial digital audio interface standards including IEC-60958, IEC-61937, AES3, and AES11. These standards define a group of protocols that are commonly associated with the S/PDIF interface standard defined by AES3, which was developed and is maintained by the Audio Engineering Society. The AES3 standard effectively defines the data and status bit structure of an S/PDIF stream. AES3-compliant data is sometimes referred to as AES/EBU compliant. This term highlights the adoption of the AES3 standard by the European Broadcasting Union.

Functional Description

The S/PDIF receiver is enabled at default to receive in two-channel mode. If the receiver is not used, programs should disable the receiver as the digital PLL may produce unwanted switching noise.

If the receiver is not used, programs should disable the digital PLL to avoid unnecessary switching. This is accomplished by writing into the DIR_RESET bit in the DIRCTL register. In most cases, when the S/PDIF receiver is used, this register does not need to be changed. After the SRU programming is complete, write to the DIRCTL register with control values. At this point, the receiver attempts to lock.

For a detailed description of this register, see “Receive Control Register (DIRCTL)” on page A-202.

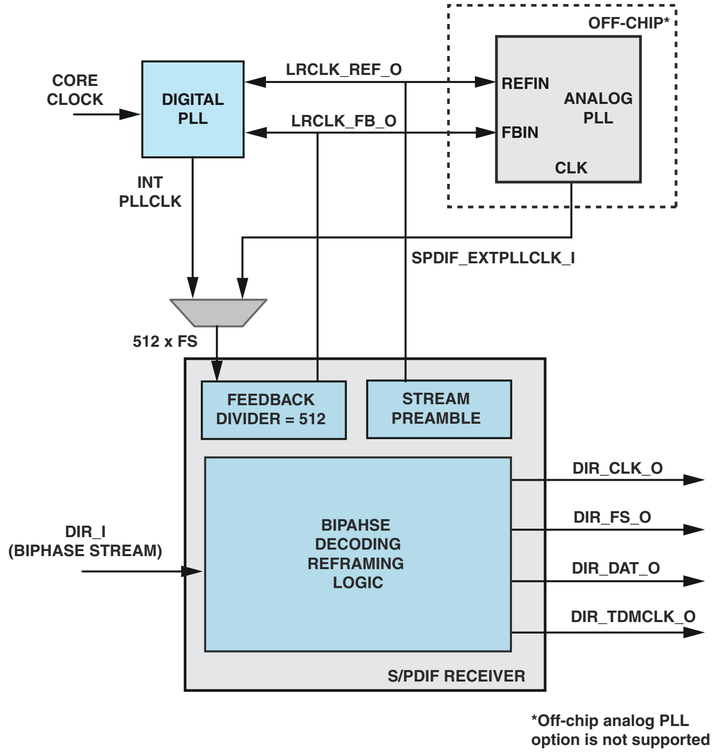

Figure 14-5. S/PDIF Receiver Block Diagram

The input to the receiver (`DIR_I`) is a biphase encoded signal that may contain two audio channels (compressed or linear PCM) or non-audio data. The receiver decodes the single biphase encoded stream, producing an I^2S compatible serial data output that consists of a serial clock, a left-right frame sync, and data (channel A/B). It provides the programmer with several methods of managing the incoming status bit information.

The S/PDIF receiver receives any S/PDIF stream with a sampling frequency range of 32 kHz – 15% to 192 kHz + 15% range.

The channel status bits are collected into memory-mapped registers, while other channel status and user bytes must be handled manually. The block start bit, which replaces the parity bit in the serial I^2S stream, indicates the reception of the Z preamble and the start of a new block of channel status and data bits.

Clock Recovery

The phased-locked loop for the AES3/SPDIF receiver is intended to recover the clock that generated the AES3/SPDIF biphase encoded stream. This clock is used by the receiver to clock in the biphase encoded data stream and also to provide clocks for either the SPORTs, sample rate converter, or the AES3/SPDIF transmitter. The recovered clock may also be used externally to the chip for clocking D/A and A/D converters.

In order to maintain performance, jitter on the clock is sourced to several peripherals. In digital PLL mode (default), after the digital PLL is locked, the outputs from the S/PDIF receiver can have +/–1 core clock cycle jitter in their period. Furthermore, once the PLL achieves lock, it is able to vary ±15% in frequency over time. This allows for applications that do not use PLL unlocking.

To be AES11 compliant, the recovered left/right clock must be aligned with the preambles within a + or – 5% of the frame period. Since the PLL generates a clock 512 times the frame rate clock (512 × FSCLK), this clock can be used and divided down to create the phase aligned jitter-free left/right clock. For more information on recovered clocks, see "Clock Recovery" on page 14-15.

Output Data Format

The extracted 24-bit audio data, V, U, C and block start bits are sent on the DIR_DAT_O pin in 32-bit I^2S format as shown in Figure 14-3. The frame sync is transmitted on the DIR_FS_O pin and serial clock is transmitted on the DIR_CLK_O pin. All three pins are routed through the SRU.

Channel Status

The channel status for the first bytes 4–0 (consumer mode) are collected into memory-mapped registers (DIRCTL and DIRCHANA/DIRCHANB registers). All other channel status bytes 23–5 (professional mode) must be manually extracted from the receiver data stream.

Only the first 5 channel status bytes (40-bit) for consumer mode of a frame are stored into the S/PDIF receiver status registers.

Operating Modes

This section describes the receiver channel status for the different modes.

Compressed or Non-linear Audio Data

The S/PDIF receiver processes compressed as well as non-linear audio data according to the supported standards. The following sections describe how this peripheral handles different data.

The AES3/SPDIF receiver is required to detect compressed or non-linear audio data according to the AES3, IEC60958, and IEC61937 standards. Bit 1 of byte 0 in the DIRSTAT register indicates whether the audio data is linear PCM, (bit 1=0), or non-PCM audio, (bit 1=1). If the channel status indicates non-PCM audio, the DIR_NOAUDIO bit flag is set. (This bit can be used to generate an interrupt.) The DIR_VALID bit (bit 3 in the DIRSTAT register) when set (=1) may indicate non-linear audio data as well. Whenever this bit is set, the VALIDITY bit flag is set in the DIR_RX_STAT register.

MPEG-2, AC-3, DTS, and AAC compressed data may be transmitted without setting either the DIR_VALID bit or bit 1 of byte 0. To detect this data, the IEC61937 and SPMTE 337M standards dictate that there be a 96-bit sync code in the 16-, 20- or 24-bit audio data stream. This sync code consists of four words of zeros followed by a word consisting of 0xF872 and another word consisting of 0x4E1F. When this sync code is detected, the DIR_NOAUDIO bit flag is set. If the sync code is not detected again within 4096 frames, the DIR_NOAUDIO bit flag is deasserted.

The last two words of the sync code, 0xF872 and 0x4E1F, are called the preamble-A and preamble-B of the burst preamble. Preamble-C of the burst preamble contains burst information and is captured and stored by the receiver. Preamble-D of the burst preamble contains the length code and is captured by the receiver. Even if the validity bit or bit 1 of byte 0 has been set, the receiver still looks for the sync code in order to record the preamble-C and D values. Once the sync code has not been detected in 4096 frames, the preamble-C and D registers are set to zero.

The S/PDIF receiver in the ADSP-2147x and ADSP-2148x processors supports DTS frame sizes of 256, 512, 1024, 2048 and 4096. To enable support for 2048 and 4096 DTS frame sizes, set the DTS_CD_4K_EN bit in the DIRCTL register. In the ADSP-2146x processor, the on-chip S/PDIF receiver supports 256, 512 and 1024 DTS frames only. The DTS test kit frames with 2048 and 4096 frame sizes can be detected by adding the sync detection logic in software by using a software counter to check for the DTS header every 2048 and 4096 frames respectively.

Emphasized Audio Data

The receiver must indicate to the program whether the received audio data is emphasized using the channel status bits as detailed below.

- In professional mode, (bit 0 of byte 0 = 1), channel status bits 2–4 of byte 0 indicate the audio data is emphasized if they are equal to 110 or 111.

- In consumer mode, (bit 0 of byte 0 = 0), channel status bits 3–5 indicate the audio data is emphasized if they are equal to 100, 010 or 110.

If emphasis is indicated in the channel status bits, the receiver asserts the EMPHASIS bit flag. This bit flag is used to generate an interrupt.

Single-Channel Double-Frequency Mode

Single-channel, double-frequency mode (SCDF) mode is selected with DIR_SCDF and DIR_SCDF_LR bits in the DIRCTL register. The DIR_BOCHANL/R bits in the DIRSTAT register also contain information about the SCDF mode. When the DIR_BOCHANL/R indicates single channel double frequency mode, the two subframes of a frame carry successive audio samples of the same signal. Bits 0–3 of channel status byte 1 are decoded by the receiver to determine one of the following:

- 0111 = single channel double frequency mode
- 1000 = single channel double frequency mode–stereo left
- 1001 = single channel double frequency mode–stereo right

Clock Recovery Modes

The S/PDIF receiver extracts audio data, channel status, and user bits from the biphase encoded AES3 and S/PDIF stream. In addition, a 50% duty cycle reference clock running at the sampling rate of the audio input data is generated for the PLL in the receiver to recover the oversampling clock.

Digital On-Chip PLL

The receiver can recover the clock from the biphase encoded stream using an on-chip digital PLL shown in Figure 14-5. Note the dedicated on-chip digital PLL is separate from the PLL that supplies the clock to the SHARC processor core and which is the default operation of the receiver.

The left/right frame reference clock for the PLL is generated using the preambles. The recovered low jitter left/right frame clock from the PLL attempts to align with the reference clock. However, this recovered left/right clock, like the reference clock, is not phase aligned with the preambles.

Interrupts

Table 14-6 provides an overview of S/PDIF interrupts.

Table 14-6. Overview of S/PDIF Interrupts

Default Programmable Interrupt	Sources	Masking	Service
DAIHI = P0I DAILI = P12I	Block start	DAI_IMASK_x	ROC from DAI_IRPTL_x + RTI instruction
	Validity No audio Emphasized audio Status change Locked No audio stream CRC error Parity error Biphase error	DAI_IMASK_RE	

Sources

The S/PDIF module drives nine interrupt signals. Eight are status signals driven from SPDIFRX and one signal is driven from SPDIFTX (block start). These signals are connected into the DAI_IRPTL latch register

Transmit Block Start

The DIT_BLKSTART_O output signal, if routed to any miscellaneous interrupt bits (DAI_INT_31-22 in the SRU_MISCx register), triggers a block start interrupt during the last frame of current block.

Receiver Status

The following four receiver status generate an interrupt.

- Validity (DIR_VALID_INT)
- No audio (DIR_NOAUDIO_INT)
- Emphasized audio (DIR_EMPHASIS_INT)
- Status change (DIR_STATCNG_INT)

Note the Status change interrupt is generated if any of the 40 status bits (bytes 4–0) have changed.

Receiver Error

The following four receiver error status bits generate an interrupt.

- Receiver Locked (DIR_LOCK_INT)
- No Audio Stream (DIR_NOSTREAM_INT)
- CRC Error (DIR_CRCERROR_INT)
- Parity or biphase Error (DIR_ERROR_INT)

Notice that parity error and biphase error are ORed together to form a DIR_ERROR_INT interrupt. The CRCERROR bit is not available in the DIRSTAT register. The CRCERROR interrupt latch bit is set whenever the CRC check of the channel status bits fails. The CRC check is only performed if channel status bit 0 of byte 0 is high, indicating professional mode.

Masking

For the S/PDIF receive the DAI_IMASK_RE register must be unmasked accordingly. For the S/PDIF transmit the DAI_IMASK_x register must be unmasked accordingly.

The DAIHI and DAILI signals are routed by default to programmable interrupt.To service the DAIHI, unmask (set = 1) the P0I bit in the IMASK register. To service the secondary DAILI, unmask (set = 1) the P12IMSK bit in the LIRPTL register. For DAI system interrupt controllers the DAI_IMASK_RE or DAI_IMASK_FE register must be unmasked. For example:

```
bit set IMASK P0I;            /* unmasks P0I interrupt */
bit set LIRPTL P12IMSK;       /* unmasks P12I interrupt */
ustat1=dm(DAI_IMASK_RE);      /* set SPDIF RX lock */
bit set ustat1 DIR_LOCK_INT;
dm(DAI_IMASK_RE)=ustat1;
```

Service

The ISR reads the DAI_IRPTL_x register to clear the interrupt request.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

Programming Model

The following sections provide information on programming the transmitter and receiver.

Programming the Transmitter

Since the S/PDIF transmitter data input is not available to the core, programming the transmitter is as simple as: 1) connecting the SRU to the on-chip (serial ports or input data port) or off-chip (DAI pins) serial devices that provide the clock and data to be encoded, and 2) selecting the desired mode in the transmitter control register. This setup can be accomplished in three steps.

1. Connect the transmitter's four required input signals and one biphase encoded output in the SRU. The four input signals are the serial clock (DIT_CLK_I), the serial frame sync (DIT_FS_I), the serial data (DIT_DAT_I), and the high frequency clock (DIT_HFCLK_I) used for the encoding. The only output of the transmitter is DIT_O.

2. If user bits are required, write 0x1 to the DITUSRUPD register for the first block of transfer. Also route the DIT_BLK_START_O signal to the DAI_INT_31-22 (DAI_IRPTLx register). This generates interrupts during the last frame of the block (192), allowing changes of user bits for the next block.

3. Initialize the DITCTL register to enable the data encoding.

4. Manually set the block start bit in the data stream once per block (every 384 words). This is necessary if automatic generation of block start information is not enabled in the DITCTL register, (DIT_AUTO = 0).

Programming the Receiver

Since the S/PDIF receiver data output is not available to the core, programming the peripheral is as simple as connecting the SRU to the on-chip (serial ports or input data port) or off-chip (DAI pins) serial devices that provide the clock and data to be decoded, and selecting the desired mode in the receiver control register. This setup can be accomplished in two steps.

5. Connect the input signal and three output signals in the SRU for. The only input of the receiver is the biphase encoded stream, `DIR_I`. The three required output signals are the serial clock (`DIR_-CLK_O`), the serial frame sync (`DIR_FS_O`), and the serial data (`DIR_DAT_O`). The high frequency clock (`DIR_TDMCLK_O`) derived from the encoded stream is also available if the system requires it.

6. Initialize the `DIRCTL` register to enable the data decoding. Note that this peripheral is enabled by default.

Interrupted Data Streams on the Receiver

When using the S/PDIF receiver with data streams that are likely to be interrupted, (in other words unplugged and reconnected), it is necessary to take some extra steps to ensure that the S/PDIF receiver's digital PLL will re lock to the stream. The steps to accomplish this are described below.

1. Set up interrupts within the DAI so that the S/PDIF receiver can generate an interrupt when the stream is reconnected.

2. Within the interrupt service routine (ISR), stop and restart the digital PLL. This is accomplished by setting and then clearing bit 7 of the S/PDIF receiver control register.

3. Return from the ISR and continue normal operation.

This method of resetting the digital PLL has been shown to provide extremely reliable performance when S/PDIF inputs that are interrupted or unplugged momentarily occur.

The following procedure and the example code show how to reset the digital PLL. Note that all of the S/PDIF receiver interrupts are handled through the DAI interrupt controller.

1. Initialize the No Stream Interrupt

```
/* Enable interrupts (globally) */
BIT SET MODE1 IRPTEN;
/* unmask DAI Hi=Priority Interrupt */
bit set imask DAIHI;
ustat1 = DIR_NOSTREAM_INT;

/* Enable no-stream Interrupt on Falling Edge. Interrupt
occurs when the stream is reconnected */
dm(DAI_IRPTL_FE) = ustat1;

/* Enable Hi-priority DAI interrupt */
dm(DAI_IRPTL_PRI) = ustat1;

/* If more than 1 DAI interrupt is being used, it is neces-
sary to determine which interrupt occurred here */

/* Interrupt Service Routine for the DAI Hi-Priority Inter-
rupt. This ISR triggered when the DIR sets no_stream bit */
_DAIisrH:
```

2. Reset the Digital PLL Inside of the ISR

```
r8=dm(DAI_IRPTL_H);         /* Reading DAI_IRPTL_H
                               clears interrupt */
ustat2=dm(DIRCTL);
 bit set ustat2 DIR_PLLDIS;  /* bit_7 disables digital
                                pll only */
 dm(DIRCTL)=ustat2;
 bit clr ustat2 DIR_PLLDIS;  /*reenable the digital pll */
 dm(DIRCTL)=ustat2;
```

Debug Features

The following feature supports S/PDIF debugging.

Loopback Routing

The S/PDIF supports an internal loopback mode by using the SRU. For more information, see "Loopback Routing" on page 10-40.

Shadow Interrupt Registers

For more information, see "Debug Features" on page 2-15.

15 PRECISION CLOCK GENERATOR

The precision clock generators (PCG) consist of four units, each of which generates a pair of signals (clock and frame sync) derived from a clock input signal. The units, A B, C, and D, are identical in functionality and operate independently of each other. The two signals generated by each unit are normally used as a serial bit clock/frame sync pair. Table 15-1 lists the PCG specifications.

Table 15-1. PCG Specifications

Feature	PCGA–B	PCGC–D
Connectivity		
Multiplexed Pinout	No	No
SRU DAI Required	Yes	Yes
SRU DAI Default Routing	No	No
SRU2 DPI Required	No	Yes
SRU2 DPI Default Routing	No	No
Interrupt Control	No	No
Protocol		
Master Capable	Yes	Yes
Slave Capable	No	No
Transmission Simplex	N/A	N/A
Transmission Half Duplex	N/A	N/A
Transmission Full Duplex	N/A	N/A

Table 15-1. PCG Specifications (Cont'd)

Feature	PCGA–B	PCGC–D
Access Type		
Data Buffer	No	No
Core Data Access	N/A	N/A
DMA Data Access	N/A	N/A
DMA Channels	N/A	N/A
DMA Chaining	N/A	N/A
Boot Capable	N/A	N/A
Local Memory	No	No
Max Clock Operation	f_{PCLK}	f_{PCLK}

Features

The following list describes the features of the precision clock generators.

- Operates on the DAI and DPI units.
- PCG input clock selection from `CLKIN`, `PCLK` or external DAI pins.
- Provides four different clock dividers for serial clock, frame sync, phase (20-bit) and pulse width (16-bit).
- Phase shift allows adjustment of the frame sync relative to the serial clock and can be shifted the full period and wrap around.
- Provides pulse width control for arbitrary frame sync signal generation.

- Bypass mode for external frame sync manipulation.
- External trigger mode starts PCG operation. No additional jitter introduced since operation is independent of the on-chip PLL by using off-chip clocks.

Pin Descriptions

Table 15-2 provides the pin descriptions for the PCGs (x = unit A, B, C, or D).

Table 15-2. PCG Pin Descriptions

Internal Nodes	I/O	Description
Inputs		
CLKIN	I	External clock input for PCG x
PCLK	I	Internal peripheral clock input for PCG x
PCG_SYNC_CLKx_I	I	External trigger used to enable the frame sync output
PCG_EXTx_I	I	External clock A input provided to the PCG x (not CLKIN)
MISCA2_I	I	External frame sync used for bypass mode PCG A
MISCA3_I	I	External frame sync used for bypass mode PCG B
MISCA4_I	I	External frame sync used for bypass mode PCG C
MISCA5_I	I	External frame sync used for bypass mode PCG D
Outputs		
PCG_CLKx_O	O	Serial clock x output
PCG_FSx_O	O	Frame sync x output

SRU Programming

To use the PCG, route the required inputs using the SRU as described Table 15-3. Also, use the SRU to connect the outputs to the desired DAI pin.

Table 15-3. PCG SRU Connections

DAI Source	DAI Group	DPI Group	DAI Destination
PCG_SYNC_CLKA_O PCG_SYNC_CLKB_O	Group A		PCG_SYNC_CLKA_I PCG_SYNC_CLKB_I PCG_SYNC_CLKC_I PCG_SYNC_CLKD_I PCG_EXTA_I PCG_EXTB_I PCG_EXTC_I PCG_EXTD_I
PCG_FSA_O PCG_FSB_O	Group C		
PCG_CLKA_O PCG_CLKB_O PCG_FSA_O PCG_FSB_O PCG_CLKC_O* PCG_CLKD_O* PCG_FSC_O* PCG_FSD_O*	Group D	Group B*	
PCG_CLKB_O PCG_FSA_O PCG_FSB_O	Group E		MISCA2_I MISCA3_I MISCA4_I MISCA5_I

ⓘ A PCG clock output cannot be fed to its own input. Setting `SRU_CLK4[4:0] = 28` connects `PCG_EXTA_I` to logic low, not to `PCG_CLKA_O`. Setting `SRU_CLK4[9:5] = 29` connects `PCG_EXTB_I` to logic low, not to `PCG_CLKB_O`. The clock and frame sync signals of

PCG C and D cannot be directly connected to other peripheral clock and frame sync signals. They can only be routed through the DAI pins.

Register Overview

The processor contains registers that are used to control the PCGs.

- **Control Register 0 (PCG_CTLx0).** Enables the clock and frame sync, it includes the frame sync divider and the upper half of the 20-bit phase value.
- **Control Register 1 (PCG_CTLx1).** Enables the clock and frame sources, it includes the clock divider and the lower half of the 20-bit phase value.
- **Pulse Width Register (PCG_PWx).** Contains the pulse with settings for normal mode (FSDIV > 1) or control bits for bypass mode (FSDIV = 1/0). Enables direct bypass or one shot mode.
- **Synchronization Register (PCG_SYNCx).** Enables PCLK as input clock to the PCGs. It also enables external FS trigger mode.

Clocking

The fundamental clock of the PCG is PCLK. The clock to this module may be shut off for power savings. For more information, see “Disabling Peripheral Clocks” on page 23-11.

Functional Description

The following sections provide information on the function of the precision clock generators.

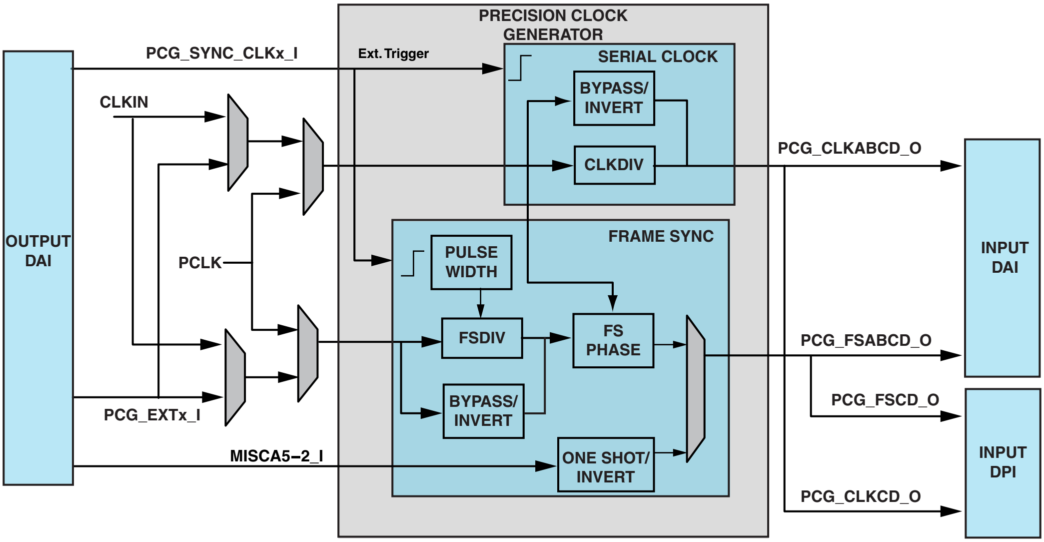

Figure 15-1. PCG Block Diagram

Serial Clock

Each of the four units (A, B, C, and D) produces a clock output. Serial clock generation from a unit is independently enabled and controlled. Sources for the serial clock generation can be either from the CLKIN, PCLK, or a DAI pin source. The clock output is derived from the input to the PCG with a 20-bit divisor.

Note that the divider is working in normal mode for CLKxDIV > 1. For CLKxDIV = 0 or 1 the divider operates in bypass mode, (input clock is fed directly to its output). Note that in bypass mode, the clock at the output can theoretically run at up to the PCLK frequency. However the DAI/DPI pin buffers limit the speed to PCLK/4.

Note that the clock output is always set (as closely as possible) to a 50% duty cycle. If the clock divisor is even, the duty cycle of the clock output is exactly 50%. If the clock divisor is odd, then the duty cycle is slightly less than 50%. The low period of the output clock is one input clock period more than the high period of the output clock. For higher values of an odd divisor, the duty cycle is close to 50%.

A PCG clock output cannot be fed to its own input.

Frame Sync

The following sections describe the use of frame syncs in the PCGs.

Frame Sync Output

Each of the four units (A through D) also produces a synchronization signal for framing serial data. The frame sync outputs are much more flexible since they need to accommodate the wide variety of serial protocols used by peripherals.

Frame sync generation from a unit is independently enabled and controlled. Sources for the frame sync generation can be either from the crystal buffer output, PCLK, or an external pin source. There is only one external source pin for both frame sync and clock output for a unit.

If an external source is selected for both frame sync and clock output for a unit, then they operate on the same input signal. Apart from enable and source select control bits, frame sync generation is controlled by a 20-bit divisor.

Divider Mode Selection

If frame sync divisor > 1 the PCG frame sync output frequency is equal to the input clock frequency, divided by a 20-bit integer. This integer is specified in the FSDIV bit field (bits 19–0 of the PCG_CTLx0 register).

However if the frame sync divisor is zero or one, the PCG's frame sync clock generation unit is bypassed, and the frame sync input is connected directly to the frame sync output. For FSDIV=0, 1 the PCG_PWx registers have different functionality than in normal mode.

Phase Shift

Phase shift is a frame sync parameter that defines the phase shift of the frame sync with respect to the input clock of the same unit. This feature allows shifting of the frame sync signal in time relative to the clock input signal. Frame sync phase shifting is often required by peripherals that need a frame sync signal to lead or lag a clock signal.

For example, the I^2S protocol specifies that the frame sync transition from high to low occur one clock cycle before the beginning of a frame. Since an I^2S frame is 64 clock cycles long, delaying the frame sync by 63 cycles produces the required framing.

Phase shifting is represented as a full 20-bit value so that even when the frame sync is divided by the maximum amount, the phase can be shifted to the full range, from zero to one input clock short of the period.

> Phase shifting is specified as a 2 x 10-bit divider value in the FSx-PHASE_HI bit field (bits 29–20) of the PCG_CTLx0 register and in the FSxPHASE_LO bit field (bits 29–20) of the PCG_CTLx1 register.

A single 20-bit value spans these two bit fields. The upper half of the word (bits 19–10) is in the PCG_CTLx0 register, and the lower half (bits 9–0) is in the PCG_CTLx1 register.

The phase shift between clock and frame sync outputs may be programmed using the PCG_PW and PCG_CTLxx registers under these conditions:

- The input clock source for the clock generator output and the frame sync generator output is the same.
- The clock and frame sync are enabled at the same time using a single atomic instruction.
- The frame sync divisor is an integral multiple of the clock divisor.

When using a clock and frame sync as a synchronous pair, the units must be enabled in a single atomic instruction before their parameters are modified. Both units must also be disabled in a single atomic instruction as shown below.

```
r0 = CLKDIV|PHASE_LO;
dm(PCG_CTLA1) = r0;
r0 = FSDIV|PHASE_HI|ENCLKA|ENFSA;   /* program dividers and
                                       enable CLK and FS */
dm(PCG_CTLA0) = r0;
```

If the phase shift is 0 (see Figure 15-2), the clock and frame sync outputs rise at the same time.
If the phase shift is 1, the frame sync output transitions one input clock period ahead of the clock transition.
If the phase shift is divisor – 1, the frame sync transitions divisor – 1 input clock periods ahead of the clock transitions.

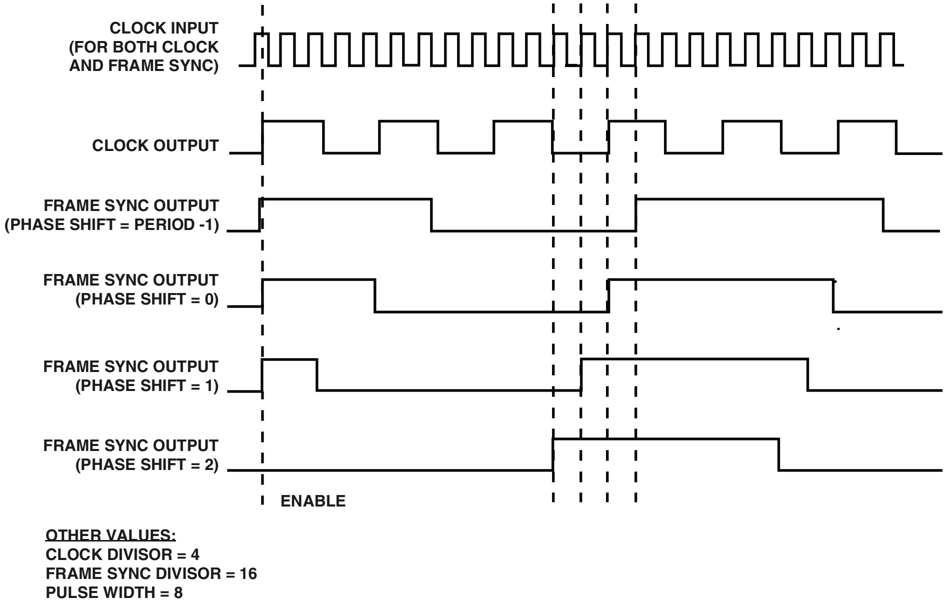

Figure 15-2. Phase and Pulse Width Settings

If generating single frame sync pulses (the length of one `SCLK` cycle) care must be taken with respect to the drive and sampling edges. If the rules are violated, for example if the SPORT is not driving data, it will not able to detect a valid sample edge.

Pulse Width

Pulse width is the number of input clock periods for which the frame sync output is high.

A 16-bit value determines the width of the framing pulse. Settings for pulse width can range from zero to `DIV` – 1. The pulse width should be less than the divisor of the frame sync. The pulse width of frame sync is specified in the `PWFSx` bits (15–0) and (31–16) of the `PCG_PWx` registers.

Default Pulse Width

If the pulse width count is equal to 0 and if FSDIV bit field is even, then the actual pulse width of the frame sync output is equal to:

For even divisors: frame sync divisor/2

If the pulse width count is equal to 0 and if FSDIV bit field is odd, then the actual pulse width of the frame sync output is equal to:

For odd divisors: frame sync divisor – 1/2

Input Clock Source Considerations

The core phase-locked loop (PLL) has been designed to provide clocking for the processor core. Although the performance specifications of this PLL are appropriate for the core, they have not been optimized or specified for precision data converters where jitter directly translates into time quantization errors and distortion.

Therefore the PCG allows the routing of external clock sources which are independent of the core PLL.

Timing Example for I^{2}S Mode

For I^{2}S mode, the frame sync should be driven at the falling edge of SCLK. In other words, the frame sync edge should coincide with the falling edge of the SCLK. To satisfy this requirement, the phase of the frame sync should be programmed accordingly in the PCG_CTLxx registers.

For example, assume that the input clock source for both clock and frame sync are the same and both the clock and frame sync are enabled at the same time. Also assume that the clock divisor value needed to generate the required SCLK is CLKxDIV = 4. Then, for a 32-bit word length, the frame sync divisor value should be FSDIV = 64 × CLKxDIV = 256.

By default, for phase = 0, the rising edge of both SCLK and frame sync will coincide. To make sure that the frame sync edges coincides with the falling edge of the SCLK, the phase value needs to be programmed as CLKxDIV/2 = 2. It can be done by following instructions:

```
ustat1=CLKDIV|((CLKDIV/2) << 20);
dm(PCG_CTLx1) = ustat1;
```

For details on how to program phase of the frame sync see "Programming Model" on page 15-20.

Operating Modes

The following sections provide information on the operating modes of the precision clock generator.

Normal Mode

When the frame sync divisor is set to any value other than zero or one, the PCGs operates in normal mode. In normal mode, the frequency of the frame sync output is determined by the divisor where:

$$\textit{Frequency of Frame Sync Output} = \left(\frac{\textit{Input Frequency}}{\textit{Divisor}} \right)$$

The high period of the frame sync output is controlled by the value of the pulse width control. The value of the pulse width control should be less than the value of the divisor.

The phase of the frame sync output is determined by the value of the phase control. If the phase is zero, then the positive edges of the clock and frame sync coincide when:

- the clock and frame sync dividers are enabled at the same time using an atomic instruction

- the divisors of the clock and frame sync are the same
- the source for the clock and frame sync is the same

The number of input clock cycles that have already elapsed before the frame sync is enabled is equal to the difference between the divisor and the phase values. If the phase is a small fraction of the divisor, then the frame sync appears to lead the clock. If the phase is only slightly less than the frame sync divisor, then the frame sync appears to lag the clock. The frame sync phase should not be greater than the divisor.

Bypass Mode

When the frame sync divisor for the frame sync has a value of zero or one, the frame sync is in bypass mode, and the PCG_PWx registers have different functionality than in normal mode.

In normal mode bits 15–0 and 31–18 of the PCG_PWx registers are used to program the pulse width count. In bypass mode bits 15–2 and 31–18 are ignored. Bits 1–0 and 17–16 are renamed to STROBEx and INFSx respectively. This is described in more detail below.

If the STROBEx bit of PCG_PWx register is cleared, then the input is directly passed (see Figure 15-3) to the frame sync output either inverted or not inverted, depending on the INVFSx bit of the PCG_PWx registers.

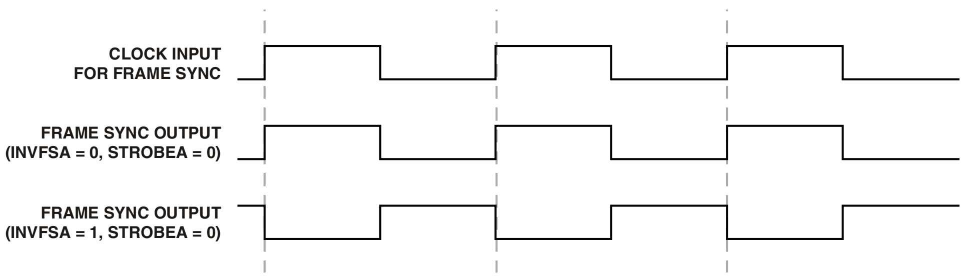

Figure 15-3. Bypass and Inverted Bypass

One-Shot Mode

In one-shot mode operation (see Figure 15-4), the PCG produces a series of periods but does not run continuously.

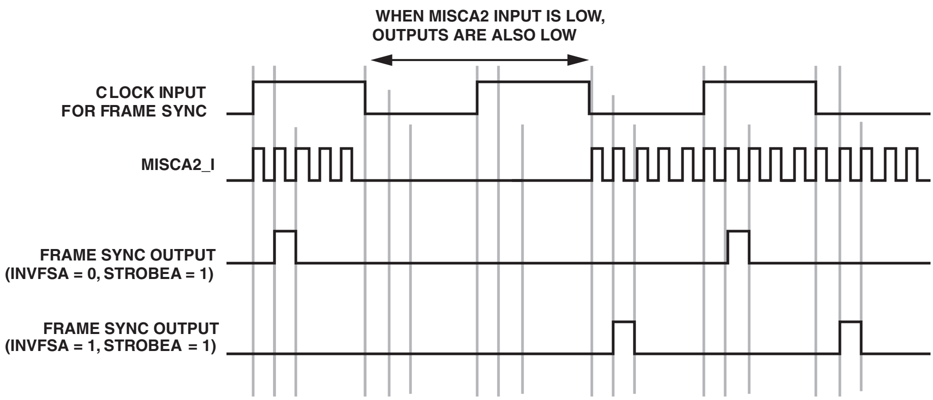

Figure 15-4. One Shot Mode PCG A (MISCA2_I input)

Bypass mode also enables the generation of a strobe pulse (one shot frame sync). Strobe usage ignores the divider counters and looks to the SRU to provide the input signal. Two bit fields determine the operation in this mode.

In the bypass mode, if the STROBEx bit of PCG_PWx register is set to 1, then a one-shot pulse is generated. This one-shot pulse has the duration equal to the period of MISCAx_I for the PCGx unit. This pulse is generated either at the falling or rising edge of the input clock, depending on the value of the INVFSx bit of the PCG_PW register. The output pulse width is equal to the period of the SRU source signal MISCAx_I. The pulse begins at the second rising edge of MISCAx_I following a rising edge of the clock input. When the INVFSx bit is set, the pulse begins at the second rising edge of MISCAx_I coinciding with or following a falling edge of the clock input.

Notice a strobe period is defined to be the period of the FS input clock signal specified by the FSxSOURCE bit (PCG_CTLx1 registers).

External Event Trigger

The trigger with the external clock is enabled by setting bits 0 and 16 of the PCG_SYNC register.

Since the rising edge of the external clock is used to synchronize with the frame sync, the frame sync output is not generated until a rising edge of the external clock is sensed (Figure 15-5).

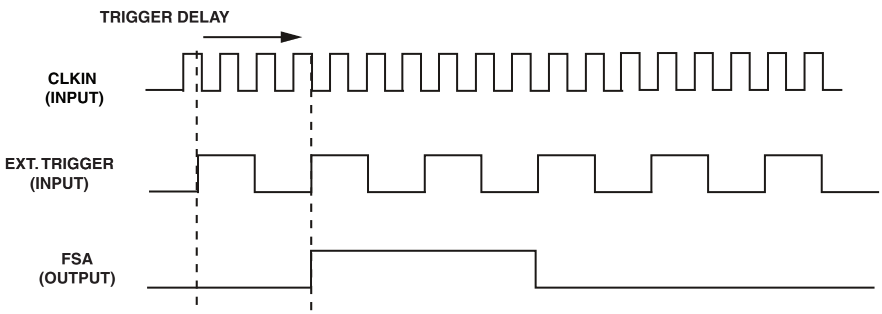

Figure 15-5. FS Output Synchronization With External Trigger Input

External Event Trigger Delay

The time delay between the rising trigger edge and the start of SCLK/FS varies between 2.5 to 3.5 input clock periods. If the input clock and the trigger signal are synchronous, the delay is 3 input clock periods. The following cases need to be considered:

- PCLK is the input source. In this case if the given trigger event is synchronous to PCLK, the delay is 3 PCLK periods. If the trigger signal is asynchronous with PCLK, the delay varies from 2.5 PCLK periods to 3.5 PCLK periods. (It depends on whether the trigger edge occurs in the positive half cycle or negative half cycle of PCLK.)

- CLKIN is the input source. In this case if the given trigger signal is synchronous to CLKIN, the delay is 3 CLKIN periods. But if they are asynchronous to CLKIN, the delay can vary between 2.5 CLKIN periods to 3.5 CLKIN periods.
- SRU is the input source. If the input clock and trigger signal are synchronous, the delay is exactly 3 input clock periods. If asynchronous, it varies between 2.5 to 3.5 input clock periods depending on the phase difference between the input clock and trigger signal.

Audio System Example

Figure 15-6 shows an example of the internal interconnections between the SPDIF receiver, ASRC, and the PCGs. The interconnections are made by programming the signal routing unit.

It shows how to set up two precision clock generators using the S/PDIF receiver and an asynchronous sample rate converter (ASRC) to interface to an external audio DAC. The PCG is configured to provide a fixed ASRC/DAC output sample rate of 65.098 kHz. The input to the S/PDIF receiver is typically 44.1 kHz if supplied by a CD player, but can also be from other source at any nominal sample rate from about 22 kHz to 192 kHz.

Similarly, the phase shift for frame syncs B, C, and D is specified in the corresponding PCG_CTLx0 and PCG_CTLx1 registers.

Three synchronous clocks are required in audio systems

1. Frame sync (FS)
2. Serial bit clock (64 × FS)
3. Master DAC clock (256 × FS)

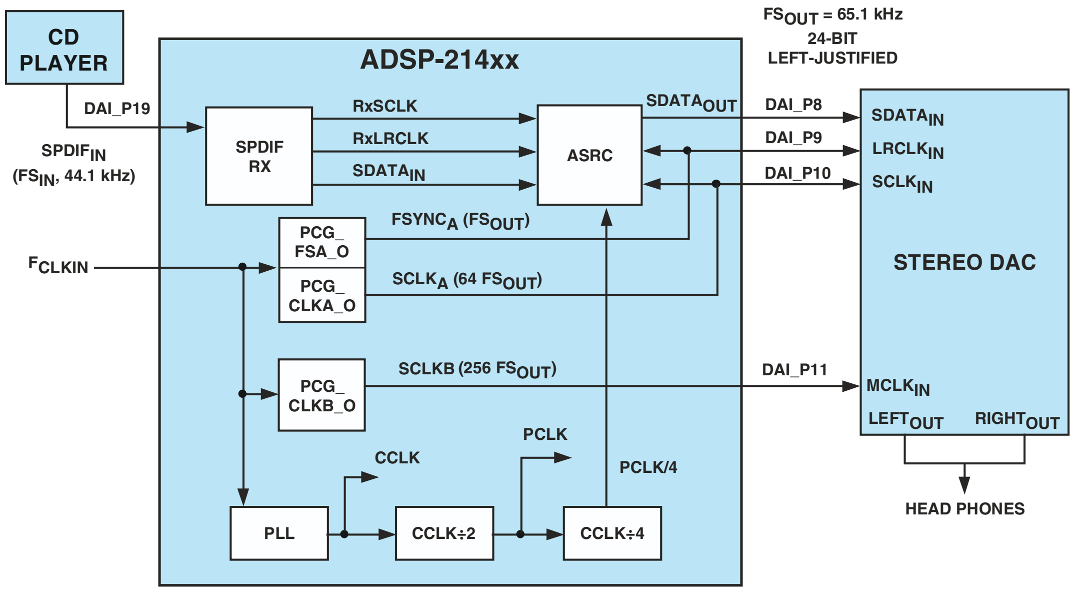

Figure 15-6. PCG Setup for I2S or Left-Justified DAI

Since each PCG has only two outputs, this example requires two PCGs. Furthermore, because the digital audio interface requires a fixed-phase relation between SCLK and FS, these two outputs should come from one PCG (PCG A) while the master clock comes from the 2nd (PCG B).

The combined PCGs can provide a selection of synchronous clock frequencies to support alternate sample rates for the ASRCs and external DACs. However, the range of choices is limited by CLKIN and the ratio of PCG_CLKx_O:SCLK:FS which is normally fixed at 256:64:1 to support digital audio left-justified, I^{2}S and right-justified interface modes.

Many DACs also support 384, 512, and 786x FS for PCG_CLKx_O, which allows some additional flexibility in choosing CLKIN.

Note the falling edge of SCLK must always be synchronous with both edges of FS. This requires that the phase of the SCLK and FS signals for a common PCG (PCG A) be adjustable.

While the frequency of the master DAC clock (PCG_CLKx_O) must be synchronous with the sample rate supplied to the external DAC, there is no fixed phase requirement.

Set the clock divisor and source and low-phase word first, followed by the control register enable bits, which must be set together. When the PCG_PW register is set to zero (default) the FS pulse width is (divisor ÷ 2) for even divisors and (divisor – 1) ÷ 2 for odd divisors. Alternatively, the PCG_PW register could be set high for exactly one-half the period of CLKIN cycles for a 50% duty cycle, provided the FS divisor is an even number.

Clock Configuration Examples

For a CLKIN = 33.330 MHz the two PCGs provide the three synchronous clocks PCGx_CLK, SCLK and FS for the SRCs and external DAC. These divisors are stored in 20-bit fields in the PCG_CTL registers.

The integer divisors for several possible sample rates based on 33.330 MHz CLKIN are shown in Table 15-4.

Table 15-4. Precision Clock Generator Division Ratios (33.330 CLKIN)

Sample Rate kHz)	**PCG Divisors**		
	CLKDIV B	**CLKDIV A**	**FSDIV A[1]**
130.195	1	4	256
65.098	**2**	**8**	**512**
43.398	3	12	768
32.549	4	16	1024
26.039	5	20	1280

Table 15-4. Precision Clock Generator Division Ratios (33.330 CLKIN) (Cont'd)

Sample Rate kHz)	PCG Divisors		
	CLKDIV B	CLKDIV A	FSDIV A[1]
21.699	6	24	1536
18.599	7	28	1792

1 The frame sync divisor should be an even integer in order to produce a 50% duty cycle waveform. See "Frame Sync" on page 15-7.

For more information on core clock setting, see "Power Management Registers (PMCTL, PMCTL1)" on page A-7.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

PCG Effect Latency

After the PCG registers are configured the effect latency is shown below. The latency to start the `CLKOUT` depends on the divisor value and input source as described below.

Input clock through `PCLK`

- If the divisor value is 0 or 1 (bypassed) the latency is 1 `PCLK` cycle
- For other divisor values the latency is 3 `PCLK` cycles

Input clock through `CLKIN`

- If the divisor is 0 or 1 (bypassed) the latency can vary from 0 to 1 oscillator period. This is because clock generation starts with the immediate positive edge of the `CLKIN`.
- For other divisor values the latency can vary between 2 to 3 oscillator periods. This is because clock generation starts with the third positive edge of `CLKIN`.

Input clock through SRU

- If the divisor is 0 or 1 (bypassed) the latency can vary from 0 to 1 input clock period. For example if the input clock has a period of 100 ns then this latency can be a maximum of 100 ns.
- For other divisor values the latency can vary between 2 to 3 input clock periods. For example if the input clock has a period of 100 ns then this latency can be between 200 and 300 ns.

Programming Model

The section describes which sequences of software steps required for successful PCG operation.

If the PCG is being disabled in order to re-program a parameter, please use a delay after writing to the disable bit should be used. This delay in core clock (CCLK) cycles = (PCG source clock period/CCLK period). In summary, the following general procedure should be used.

1. Clear the PCG enable bits without modifying any other settings.
2. Wait for N CCLK cycles (N = PCG source clock period/processor clock period).

3. Program all new parameters without setting the PCG enable bit.
4. Enable the PCG.

Frame Sync Phase Setting

The phase unit requires that the clock and FS is enabled simultaneously in an atomic instruction.

1. Write the clock divider/low 10-bit Phase divider to PCG_CTLx1 register.
2. Program the FS divider/high 10-bit phase divider, enable both the ENCLKx and ENFSx bits in the PCG_CTLx0 registers.

Note that both units must be disabled in the same way.

External Event Trigger

The trigger with the external clock is enabled by setting bits 0 and 16 of the PCG_SYNC register. The phase must be programmed to 3, so that the rising edge of the external clock is in sync with the frame sync (Figure 15-5).

Programming should occur in the following order.

1. Program the PCG_SYNC and the PCG_CTLA0-1, PCG_CTLB0-1 registers appropriately.
2. Enable clock or frame sync, or both.

Since the rising edge of the external clock is used to synchronize with the frame sync, the frame sync output is not generated until a rising edge of the external clock is sensed.

Debug Features

Care should be taken in cases where any input to the phase unit is modified. Any individual change of the CLKDIV or FSDIV dividers may cause a failure in PCG sync operation between the serial clock and the frame sync. Only the programming model ensures a correct setup for phase settings.

16 SERIAL PERIPHERAL INTERFACE PORTS

The ADSP-214xx processors are equipped with two synchronous serial peripheral interface ports that are compatible with the industry-standard serial peripheral interface (SPI). Each SPI port also has its own set of registers (the secondary register set contains a B as in SPIBAUDB). The SPI ports support communication with a variety of peripheral devices including codecs, data converters, sample rate converters, S/PDIF or AES/EBU digital audio transmitters and receivers, LCDs, shift registers, microcontrollers, and FPGA devices with SPI emulation capabilities. The interface specifications are shown in Table 16-1.

Table 16-1. SPI Port Specifications

Feature	SPI/SPIB
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	No
SRU DAI Default Routing	N/A
SRU2 DPI Required	Yes
SRU2 DPI Default Routing	Yes
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	Yes
Transmission Simplex	Yes
Transmission Half Duplex	Yes

Table 16-1. SPI Port Specifications (Cont'd)

Feature	SPI/SPIB
Transmission Full Duplex	Yes (Core and DMA)
Access Type	
Data Buffer	Yes
Core Data Access	Yes
DMA Data Access	Yes
DMA Channels	1
DMA Chaining	Yes
Boot Capable	Yes
Local Memory	No
Max Clock Operation	$f_{PCLK}/4$ (slave) $f_{PCLK}/8$ (master)

Features

The processor's SPI ports provide the following features and capabilities.

- A simple 4-wire interface consisting of two data pins, a device select pin, and a clock pin.
- Special data formats to accommodate little and big endian data, different word lengths, and packing modes.
- Master and multiples slave (multi devices) in which the ADSP-214xx master processor can be connected to up to four other SPI devices.
- Parallel core and DMA access allow full duplex operation.
- Open drain outputs to avoid data contention and to support multimaster scenarios.

- Programmable baud rates, clock polarities, and phases (SPI mode 0–3).
- Master or slave booting from a master SPI device. See "SPI Port Booting" on page 24-12.
- DMA capability to allow transfer of data without core overhead. See "DMA Transfers" on page 16-26.
- Internal loopback mode (by connecting `MISO` to `MOSI`).

Note the SPI interface does not support daisy chain operation, where the `MOSI` and `MISO` pins are internally connected through a FIFO, allowing bypass of data streams.

Pin Descriptions

The SPI protocol uses a 4-wire protocol to enable full-duplex serial communication. Table 16-2 provides detailed pin descriptions and Figure 16-1 shows the master-slave connections between two devices.

Table 16-2. SPI Pin Descriptions

Internal Node	Type	Description
SPI_CLK_I/O SPIB_CLK_I/O	I/O	**SPI Clock Signal.** This control line is clock driven by the master and regulates the flow of the data bits. The master may transmit data at a variety of baud rates. The CLK line cycles once for each bit that is transmitted. It is an output signal if the device is configured as a master; it is an input signal if configured as a slave.
SPI_DS_I SPIB_DS_I	I	**SPI Slave Device Select.** This is an active-low input signal that is used to enable slave devices. This signal is like a chip select signal for the slave devices and is provided by the master device. For a master device, it can act as an error input signal in a multi-master environment. In multi-master mode, if the SPI_DS_I input signal of a master is asserted (Low) an error has occurred. This means that another device is also trying to be the master.
SPI_MOSI_I/O SPIB_MOSI_I/O	I/O	**SPI Master Out Slave In.** This data line transmits the output data from the master device and receives the input data to a slave device. This data is shifted out from the MOSI pin of the master and shifted into the MOSI input(s) of the slave(s).
SPI_MISO_I/O SPIB_MISO_I/O	I/O	**SPI Master In Slave Out.** This data line transmits the output data from the slave device and receives the input data to the master device. This data is shifted out from the MISO pin of the slave and shifted into the MISO input of the master. There may be no more than one slave that is transmitting data during any particular transfer.
SPI_FLG3-0_O SPIB_FLG3-0_O	O	**SPI Slave Select Out.** The slave select pins are used to address up to 4 slaves in a multi device system. This functionality can be routed to any of the DPI pins. This frees up the multiplexed core flags for other purposes.
SPI_CLK_PBEN_O SPIB_CLK_PBEN_O SPI_MOSI_PBEN_O SPIB_MOSI_PBEN_O SPI_MISO_PBEN_O SPIB_MISO_PBEN_O SPI_FLG3-0_PBEN_O SPIB_FLG3-0_PBEN_O	O	**SPI Pin Buffer Enable Out Signal.** Only driven in master mode. The SPIx_FLGx_PBEN_O signals are enabled if the corresponding DSxEN bits in the SPIFLAG register are set.

SRU Programming

Both SPI and SPIB signals are available through the SRU2, and are routed as described in Table 16-3.

Since the SPI supports a gated clock, it is recommended that programs enable the SPI clock output signal with its related pin buffer enable. This can be done using the macro SRU (SPI_CLK_PBEN_O, PBEN_03_I). If these signals are routed statically high as in SRU (high, PBEN_03_I) some SPI timing modes that are based on polarity and phase may not work correctly because the timing is violated.

Table 16-3. SPI SRU2 Signal Connections

SPI Source	DPI Group	SPI Destination
	Group A	SPI_CLK_I SPIB_CLK_I SPI_DS_I SPIB_DS_I SPI_MOSI_I SPIB_MOSI_I SPI_MISO_I SPIB_MISO_I
SPI_CLK_O SPIB_CLK_O SPI_MOSI_O SPIB_MOSI_O SPI_MISO_O SPIB_MISO_O SPI_FLG3–0_O SPIB_FLG3–0_O	Group B	
SPI_CLK_PBEN_O SPIB_CLK_PBEN_O SPI_MOSI_PBEN_O SPIB_MOSI_PBEN_O SPI_MISO_PBEN_O SPIB_MISO_PBEN_O SPI_FLG3–0_PBEN_O SPIB_FLG3–0_PBEN_O	Group C	

Register Overview

This section provides brief descriptions of the major registers. For complete information see "Serial Peripheral Interface Registers" on page A-221.

SPI Control (SPICTLx). Configures the fundamental transfer initiation mode (core or DMA) and configure timing bits and enable the SPI port.

SPI DMA Control (SPIDMACx). Controls the DMA channel on the SPI. Corresponding status bits provide status or error information on transmission.

SPI Flag (SPIFLAGx). Enables the chip selects output in master mode and returns status for errors in multiprocessor systems.

SPI Status (SPISTATx). Provides information on transmission errors for the core.

SPI Baud rate (SPIBAUDx). For master devices, the clock rate is determined by the 15-bit value of the baud rate registers (`SPIBAUDx`) as shown in Table 16-4. For slave devices, the value in the `SPIBAUDx` register is ignored.

Clocking

The fundamental timing clock of the SPI module is peripheral clock/4 (PCLK/4) for slave mode and peripheral clock/8 (PCLK/8) for master mode. In master mode the settings define the SPI master clock.

Master Baud Rate = PCLK/(4 × BAUDR) for BAUDR 2–32767.

The baud rate settings are shown in Table 16-4.

Table 16-4. SPI BAUD Rate – PCLK = 200 MHz

BAUDR Bit Setting	Divider	SPICLK
0	N/A	N/A
1	N/A	N/A
2	8	25
3	12	16.66
4	16	12.5
25	100	2.0 (master boot)
32,767	131068	1526 Hz

The clock to this module may be shut off for power savings. For more information, see “Disabling Peripheral Clocks” on page 23-11.

Choosing the Pin Enable for the SPI Clock

When using the SPI in master mode, and the SPIxCLK signal is routed onto the DPI pin, then the DPI_PBENxx_I signal for that DPI pin being used for the clock must be connected to high.

However, depending on the SPI mode being used (based on the setting of CPHASE and CLKPL bits in the SPICTL register), SPIx_CLK_PBEN_O signal may be used.

Choosing the correct pin enable ensures that the very first edge on SPIx_-CLK (DPI pin) output is not incorrectly chosen as a sampling edge by the slave SPI. Table 16-5 shows the correct pin enable to use for a chosen SPI mode.

Table 16-5. Pin Enable Selection by Mode

Mode	CLKPL	CPHASE	Pin Enable
0	0	0	HIGH
1	0	1	HIGH
2	1	0	SPIx_CLK_PBEN_O
3	1	1	SPIx_CLK_PBEN_O

All other SPI signals SPIx_MOSI, SPIx_MISO and SPIx_FLGx signals when routed on the DPI pins, the SPIx_MISO_PBEN_O, SPIx_MOSI_PBEN_O, or SPIx_FLG_PBEN_O signals should be connected to corresponding DPI_P-BENxx_I signals. The DPI_PBENxx_I signals should not be statically connected to high, as it affects the functioning of certain bits in the SPICTLx register.

Functional Description

Each SPI interface contain its own transmit shift (TXSR, TXSRB) and receive shift (RXSR, RXSRB) registers (not user accessible). The TXSRx registers serially transmit data and the RXSRx registers receive data synchronously with the SPI clock signal (SPICLK). Figure 16-1 shows a block diagram of the SHARC processor SPI interface. The data is shifted into or out of the shift registers on two separate pins: the master in slave out (MISO) pin and the master out slave in (MOSI) pin.

During data transfers, one SPI device acts as the SPI master by controlling the data flow. It does this by generating the SPICLK and asserting the SPI device select signal ($\overline{\text{SPI\_DS\_I}}$). The SPI master receives data using the MISO pin and transmits using the MOSI pin. The other SPI device acts as the SPI slave by receiving new data from the master into its receive shift register using the MOSI pin. It transmits requested data out of the transmit shift register using the MISO pin.

Each SPI port contains a dedicated transmit data buffer (TXSPI, TXSPIB) and a receive data buffer (RXSPI, RXSPIB). Transmitted data is written to TXSPIx and then automatically transferred into the transmit shift register. Once a full data word has been received in the receive shift register, the data is automatically transferred into RXSPIx, from which the data can be read. When the processor is in SPI master mode, programmable flag pins provide slave selection. These pins are connected to the $\overline{\text{SPI\_DS\_I}}$ of the slave devices.

The SPI has a single DMA engine which can be configured to support either an SPI transmit channel or a receive channel, but not both simultaneously. Therefore, when configured as a transmit channel, the received data is essentially ignored. When configured as a receive channel, what is transmitted is irrelevant. A 4-word deep FIFO is included to improve throughput on the IOD0 bus.

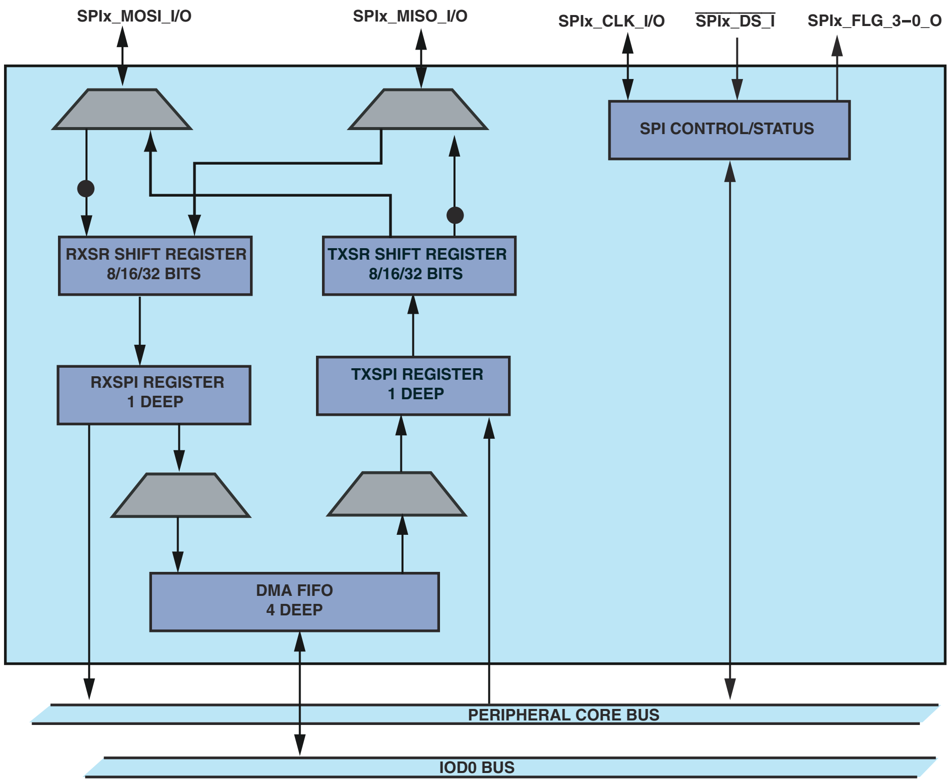

Figure 16-1. SPI Block Diagram

SPI Transaction

An SPI transaction defined start and end depend on whether the device is configured as a master or a slave, whether CPHASE mode is selected, and whether the transfer initiation mode is (TIMOD) selected. For a master SPI with CPHASE = 0, a transfer starts when either the TXSPI register is written or the RXSPI register is read, depending on the TIMOD selection. At the start of the transfer, the enabled slave-select outputs are driven active (low).

However, the SPICLK starts toggling after a delay equal to one-half (0.5) the SPICLK period. For a slave with CPHASE = 0, the transfer starts as soon as the SPI_DS_I input transitions to low.

For CPHASE = 1, a transfer starts with the first active edge of SPICLK for both slave and master devices. For a master device, a transfer is considered complete after it sends and simultaneously receives the last data bit. A transfer for a slave device is complete after the last sampling edge of SPICLK.

Single Master Systems

Figure 16-2 illustrates how the SHARC processor can be used as the slave SPI device. The 16-bit host (A Blackfin ADSP-BF53x processor) is the SPI master. The processor can be booted via its SPI interface to allow application code and data to be downloaded prior to runtime.

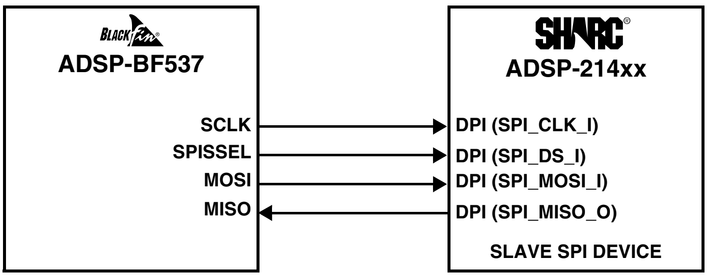

Figure 16-2. SHARC Processor as SPI Slave

Figure 16-3 shows an example SPI interface where the SHARC processor is the SPI master. With the SPI interface, the processor can be directed to alter the conversion resources, mute the sound, modify the volume, and power down the AD1855 stereo DAC.

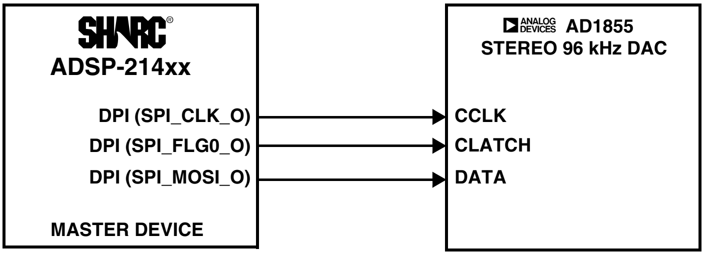

Figure 16-3. SHARC Processor as SPI Master

Multi Master Systems

The SPI does not have an acknowledgement mechanism to confirm the receipt of data. Without a communication protocol, the SPI master has no knowledge of whether a slave even exists. Furthermore, the SPI has no flow control.

Slaves can be thought of as input/output devices of the master. The SPI does not specify a particular higher-level protocol for bus mastership. In some applications, a higher-level protocol, such as a command-response protocol, may be necessary. Note that the master must initiate the frames for both its' command and the slave's response.

Multi master mode allows an SPI system to transfer mastership from one SPI device to another. In a multi device SPI configuration, several SPI ports are connected and any one (but only one) of them can become a master at any given time.

In this configuration, every `MOSI` pin in the SPI system is connected. Likewise, every `MISO` pin in the system is on a single node, and every `SPICLK` pin should be connected (see Figure 16-4). SPI transmission and reception are always enabled simultaneously, unless the broadcast mode has been selected.

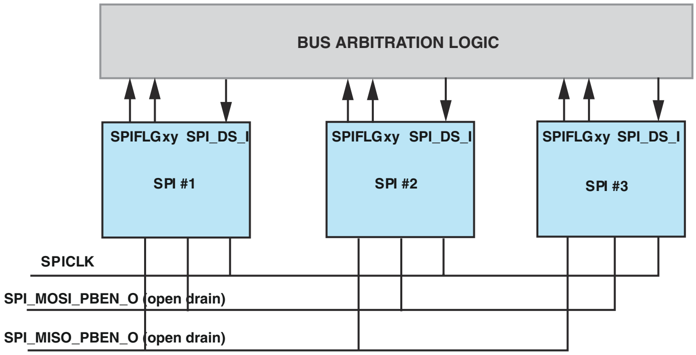

Figure 16-4. Multi-Master System

The master's FLAGx pins connect to each of the slave SPI devices in the system via their SPI_DS_I pins. To enable the different slaves, connect the slave SPI_DS_I pins to the DPI pins of the master SHARC. Since these flags are NOT open drain, slave select pins cannot be shorted together in a multi-master environment. To control slave selects, an external glue logic is required in a multi-master environment.

Another feature is implemented to troubleshoot the bus mastership protocol. If a recent SHARC bus master receives an invalidly asserted SPI_DS_I signal, it triggers an error handling scenario using the MME bit (SPIMME bit for DMA) and ISSEN bit to reconfigure the SPI to slave mode, and jump into an ISR. This ensures that any potential driver conflict is solved. For more information, see "Control Registers (SPICTL, SPICTLB)" on page A-221.

Operating Modes

This sections describes the different mechanisms used for master or slave select operation modes.

Transfer Initiate Mode

When the processor is enabled as a master, the initiation of a transfer is defined by the TIMOD bits (1–0). Based on these two bits and the status of the interface, a new transfer is started upon either a read of the RXSPIx registers or a write to the TXSPIx registers. This is summarized in Table 16-6.

Table 16-6. Transfer Initiation

TIMOD	Function	Transfer Initiated Upon	Action, Interrupt
00	Core Receive and Transmit	Initiate new single word transfer upon read of RXSPI and previous transfer completed. The SPICLK is generated after the data is read from the buffer. In this configuration, a dummy read is needed initially to receive all the data transmitted from the transmitter.	The SPI interrupt is latched in every core clock cycle in which the RXSPI buffer has a word in it. Emptying the RXSPI buffer or disabling the SPI port at the same time (SPIEN = 0) stops the interrupt latch.
01	Core Transmit and Receive	Initiate new single word transfer upon write to TXSPI and previous transfer completed.	The SPI interrupt is latched in every core clock cycle in which the TXSPI buffer is empty. Writing to the TXSPI buffer or disabling the SPI port at the same time (SPIEN = 0) stops the interrupt latch.
10	Transmit or Receive with DMA	Initiate new multiword transfer upon write to DMA enable bit. Individual word transfers begin with either a DMA write to TXSPI or a DMA read of RXSPI depending on the direction of the transfer as specified by the SPIRCV bit.	If chaining is disabled, the SPI interrupt is latched in the cycle when the DMA count decrements from 1 to 0. If chaining is enabled, interrupt function is based on the PCI bit in the CP register. If PCI = 0, the SPI interrupt is latched at the end of the DMA sequence. If PCI = 1, then the SPI interrupt is latched after each DMA in the sequence.
11	Reserved		

SPI Modes

The SPI supports four different combinations of serial clock phases and polarity called SPI modes. The application code can select any of these combinations using the CLKPL and CPHASE bits (10 and 11).

Figure 16-5 on page 16-16 shows the transfer format when CPHASE = 0 and Figure 16-6 on page 16-17 shows the transfer format when CPHASE = 1. Each diagram shows two waveforms for SPICLK—one for CLKPL = 0 and the other for CLKPL = 1. The diagrams may be interpreted as master or slave timing diagrams since the SPICLK, MISO, and MOSI pins are directly connected between the master and the slave. The MISO signal is the output from the slave (slave transmission), and the MOSI signal is the output from the master (master transmission).

The SPICLK signal is generated by the master, and the SPI_DS_I signal represents the slave device select input to the processor from the SPI master. The diagrams represent 8-bit transfers (WL = 0) with MSB first (MSBF = 1). Any combination of the WL and MSBF bits of the SPICTL register is allowed. For example, a 16-bit transfer with the LSB first is one possible configuration.

The clock polarity and the clock phase should be identical for the master device and slave devices 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.

When CPHASE = 0, the slave-select line, SPI_DS_I, must be inactive (HIGH) between each word in the transfer. Even in SPI slave mode when CPHASE = 0, the master should de assert the SPI_DS_I line between each transfer. When CPHASE = 1, SPI_DS_I may either remain active (LOW) between successive transfers or be inactive (HIGH).

Figure 16-5 shows the SPI transfer protocol for CPHASE = 0. Note that SPICLK starts toggling in the middle of the data transfer where the bit settings are WL = 0, and MSBF = 1.

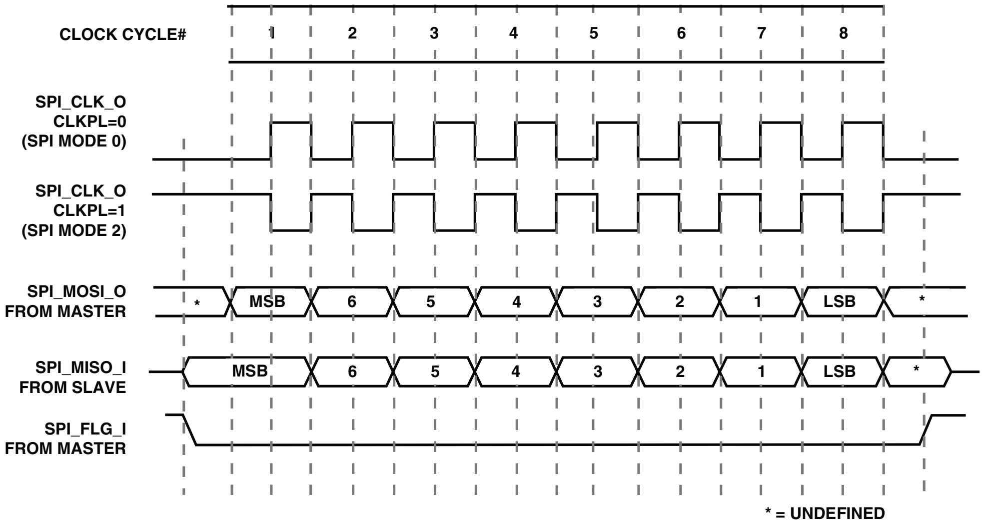

Figure 16-5. SPI Transfer Protocol for CPHASE = 0

Figure 16-6 shows the SPI transfer protocol for CPHASE = 1. Note that SPICLK starts toggling at the beginning of the data transfer where the bit settings are WL = 0, and MSBF = 1.

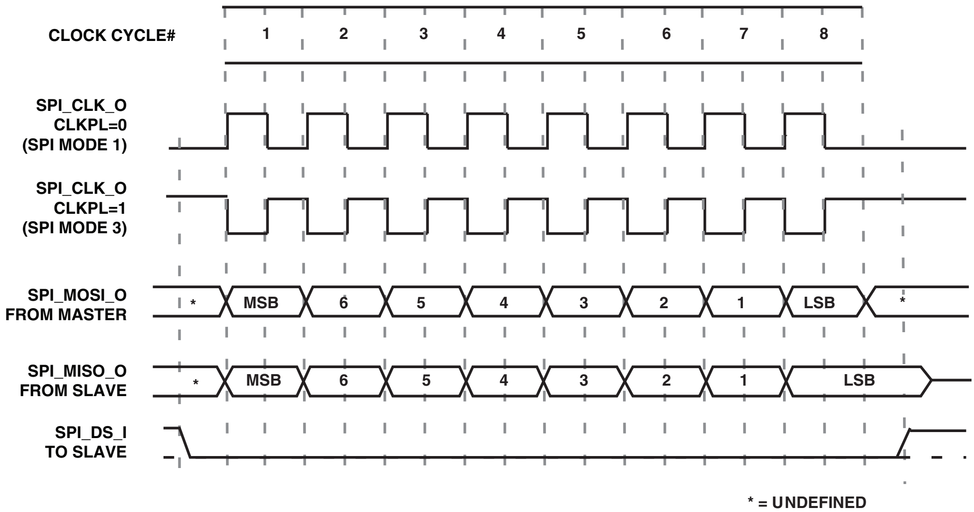

Figure 16-6. SPI Transfer Protocol for CPHASE = 1

Slave Select Outputs

If the SPI is enabled and configured as a master, any of the 14 DPI I/O pins may be used as slave-select outputs. For each DSxEN bit which is set in the SPIFLG register, the corresponding SPI_FLGx_O is configured as a slave-select output.

For example, if DS1EN = 1 is set, SPI_FLG1_O is driven as a slave-select. At the chip-level, SPI_FLG1_O can be connected to any of the DPI pins through SRU programming. For those DSxEN bits which are not set, the corresponding SPIx_FLGx_PBEN_O is driven low.

The behavior of the SPI_FLGx output depends on the value of the CPHASE configuration bit. If CPHASE = 1, all selected outputs may either remain asserted (active-low) between transfers or be deasserted between transfers. This is controlled in software using the SPIFLGx bits (SPIFLG register). For example, to configure SPI_FLG1_O as a slave-select, set DS1EN = 1 and

SPIFLG1 = 0. As soon as this SPIFLG register write takes effect, the SPI_FLG1_O (slave-select output pin) becomes active (Low).

If needed, SPI_FLGx_O can be cycled high and low between transfers by setting the SPIFLG[x] bit to 1 and back to 0. Otherwise, SPI_FLGx_O remains active between transfers.

If CPHASE = 0 or CHPASE = 1 and AUTOSDS = 1, all selected outputs are asserted only for the duration of the transfer. This is controlled by the internal SPI hardware. In this case, the SPIFLGx bits are ignored. For example, to configure SPI_FLG1_O as a slave-select, it is only necessary to set DS1EN=1.

Note that the SPI_FLGx_O signals behave as slave-select outputs only if the SPI module is enabled as a master. Otherwise, none of the bits in the SPIFLG register have any effect.

Variable Frame Delay for Slave

When the processor is configured as an SPI slave, the SPI master must drive an SPICLK signal that conforms with Figure 16-7. For exact timing parameters, please refer to the appropriate product data sheet.

As shown in Figure 16-7, the SPI_DS_I lead time (T1), the SPI_DS_I lag time (T2), and the sequential transfer delay time (T3) must always be greater than or equal to one-half the SPICLK period. The minimum time between successive word transfers (T4) is two SPICLK periods. This time period is measured from the last active edge of SPICLK of one word to the first active edge of SPICLK of the next word. This calculation is independent from the configuration of the SPI (CPHASE, SPIMS, and so on).

This is shown as: T4 = 1.5 SPI clock period + T3 and
T3 = 0.5 SPICLK period for sequential transfer delay (STDC) = 0.
T3 = STDC × SPICLK period for STDC > 0.

Unlike previous SHARC processors, a variable frame delay is included to increase SPI timing flexibility.

For a master device with CPHASE = 0 or CPHASE = 1 (with AUTOSDS set to 1 in the SPCTL register), this means that the slave-select output is inactive (high) for at least one-half the SPICLK period. In this case, T1 and T2 are each always be equal to one-half the SPICLK period.

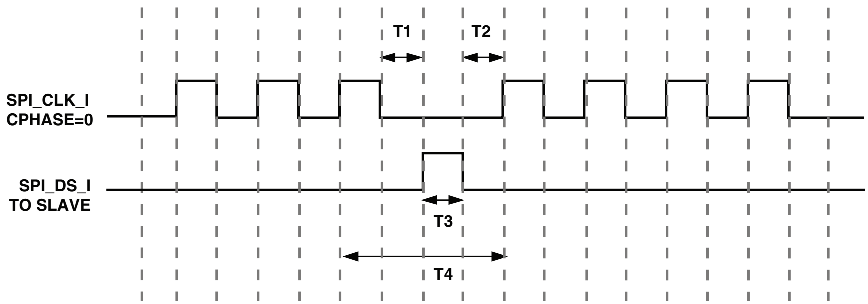

Figure 16-7. SPICLK Timing

When word to word delay is enabled (WTWDEN = 1) in the SPICTL register, then T3 may vary with respect to the value programmed using the STDC bits in the SPIBAUD register. So the word to word delay T4 is:
T4 = 1.5 SPI clock period + T3 and
T3 = 1.5 SPI clock period for STDC = 0, BAUDR = 1, RX master
T3 = 0.5 SPI clock period for STDC = 0, in all other cases.
T3 = STDC × SPI clock period for STDC > 0.

Data Transfers

The SPI is capable of transferring data via the core and DMA. The following sections describe these transfer types.

Serial Shift Register

The SPI allows three different word lengths. The transmit and receive shift registers use these for different packing methods as described below.

Output Shift Register

The transmit shift register receives 32-bit wide data and serially shifts it out externally off-chip.

32-bit word. The Shift register sends the entire 32-bit data.

16-bit word. When transmitting, the shift register sends out only the lower 16 bits of the word written to the SPI buffer.

8-bit word. When transmitting, the shift register sends out only the lower 8 bits of the word written to the SPI buffer.

Input Shift Register

The receive shift register receives its data serially from off chip. Internally the receive shift register is 32 bits wide and data received can be transferred to the buffer.

32-bit word. The shift register receives the entire 32-bit word.

16-bit word. When receiving, the shift register packs the 16-bit word to the lower 32 bits of the `RXSPI` buffer while the upper bits in the register are zeros.

8-bit word. When receiving, the SPI port packs the 8-bit word to the lower 32 bits of the `RXSPI` buffer while the upper bits in the registers are zeros.

Buffers

The SPI contains a transmit and receive buffer which operate as described below.

Transmit Buffer

The transmit buffer is accessible by both the core and DMA. Data is loaded into this register before being transmitted. Just prior to the beginning of a data transfer, the data in TXSPI is loaded into the shift register. A core read of TXSPI can be performed at any time and does not interfere with, or initiate, SPI transfers.

Receive Buffer

The receive buffer is accessible by both the core and DMA. At the end of a data transfer, the data in the shift register is loaded into RXSPI. During a DMA receive operation, the data in the shift register is automatically read by the DMA. When RXSPI is read via the core, the RXS bit is cleared and an SPI transfer may be initiated (only if TIMOD = 00).

Buffer Packing

The SPI unpacks data when it transmits and packs data when it receives. In order to communicate with 8-bit SPI devices and store 8-bit words in internal memory, a packed transfer feature is built into the SPI port.

- PACKEN = 0: No buffer packing
- PACKEN = 1: 8 to 16-bit buffer packing

This bit may be 1 only when WL = 00 (8-bit transfer). When in transmit mode, the PACKEN bit unpacks data. When packing is enabled, two 8-bit words are packed into one 32-bit word. When the SPI port is transmitting, two 8-bit words are unpacked from one 32-bit word. When receiving, words are packed into one 32-bit word from two 8-bit words.

The value 0xXXLMXXJK (where XX is any random value and JK and LM are data words to be transmitted out of the SPI port) is written to the TXSPI register. The processor transmits 0xJK first and then transmits 0xLM.

The receiver packs the two words received, 0xJK and then 0xLM, into a 32-bit word. They appear in the RXSPI register as:

- 0x00LM00JK => if SGN is configured to 0 or L, J < 7
- 0xFFLMFFJK => if SGN is configured to 1 and L, J > 7

Buffer Errors

The following errors are reported in the SPISTAT register.

Transmission Error

This error bit is set when all the conditions of transmission are met and there is no new data in TXSPI (TXSPI is empty). In this case, what is transmitted depends on the state of the SENDZ bit in the SPICTL register. The TUNF bit is cleared by a write-1 (W1C) software operation.

Reception Error

The ROVF flag is set when a new transfer has completed before the previous data could be read from the RXSPI register. This bit indicates that a new word was received while the receive buffer was full. The ROVF bit is cleared by a software write-1 (W1C) operation. The state of the GM bit in the SPICTL register determines whether or not the RXSPI register is updated with the newly received data.

Transmit Collision Error

The TXCOL flag is set when a write to the TXSPI register coincides with the load of the shift register by a write to TXSPI through the core or DMA bus. This bit indicates that corrupt data may have been loaded into the shift register and transmitted. In this case, the data in TXSPI may not match with what was transmitted. It is important to note that this bit is never set when the SPI is configured as a slave with CPHASE = 0; the collision error may occur, but it won't be detected. In any case, this error can easily be avoided by proper software control as described below.

To avoid the TXCOL condition, programs should write to TXSPI well before the load to the shift register takes place. This can be done by writing to TXSPI whenever TXS is cleared and refrain from writing to TXSPI when TXS is set. For slave mode this means that data should be in TXSPI before the first SPI clock edge (or the negative edge of device select) occurs.

However, a potential case of TXCOL arises when there is a TUNF condition while trying to write to TXSPI. In this case TXS is not set and attempts to send new data and it isn't clear if this write to TXSPI takes place when the load to the shift register is occurring (in other words a TXCOL condition). To be absolutely safe, when a TUNF = 1, write to the TXSPI register as soon as SPIF goes from 1 to 0. This ensures that TXSPI is written into well before the next load to the shift register takes place.

The TXCOL bit is cleared by a software write-1 (W1C) operation.

Flush Buffer

The SPI RX/TX buffers are flushed by disabling the SPI port or by setting the TXFLSH/RXFLSH bits. The SPI DMA buffer is flushed only by setting the FIFOFLSH bit.

ⓘ None of the three flush bits in SPI (TXFLSH, RXFLSH, and FIFOFLSH) is self clearing. They have to be explicitly cleared by the software.

Core Buffer Status

For core accesses to the SPI, master and slave modes operate differently as described below.

1. If core access to a SPI slave is unable to keep up with the transmit/receive stream during a transfer operation (because of an interrupt or any other reason) the SPI operates according to the states of the SENDZ and GM bits in the SPICTLx register.
   - If SENDZ = 1 and the transmit buffer is empty, the device repeatedly transmits zeros on the MOSI pin. One word is transmitted for each new transfer initiate command.
   - If SENDZ = 0 and the transmit buffer is empty, the device repeatedly transmits the last word transmitted before the transmit buffer became empty.
   - If GM = 1 and the receive buffer is full, the device continues to receive new data from the MISO pin, overwriting the older data in the RXSPI buffer.
   - If GM = 0 and the receive buffer is full, the incoming data is discarded, and the RXSPI register is not updated.
2. If core access to a SPI master is unable to keep up with the transmit/receive stream during a transfer operation (because of an interrupt or another reason) the SPI stalls the SPICLK until new data is read/written into the TXSPI/RXSPI buffers. In this scenario the TUNF/ROVF condition bits are set indicating an exception in the data stream.

DMA Buffer Status

If the DMA engine is unable to keep up with the transmit/receive stream during a transfer operation because of latency caused by using multiple

DMA channels, the SPI operates according to the states of the SENDZ and GM bits in the SPICTLx register.

- If SENDZ = 1 and the transmit buffer is empty, the device repeatedly transmits zeros on the MOSI pin. One word is transmitted for each new transfer initiate command.
- If SENDZ = 0 and the transmit buffer is empty, the device repeatedly transmits the last word transmitted before the transmit buffer became empty.
- If GM = 1 and the receive buffer is full, the device continues to receive new data from the MISO pin, overwriting the older data in the RXSPI buffer.
- If GM = 0 and the receive buffer is full, the incoming data is discarded, and the RXSPI register is not updated.

Core Transfers

The RXS bit defines when the receive buffer can be read. The TXS bit defines when the transmit buffer can be filled. The end of a single word transfer occurs when the RXS bit is set. This indicates that a new word has been received and latched into the receive buffer, RXSPI. The RXS bit is set shortly after the last sampling edge of SPICLK. There is a 4 PCLK cycle latency for a master/slave device, depending on synchronization. This is independent of the CPHASE, TIMOD bit settings, and the baud rate.

Backward Compatibility

To maintain software compatibility with other SPI devices (68HC11), the SPI transfer finished bit (SPIF) is also available for polling. This bit may have slightly different behavior from that of other commercially available devices. For a slave device, SPIF is set at the same time as RXS. For a master device, SPIF is set one-half (0.5) of the SPICLK period after the last SPICLK edge, regardless of CPHASE or CLKPL. The baud rate determines when the

SPIF bit is set. In general, SPIF is set after RXS, but at the lowest baud rate settings (SPIBAUD < 4). The SPIF bit is set before the RXS bit, and consequently before new data has been latched into the RXSPI buffer. Therefore, for SPIBAUD = 2 or SPIBAUD = 3, the processor must wait for the RXS bit to be set (after SPIF is set) before reading the RXSPI buffer. For larger SPIBAUD settings (SPIBAUD > 4), RXS is set before SPIF.

DMA Transfers

The SPI ports support both master and slave mode DMA. DMA is enabled for TIMOD bit = 10.

Enable the SPI port before enabling DMA.

For master mode, a DMA transfer starts after the DMA engine is enabled. For slave mode the slave select pin (SPI_DS_I) needs to be asserted to start slave DMA operation.

When enabled as a master, the DMA engine transmits or receives data as follows:

- If the SPI system is configured for transmitting, the DMA engine reads data from memory into the DMA FIFO. Data from the DMA FIFO is loaded into the TXSPIx buffer and then into the transmit shift register. This initiates the transfer on the SPI port.
- If configured to receive, data from the RXSPIx buffer is automatically loaded into the DMA FIFO as long as FIFO is not full. (It is recommended to flush the DMA FIFO before initiating the transfer, if there is no valid data in the FIFO).

  Once the data from RXSPIx gets written into the FIFO, the DMA engine reads data from the DMA FIFO and writes to memory. Then the SPI initiates the receive transfer. The SPI generates the programmed signal pulses on SPICLK and the data is shifted out of MOSI and in from MISO simultaneously. The SPI continues sending or receiving words until the DMA word count register transitions from 1 to 0. When the SPI is configured as master, the SPI continues to generate SPICLK until the DMA FIFO is full, even if the DMA word count transitions to zero.

Do not write to the TXSPIx buffer during an active SPI transmit DMA operation because DMA data will be overwritten. Similarly, do not read from the RXSPIx buffer during active SPI DMA receive operations. DMA interrupts are generated based on DMA events and are configured in the SPIDMACx registers. In order for a transmit DMA operation to begin, the transmit buffer (TXSPIx) must initially be empty (TXS = 0). While this is normally the case, this means that the TXSPIx buffer should not be used for any purpose other than SPI transfers. Writing to the TXSPIx buffer via the software sets the TXS bit.

For receive master DMA, the SPICLK stops only when the RXSPI buffer and DMA FIFO are full (even if the DMA count is already zero). Therefore, SPICLK runs for an additional five word transfers filling junk data in the RXSPIx buffer and DMA FIFO. The FIFOs must be flushed before a new DMA is initiated. In some slave devices such as SPI flash, the starting address is usually sent along with the read command in the beginning. The read address later increments automatically after every read. These additional clock cycles might fetch additional words in the FIFO from the SPI flash device and thus might result in unintended increment of the read address of the flash memory. If another read DMA has to be initiated to read the following data from the flash, flushing the DMA FIFO may not be practical as it might result in data loss. In such cases, either of the two following methods can be used to avoid the data loss:

1. Flush both the DMA FIFO and the SPI receive buffer, send a new read command with the appropriate start address to the slave device again, and then re-initialize the read DMA. The advantage

of this method is that the successive DMA transfers can be made completely independent and thus even disabling the SPI after one DMA is done and re-enabling it again before initializing the next DMA does not result in any data loss.

2. After completion of a DMA transfer, do not flush the FIFO, and do not modify the contents of the SPICTL and SPIDMAC registers. Change the DMA index and modifier registers (if required), and finally re-initialize the DMA count register to initiate a new DMA. This method avoids software overhead required to flush the FIFO and send a new read command for each DMA. It should be noted that the SPI and the DMA engine can't be disabled when using this method. Also, the DMA index and/or modifier register values should be changed only after the ongoing DMA is finished and before loading the DMA count register for the next DMA transfer.

DMA Chaining

The serial peripheral interfaces support both single and chained DMA. However, unlike the serial ports, programs cannot insert a TCB in an active chain. For more information, see "SPI TCB" on page 3-15.

Configuring and starting chained DMA transfers over the SPI port is the same as that of the serial ports, with one exception. Contrary to SPORT DMA chaining, (where the first DMA in the chain is configured by the first TCB), for SPI DMA chaining, the first DMA is not initialized by a TCB. Instead, the first DMA in the chain must be loaded into the SPI parameter registers (IISPI, IMSPI, CSPI, IISPIB, IMSPIB, CSPIB), and the chain pointer registers (CPSPI, CPSPIB) point to a TCB that describes the second DMA in the sequence.

ⓘ Writing an address to the CPSPIx, registers does not begin a chained DMA sequence unless the IISPI, IMSPI, CSPI, IISPIB, IMSPIB, and CSPIB registers are initialized, SPI DMA is enabled, the SPI port is enabled, and SPI DMA chaining is enabled.

DMA Transfer Count

When the SPI is configured for receive/transmit DMA, the number of words configured in the DMA count register should match the actual data transmitted. When the SPI DMA is used, the internal DMA request is generated for a DMA count of four. In case the count is less than four, one DMA request is generated for all the bytes.

For example, when a DMA count of 16 is programmed, four DMA requests are generated (that is, four groups of four). For a DMA count of 18, five DMA requests are generated (four groups of four and one group of two). In case the SPI DMA is programmed with a value more than the actual data transmitted, some bytes may not be received by the SPI DMA due to the condition for generating the DMA request.

Full Duplex Operation

The SPI interface allows full-duplex operation running the DMA channel to the transmit/receive path and core access to the alternate transmit/receive path. For full-duplex operation, set `TIMOD` = 10 which generates the interrupts for DMA only.

Reads from the `RXSPIx` buffer are allowed at any time during transmit DMA. Note the `TXS` bit is cleared when the `TXSPIx` buffer is read but the DMA FIFO is not available in the receive path. The receive interface cannot generate an interrupt, but the `RXS` status bits can be polled.

Writes to the `TXSPIx` buffer during an active SPI receive DMA operation are permitted. Note the `RXS` bit is cleared when the `RXSPIx` buffer is read but the DMA FIFO is not available in the transmit path. The transmit interface cannot generate an interrupt, but the `TXS` status bits can be polled.

Interrupts

Table 16-7 provides an overview of SPI interrupts.

Table 16-7. Overview of SPI Interrupts

Default Programmable Interrupt	Sources	Masking	Service
SPIHI = P1I SPILI = P18I	DMA complete Core buffer service Internal transfer completion Access completion	TIMOD bit (SPICTL) INTEN bit (SPIDMAC)	RTI instruction
	DMA buffer underflow DMA Buffer overflow Multimaster error Transmit collision error	INTERR bit (SPIDMAC)	RW1C to SPISTAT + RTI instruction

Sources

The SPI module drives one interrupt signal, SPIHI/SPILI. The internal status for core/DMA and protocol are logical ORed into the interrupt signal.

The primary SPI uses the SPIHI interrupt and the secondary SPI uses the SPILI interrupt. Whenever an SPI interrupt occurs (regardless of the cause), the SPILI or SPIHI interrupts are latched. The SPI ports can generate interrupts as described in the following sections.

Core Buffer Service Request

When DMA is disabled the processor core may read from the RXSPI buffer or write to the TXSPI buffer. An interrupt is generated when the receive buffer is not empty or the transmit buffer is not full.

If configured to generate an interrupt when RXSPI is full (TIMOD = 00), the interrupt becomes active 1 PCLK cycle after the RXS bit is set.

Data Buffer Packing

When SPI port data packing is enabled (PACKEN = 1 in the SPICTL registers), the transmit and receive interrupts are generated for 32-bit packed words, not for each 16-bit word.

DMA Complete

For receive or transmit DMA after the DMA counter is zero.

Internal Transfer Complete

Depending upon the state of INTETC bit the interrupt can be generated when the internal count becomes zero or the external transfer is complete. At the completion of a single DMA transfer when DMA count = 0 and INTETC bit is zero.

Access Complete

The DMA interrupt is generated when DMA count reaches zero (INTETC = 0) or the DMA interrupt is generated when last bit of last word is shifted out or when the last data is transferred externally (INTETC = 1). This setting also generates an interrupt at the completion of a number of DMA sequences when DMA chaining is enabled.

Chained DMA

For chained DMA, if the PCI bit is cleared (= 0), the DMA complete interrupt is generated only after the entire chained DMA access is complete. If the PCI bit is set (= 1), then a DMA interrupt is generated for each TCB.

DMA Buffer Over/Underflow

For DMA transfers (TIMOD = 10) interrupts are latched in case for receive buffer overflow (SPIOVF bit) or transmit buffer underflow (SPIUNV bit).

Multimaster Error

The SPIMME bit (1) is set when the SPI_DS_I input pin of a device that is enabled as a master is driven low by some other device in the system. This occurs in multimaster systems when another device is also trying to be the master.

Masking

The SPIHI and SPILI signals are routed by default to programmable interrupt. To service the primary SPI port, unmask (set = 1) the P1I bit in the IMASK register. To service the secondary SPIB port, unmask (set = 1) the P18IMSK bit in the LIRPTL register. For example:

```
bit set IMASK P1I;          /* unmasks P1I interrupt */
bit set LIRPTL P18IMSK;     /* unmasks P18I interrupt */
```

The TIMOD bit in the SPICTL register determines whether the interrupt is based on DMA or on core buffer service request (TXSPI or RXSPI buffer).

For DMA transfer status based interrupts, set the INTEN bit in the SPIDMAC register. Note that the SPIDMAC register must be initialized properly to enable DMA interrupts.

In order to trigger data stream errors set the INTERR bit in the SPIDMAC register. This triggers an error interrupt condition when a receive buffer overflow (SPIRCV bit = 1) or a transmit buffer underflow (SPIRCV bit = 0) occur.

To detect errors in a multi-master environment, set the ISSEN bit in the SPICTL register to trigger an interrupt for a conflict situation.

Service

As soon as DMA buffer under/overflow error is detected by reading the SPISTAT register, the ISR should perform a RW1C operation on the bit that caused the exception in the SPISTAT register.

As soon as master error is detected, the following actions are taken:

1. The SPIMS control bit in SPICTL is cleared, configuring the SPI interface as a slave.
2. The SPIEN control bit in SPICTL is cleared, disabling the SPI system.
3. The MME status bit in SPISTAT is set.
4. An SPI interrupt is generated.

These four conditions persist until the MME bit is cleared by a read-write 1-to-clear (RW1C type) software operation. Until the MME bit is cleared, the SPI cannot be re-enabled, even as a slave. Hardware prevents the program from setting either SPIEN or SPIMS while MME is set.

When MME is cleared, the interrupt is deactivated. Before attempting to re-enable the SPI as a master, the state of the SPI_DS_I input pin should be checked to ensure that it is high; otherwise, once SPIEN and SPIMS are set, another mode-fault error condition will immediately occur. The state of the input pin is reflected in the input slave select status bit (bit 7) in the SPIFLG register.

As a result of SPIEN and SPIMS being cleared, the SPI data and clock pin drivers (MOSI, MISO, and SPICLK) are disabled. However, the slave-select output pins revert to control by the processor flag I/O module registers. This may cause contention on the slave-select lines if these lines are still being driven by the processor.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

SPI Effect Latency

After the SPI registers are configured the effect latency is 2 PCLK cycles to enable and 2 PCLK cycles to disable.

Programming Model

The section describes which sequences of software steps are required to get the peripheral working successfully.

SPI Routing

For proper master operation configure the MOSI, MISO, SPICLK and SPI_FLGx_O master output select. For slave operation route the MOSI, MISO, SPICLK signals including the SPI_DS_I as slave select input.

Master Mode Transfers

For core or DMA transfers, when the SPI is configured as a master, the ports should be configured and transfers started using the following steps:

1. Route all required signals (MOSI, MISO, SPICLK) for master mode including the SPI_FLGx_0 as slave select outputs.
2. Before enabling the SPI port, programs should specify which of the slave-select signals (DPI pins) to use, setting one or more of the required SPI flag select bits (DSxEN) in the SPIFLGx register. For DMA operation set TIMOD = 10.
3. Set AUTOSDS bit to 1, to ensure the slave-selects are automatically controlled by the SPI port. (When AUTOSDS = 0, only the CPHASE = 0 setting has automated control as with previous SHARC processors)
4. Write to the SPICTLx register and set the SPIMS bit to enable the device as a master. Configure the SPIBAUDx registers, and configuring the appropriate word length, transfer format, baud rate, and other necessary information.

The next steps are dependant on whether the access is a core or a DMA access.

Core Master Transfers

When a device is to be used as a master, configure the ports using the following procedure.

1. Initiate the SPI transfer by writing or reading to/from SPI buffers. The trigger mechanism for starting the transfer is dependent upon the TIMOD bits in the SPICTLx registers. See Table 16-6 on page 16-14 for more details.

2. The SPI generates the programmed clock pulses on SPICLK. The data is shifted out of MOSI and shifted in from MISO simultaneously. Before starting to shift, the transmit shift register is loaded with the contents of the TXSPIx registers. At the end of the transfer, the contents of the receive shift register are loaded into the RXSPI buffer.

3. With each new buffer access, the SPI continues to send and receive words, according to the SPI transfer mode (TIMOD bit in SPICTLx registers). See Table 16-6 on page 16-14 for more details.

4. If there are no further SPI buffer accesses the SPICLK signal is stalled until new core requests are received.

DMA Master Transfers

To configure the SPI port for master mode DMA transfers:

1. Define DMA receive (or transmit) transfer parameters by writing to the IISPIx, IMSPIx, and CSPIx registers.

2. Write to the SPIDMACx register to enable the SPI DMA engine (SPIDEN, bit 0). And configure the following:
   - A receive access (SPIRCV = 1) or
   - A transmit access (SPIRCV = 0)

Slave Mode Transfers

When the SPI is configured as a master, regardless of core or DMA the SPI ports should be configured and transfers started using the following steps.

1. Route all required signals (MOSI, MISO, SPICLK) for slave mode including the SPI_DS_I as slave select input.
2. Write to the SPICTLx and keep the (SPIMS) cleared, enabling the device as a slave and configuring the SPI system by specifying the appropriate word length, transfer format and other necessary information. For DMA operation set TIMOD = 10.

The next steps are dependant on whether the access is a core or a DMA access.

Core Slave Transfers

The following steps illustrate SPI operation in slave mode.

1. Write the data to be transmitted into the TXSPIx buffer to prepare for the data transfer.
2. When a device is enabled as a slave, the start of a transfer is triggered by a transition of the SPI_DS_I select signal to the active state (low) or by the first active edge of the clock (SPICLK), depending on the state of CPHASE.
3. The reception or transmission continues until SPI_DS_I is released or until the slave has received the proper number of clock cycles.
4. The slave device continues to receive or transmit with each new falling-edge transition on SPI_DS_I or active SPICLK clock edge.

DMA Slave Transfers

To configure the SPI port for slave mode DMA transfers:

1. Define DMA receive (or transmit) transfer parameters by writing to the IISPIx, IMSPIx, and CSPIx registers.
2. Write to the SPIDMACx register to enable the SPI DMA engine (SPIDEN, bit 0) and configure the following:

- A receive access (SPIRCV = 1) or
- A transmit access (SPIRCV = 0)

Chained DMA Transfers

The sequence for setting up and starting a chained DMA is outlined in the following steps.

1. Clear the chain pointer register.
2. Configure the TCB associated with each DMA in the chain except for the first DMA in the chain.
3. Write the first three parameters for the initial DMA to the IISPI, IMSPI, CSPI, IISPIB, IMSPIB, and CSPIB registers directly.
4. Configure the DMA settings for the entire sequence, enabling DMA and DMA chaining in the SPIDMAC register.
5. Begin the DMA by writing the address of a TCB (describing the second DMA in the chain) to the CPSPI, CPSPIB registers.

Stopping SPI Transfers

External transfer completion is indicated by the SPI status bit SPIFE. For core-driven transfers it shows that the read transfer (TIMOD = 00) or write transfer (TIMOD = 01) has been completed on the external interface. For receive DMA the status bit is asserted when the DMA count becomes zero. For transmit DMA the SPIFE goes high when:

- the DMA count becomes zero and
- the DMA FIFO becomes empty and
- the TXSPI buffer becomes empty (TXS bit high) and
- transfer is complete (SPIF bit goes high)

Note that the `SPIFE` bit can go high between two DMA blocks of a chained DMA.

Changing SPI Timing Configuration

Programs should take the following precautions when changing SPI configurations.

- The SPI configuration must not be changed during a data transfer.
- Change the clock polarity only when no slaves are selected.
- Change the SPI configuration when `SPIEN` = 0. For example, if operating as a master in a multislave system, and there are slaves that require different data or clock formats, then the master SPI should be disabled, reconfigured, and then re-enabled.

However, when a SPI communication link consists of:

1. A single master and a single slave,
2. `CPHASE` = 1 and `AUTOSDS` = 0 for master, `CPHASE` = 1 for slave
3. The slave's slave select input is tied low

Then the program can change the SPI configuration. In this case, the slave is always selected. Data corruption can be avoided by enabling the slave only after configuring both the master and slave devices.

Switching From Transmit DMA to a New DMA

The following sequences detail the steps for switching from transmit to transmit/receive DMA. In the first sequence the SPI is disabled then re enabled. In the second the SPI buffers and registers are cleared but the SPI itself is not disabled.

Disabling SPI:

1. Poll the SPIFE bit in the SPISTAT register. If this bit is high the SPI can be disabled. The external transfer done interrupt (DMA done interrupt with INTETC bit set) can as well be used if polling SPIFE has to be avoided.

2. Clear the SPICTLx register to disable the SPI. Disabling the SPI also clears the RXSPIx/TXSPIx buffer and the buffer status.

3. Disable DMA by clearing the SPIDMACx register (write 0x00000000 to it).

4. Clear all errors by writing to the RW1C-type bits in the SPISTATx registers. This ensures that no interrupts occur due to errors from a previous DMA operation.

5. Reconfigure the SPICTLx register and enable the SPI ports.

6. Configure the new DMA by writing to the DMA parameter registers and the SPIDMACx registers and enable the DMA using the SPIDEN bit (bit 0).

Not disabling SPI:

1. Poll the SPIFE bit in the SPISTAT register. If this bit is high the SPI buffer can be cleared. The external transfer done interrupt (DMA done interrupt with INTETC bit set) can be used to avoid polling the SPIFE bit.

2. Clear the RXSPIx/TXSPIx buffers and the buffer status without disabling the SPI. This can be done by ORing 0xC0000 with the present value in the SPICTLx register. For example, programs can use the RXFLSH and TXFLSH bits to clear RXSPIx/TXSPIx and the buffer status.

3. Clear the SPIDMAC register by writing 0x00000000 to it.

4. Clear all errors by writing to the RW1C-type bits in the SPISTAT register. This ensures that no interrupts occur due to errors from a previous DMA operation.

5. Reconfigure the SPICTL register to remove the clear condition on the TXSPI/RXSPI registers.

6. Configure the new DMA by writing to the DMA parameter registers and the SPIDMACx registers and enable the DMA using the SPIDEN bit (bit 0).

Switching From Receive to a New DMA

Use the following sequence to switch from receive to transmit DMA. Note that TXSPIx and RXSPIx are registers but they may not contain any bits, only address information. In the first sequence the SPI is disabled then re enabled. In the second the SPI buffers and registers are cleared but the SPI itself is not disabled.

Disabling SPI:

1. Poll the SPIFE bit in the SPISTAT register. If this bit =1 the SPI can be disabled.

2. Clear the SPICTLx registers to disable the SPI. Disabling the SPI also clears the RXSPIx/TXSPIx register contents and the buffer status.

3. Disable DMA and clear the DMA FIFO by setting the FIFOFLSH bit is the SPIDMACx register (write 0x00000080 to it). This ensures that any data from a previous DMA operation is cleared as the SPICLK signal runs for five more word transfers even after the DMA count falls to zero in the receive DMA.

4. Clear all errors by writing to the SPISTATx registers. This ensures that no interrupts occur due to errors from a previous DMA operation.

5. Reconfigure the SPICTLx registers and enable the SPI.

6. Configure the new DMA by writing to the DMA parameter registers and the SPIDMACx registers and enable the DMA using the SPIDEN bit (bit 0). Since the flush bits (TXFLSH, RXFLSH, and FIFOFLSH) are not self clearing in the SPI, ensure that the FIFOFLSH bit in the SPIDMACx (which was set in step 3) is cleared in this step.

No disabling SPI:

1. Poll the SPIFE bit in the SPISTAT register. If this bit =1 the SPI can be disabled.

2. Clear the RXSPIx/TXSPIx registers and the buffer status without disabling the SPI by ORing 0xC0000 with the present value in the SPICTLx registers. Use the RXFLSH (bit 19) and TXFLSH (bit 18) bits in the SPICTLx registers to clear the RXSPIx/TXSPIx registers and the buffer status.

3. Disable the DMA and clear the DMA FIFO using the FIFOFLSH bit in the SPIDMACx register (write 0x00000080 to it). This ensures that any data from a previous DMA operation is cleared because SPICLK runs for five more word transfers even after the DMA count is zero in receive DMA.

4. Clear all errors by writing to the RW1C-type bits in the SPISTATx registers. This ensures that no interrupts occur due to errors from a previous DMA operation.

5. Reconfigure the SPICTLx registers to remove the clear condition on the TXSPIx/RXSPIx registers.

6. Configure the new DMA by writing to the DMA parameter registers and the SPIDMACx register and enable the DMA using the SPIDEN bit (bit 0). Since the flush bits (TXFLSH, RXFLSH, and FIFOFLSH) are not self clearing in the SPI, ensure that the FIFOFLSH bit in the SPIDMACx (which was set in step 3) is cleared in this step.

Switching from Receive DMA to Receive DMA Without Disabling the SPI and DMA

For receive master DMA, the SPICLK stops only when the RXSPI buffer and DMA FIFO are full (even if the DMA count is already zero). Therefore, the SPICLK runs for an additional five word transfers, filling extra data in the RXSPIx buffer and DMA FIFO.

In some SPI slave devices such as a SPI flash, the starting read address is usually sent along with the read command in the beginning. The read address later increments automatically after every read. These additional clock cycles might fetch additional words in the FIFO from the SPI flash device and thus might result in an unintended increment of the read address. If another read DMA has to be initiated to read the following data from the flash, the above programming model (disabling the DMA and or SPI) may not be practical as it might result in data loss. For such a case, the following programming model can be used.

1. Poll the SPIFE bit of the SPISTAT register. If this bit =1, this indicates that the previous DMA transfer is complete and the additional words (to keep the DMA FIFO and the RXSPI register full) are also received.
2. Re initialize the DMA index and modifier registers (if required).
3. Re initialize the DMA count register to the required non-zero value to initiate the new receive DMA.

DMA Error Interrupts

The SPIUNF and SPIOVF bits of the SPIDMACx registers indicate transmission errors during a DMA operation in slave mode. When one of the bits is set, an SPI interrupt occurs. The following sequence details the steps to respond to this interrupt.

With SPI disabled:

1. Disable the SPI port by writing 0x00 to the SPICTLx registers.
2. Disable DMA and clear the DMA FIFO by FIFOFLSH bit in the SPIDMACx register. This ensures that any data from a previous DMA operation is cleared before configuring a new DMA operation.
3. Clear all errors by writing to the RW1C-type bits in the SPISTATx registers. This ensures that the error bits SPIOVF and SPIUNF (in the SPIDMACx registers) are cleared when a new DMA is configured.
4. Reconfigure the SPICTLx registers and enable the SPI using the SPIEN bit.
5. Configure DMA by writing to the DMA parameter registers and the SPIDMACx registers.

With SPI enabled:

1. Disable DMA and clear the DMA FIFO by FIFOFLSH bit in the SPIDMACx register. This ensures that any data from a previous DMA operation is cleared before configuring a new DMA operation.
2. Clear the RXSPIx/TXSPIx registers and the buffer status without disabling SPI. This can be done by ORing 0xC0000 with the present value in the SPICTLx registers. Use the RXFLSH and TXFLSH bits to clear the RXSPIx/TXSPIx registers and the buffer status.
3. Clear all errors by writing to the RW1C-type bits in the SPISTAT register. This ensures that error bits SPIOVF and SPIUNF in the SPIDMACx registers are cleared when a new DMA is configured.
4. Reconfigure the SPICTL register to remove the clear condition on the RXSPI/TXSPI register bits.
5. Configure DMA by writing to the DMA parameter registers and the SPIDMACx register.

Multi-Master Transfers

The following steps show how to implement a system with two SPI devices. Since the slaves cannot initiate transfers over the bus, the master must send frames over the MOSI pin. This ensures that slaves can respond to the bus by sending messages over the MISO pin to the bus master.

1. Slave writes message to its MISO pin.
2. Slave starts polling its SPI_DS_I pin which is currently low.
3. Message is latched by current master and decoded.
4. Master deasserts the slave select signal and clears the SPIMS bit to become a slave.
5. If bus requester detects the SPI_DS_I pin high, it sets the SPIMS bit to get bus mastership.
6. The master selects a slave by driving its' slave select flag pin.

Debug Features

The following sections provide information on features that help in debugging SPI software.

Shadow Receive Buffers

A pair of read-only (RO) shadow registers for the receive data buffers, RXSPI and RXSPIB are available for use in debugging software. These registers, RXSPI_SHADOW and RXSPIB_SHADOW, are located at different addresses from RXSPI, but their contents are identical to that of RXSPI. When RXSPI is read from core, the RXS bit is cleared (read only-to-clear) and an SPI transfer may be initiated (if TIMOD = 00). No such hardware action occurs when the shadow register is read. RXSPI_SHADOW is only accessible by the core.

> When transferring data from one SPI configured as slave to another SPI configured as master in DMA mode, the following steps should be followed to avoid data loss.

1. Enable slave SPI DMA.
2. Wait for the TX buffer of the slave to be full by polling the `TXS` bit (bit 3) of the `SPIxSTAT` register.
3. Enable the master SPI DMA.

Internal Loopback Mode

In this mode different types of loopback are possible since there is only one DMA channel available:

- Core receive and transmit transfers
- Transmit DMA and core receive transfers
- Core Transmit and DMA receive transfers

To loop data back from `MOSI` to `MISO`, the `MISO` pin is internally disconnected. The `MOSI` pin will contain the value being looped back. Programs should set the `SPIEN`, `SPIMS`, and `ILPBK` bits in the `SPICTLx` register.

> Loopback operation is only used in master mode.

Loopback Routing

The SPI supports an internal loopback mode using the SRU. For more information, see "Loopback Routing" on page 10-40.

17 PERIPHERAL TIMERS

In addition to the internal core timer, the ADSP-214xx processors contain identical 32-bit peripheral timers that can be used to interface with external devices. Each timer can be individually configured in three operation modes. The timers specifications are shown in Table 17-1.

Table 17-1. Timer Specifications

Feature	Timer1–0
Connectivity	
Multiplexed Pinout	No
SRU DPI Required	Yes
SRU DPI Default Routing	Yes
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	Yes
Transmission Simplex	N/A
Transmission Half Duplex	N/A
Transmission Full Duplex	N/A
Access Type	
Data Buffer	No
Core Data Access	N/A
DMA Data Access	N/A
DMA Channels	N/A

Table 17-1. Timer Specifications (Cont'd)

Feature	Timer1–0
DMA Chaining	N/A
Boot Capable	N/A
Local Memory	No
Max Clock Operation	$f_{PCLK}/4$

Features

The peripheral timers have the features described below.

- Independent general-purpose timers.
- Three operation modes (PWM, Width capture, external watchdog).
- Global control/status registers for synchronous operation of multiple timers.
- Buffered timer registers (Period and Width) to allow changes on the fly.
- Supported timer period in the range from $4 \times t_{PCLK}$ to $2 \times 10^9 \times t_{PCLK}$.

The core timer is controlled by system registers while the peripheral timers are controlled by memory-mapped registers. For information on system registers, see *SHARC Processor Programming Reference*.

Pin Descriptions

The timer has only one pin which acts as input or output based on the timer mode as shown in Table 17-2.

Table 17-2. Peripheral Timer Pin Descriptions

Internal Node	Type	Description
TIMER1–0_I	I	**Timer Signal.** This input is active sampled during pulse width and period capture (width capture mode) or external event watchdog (external clock mode).
TIMER1–0_O	O	**Timer Signal.** This output is active driven in pulse width modulation (PWM out mode).
TIMER1–0_PBEN_O	O	**Timer Pin Buffer Enable Output Signal.** This output is only driven in PWM out mode.

SRU Programming

Since the timer has operation modes for input (capture and external clock mode) and output (PWM out mode), it requires bidirectional junctions. Table 17-3 shows the required SRU routing.

Table 17-3. Peripheral Timer SRU2 Signal Connections

TIMERx Source	DPI Group	TIMERx Destination
TIMER1–0_O	Group A	TIMER1–0_I
TIMER1–0_O	Group B	
TIMER1–0_PBEN_O	Group C	

See also "DPI Routing Capabilities" on page 10-25.

Register Overview

The following sections provide brief descriptions of the primary registers used to program the timers. For complete information on the timer registers, see "Peripheral Timer Registers" on page A-264.

Control Registers (TMxCTL). Controls the operation mode (external clock, width capture, PWM out) and enables interrupt flow. Bit for waveform control is also provided in this register.

Global Status and Control Register (TMSTAT). Indicates the status of both timers using a single read. The TMSTAT register also contains timer enable bits. Within TMSTAT, each timer has a pair of sticky status bits, that require a write one-to-set (TIMxEN) or write one-to-clear (TIMxDIS) to enable and disable the timer respectively.

Counter Registers (TMxCNT). When disabled, the timer counter retains its state. When re-enabled, the timer counter is re initialized from the period/width registers based on configuration and mode. The timer counter value should not be set directly by the software. It can be set indirectly by initializing the period or width values in the appropriate mode. The counter should only be read when the respective timer is disabled. This prevents erroneous data from being returned.

Period Registers (TMxPRD). When enabled and running, the processor writes new values to the timer period and pulse width registers. The writes are buffered and do not update the registers until the end of the current period (when the timer counter register equals the timer period register).

Pulse Width Register (TMxW). During the pulse width modulation (PWM_OUT), the width value is written into the timer width registers. Both width and period register values must be updated "on the fly" since the period and width (duty cycle) change simultaneously. To insure period and width value concurrency, a 32-bit period buffer and a 32-bit width buffer are used.

Read-Modify-Write

The traditional read-modify-write operation to enable/disable a peripheral is different for the timers. For more information, see "Peripheral Timer Registers" on page A-264.

Clocking

The fundamental timing clock of the peripheral timers is peripheral clock (PCLK). The clock to this module may be shut off for power savings. For more information, see "Disabling Peripheral Clocks" on page 23-11.

Functional Description

Each timer has one dedicated bidirectional chip signal, TIMERx. The two timer signals are connected to the 14 digital peripheral interface (DPI) pins through the signal routing unit (SRU). The timer signal functions as an output signal in PWM_OUT mode and as an input signal in WDTH_CAP and EXT_CLK modes. To provide these functions, each timer has four, 32-bit registers shown in Figure 17-1.

During the *pulse width modulation* (PWM_OUT), the period value is written into the timer period registers. Both period and width register values must be updated "on the fly" since the period and width (duty cycle) change simultaneously. To insure the period and width value concurrency, a 32-bit period buffer and a 32-bit width buffer are used.

During the *pulse width and period capture* (WDTH_CAP) mode, the period values are captured at the appropriate time. Since both the period and width registers are read-only in this mode, the existing 32-bit period and width buffers are used (see Figure 17-1).

During the *external event watchdog* (EXT_CLK) mode, the period register is write-only. Therefore, the period buffer is used in this mode to insure high/low period value coherency. When the processor is in EXT_CLK mode, the width register is unused.

When clocked internally, the clock source is the processor's peripheral clock (PCLK). The timer produces a waveform with a period equal to 2 x TMxPRD and a width equal to 2 × TMxW. The period and width are set through the TMxPRD30-0 and the TMxW30-0 bits. Bit 31 is ignored for both.

The equation for the timer period is: 2 × (Period Register) × t_{PCLK}.

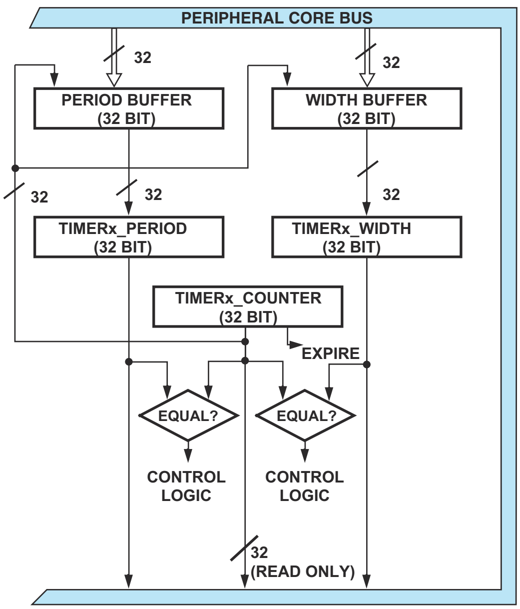

Figure 17-1. Timer Block Diagram with Buffered Period and Width Registers

Operating Modes

The three operating modes of the peripheral timer; PWM_OUT, WDTH_CAP, and EXT_CLK, are described in Table 17-4 and the following sections.

Table 17-4. Timer Bits Comparison

Bits	PWM_OUT Mode	WIDTH_CAP Mode	EXT_CLK Mode
Timer Control Registers (TMxCTL)			
TIMODE	01 = PWM Out	10 = Width Capture	11 = External Clock
PULSE	1 = Generate High Width 0 = Generate Low Width	1 = Measure High Width 0 = Measure Low Width	1 = Count at event rise 0 = Count at event fall
PRDCNT	1 = Generate PWM 0 = Single Width Pulse	1 = Measure Period 0 = Measure Width	Unused
IRQEN	1 = Enable Interrupt 0 = Disable Interrupt		
Timer Status Register (TMSTAT)			
TMxOVF (IRQ also set)	Set if Initialized with: Period < Width or Period == Width or Period == 0	Set if the Counter wraps (Error Condition)	Unused
TMxIRQ (If enabled)	If PRDCNT: 1 = Set at end of Period 0 = Set at end of Width		Set after period expires and PCLK is running
TIMxEN	Enable and start timer		
TIMxDIS	Disable timer		
Counter Registers			
TMxPRD	WO: Period value	RO: Period value	WO: Period value
TMxW	WO: Width value	RO: Width value	Unused
TMxCNT	RO: Only if not enabled Counts down on PCLK		RO: Only if not enabled counts down on event

Pulse Width Modulation Mode (PWM_OUT)

In PWM_OUT mode, the timer supports on-the-fly updates of period and width values of the PWM waveform. The period and width values can be updated once every PWM waveform cycle, either within or across PWM cycle boundaries.

To enable PWM_OUT mode, set the TIMODE1-0 bits to 01 in the timer's configuration (TMxCTL) register. This configures the timer's TIMERx signal as an output with its polarity determined by PULSE as follows:

- If PULSE is set (= 1), an active high width pulse waveform is generated at the TIMERx signal.
- If PULSE is cleared (= 0), an active low width pulse waveform is generated at the TIMERx signal.

The timer is actively driven as long as the TIMODE field remains 01.

Figure 17-2 shows a flow diagram for PWM_OUT mode. When the timer becomes enabled, the timer checks the period and width values for plausibility (independent of the value set with the PRDCNT bit) and does *not* start to count when any of the following conditions are true:

- Width is equal to zero
- Period value is lower than width value
- Width is equal to period

On invalid conditions, the timer sets both the TIMxOVF and the TIMIRQx bits and the Count register is not altered. Note that after reset, the timer registers are all zero. The PWM_OUT timing is shown in Figure 17-3.

As mentioned earlier, 2 x TMxPRD is the period of the PWM waveform and 2 x TMxW is the width. If the period and width values are valid after the timer is enabled, the count register is loaded with the value resulting from 0xFFFF FFFF – width. The timer counts upward to 0xFFFF FFFF.

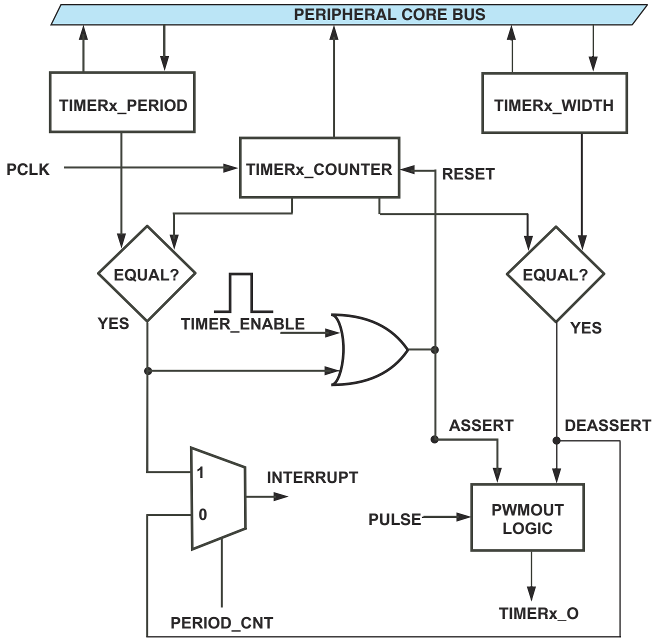

Figure 17-2. Timer Flow Diagram – PWM_OUT Mode

Instead of incrementing to 0xFFFF FFFF, the timer then reloads the counter with the value derived from 0xFFFF FFFF – (period – width) and repeats.

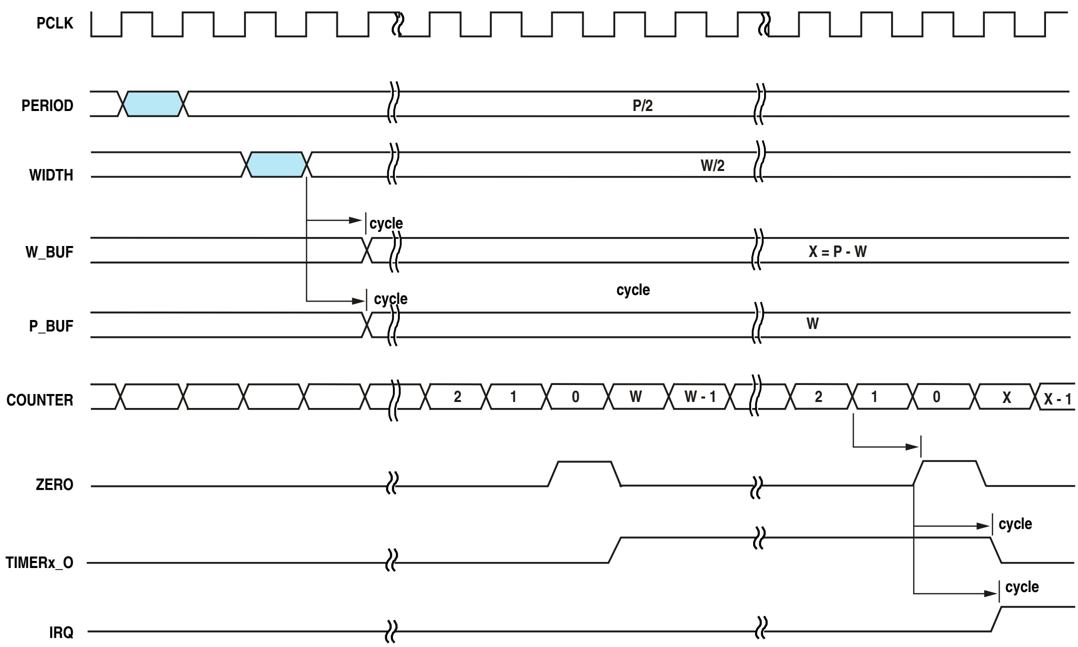

Figure 17-3. PWM_OUT Timing

PWM Waveform Generation

If the PRDCNT bit is set, the internally-clocked timer generates rectangular signals with well-defined period and duty cycles. This mode also generates periodic interrupts for real-time processing.

The 32-bit period (TMxPRD) and width (TMxW) registers are programmed with the values of the timer count period and pulse width modulated output pulse width.

When the timer is enabled in this mode, the TIMERx signal is pulled to a deasserted state each time the pulse width expires, and the signal is asserted again when the period expires (or when the timer is started).

To control the assertion sense of the TIMERx_O signal, the PULSE bit in the corresponding TMxCTL register is either cleared (causes a low assertion level) or set (causes a high assertion level).

When enabled, a timer interrupt is generated at the end of each period. An ISR must clear the interrupt latch bit TIMxIRQ and might alter period and/or width values. In pulse width modulation applications, the program can update the period and pulse width values while the timer is running.

When a program updates the timer configuration, the TMxW register must always be written to last, even if it is necessary to update only one of the registers. When the TMxW value is not subject to change, the ISR reads the current value of the TMxW register and rewrites it again. On the next counter reload, all of the timer control registers are read by the timer.

To generate the maximum frequency (or minimum period) on the TIMERx_O output signal, set the period value to 2 and the pulse width to 1. This makes the TIMERx signal toggle every 2 PCLK clock cycles as shown in Figure 17-9 on page 17-21. Assuming PCLK = 133 MHz:

$$\text{Maximum period} = 2 \times (2^{31} - 1) \times 7.5 \text{ ns} = 32 \text{ seconds.}$$

If your application requires a more sophisticated PWM output generator, refer to Chapter 8, Pulse Width Modulation.

Single-Pulse Generation

If the PRDCNT bit is cleared, the PWM_OUT mode generates a single pulse on the TIMERx_O signal. This mode can also be used to implement a well defined software delay that is often required by state machines. The pulse width (= 2 × TMxW) is defined by the width register and the period register should be set to a value greater than the pulse width register.

At the end of the pulse, the interrupt latch bit (TIMxIRQ) is set and the timer is stopped automatically. If the PULSE bit is set, an active high pulse is generated on the TIMERx_O signal. If the PULSE bit is not set, the pulse is active low.

Pulse Mode

The waveform produced in PWM_OUT mode with PRDCNT = 1 normally has a fixed assertion time and a programmable deassertion time (via the TMxW register). When both timers are running synchronously by the same period settings, the pulses are aligned to the asserting edge as shown in Figure 17-4. Note that the timer does not support toggling of the PULSE bit in each period.

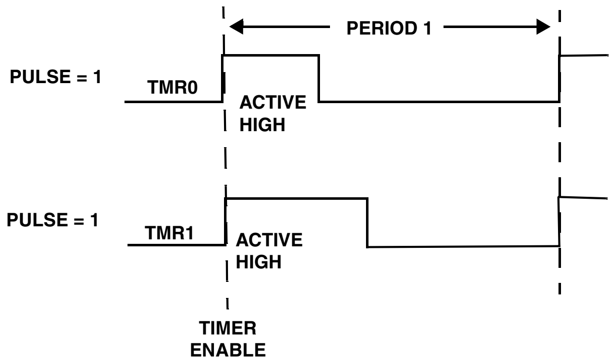

Figure 17-4. Timers with Pulses Aligned to Asserting Edge

Pulse Width Count and Capture Mode (WDTH_CAP)

To enable WDTH_CAP mode, set the TIMODE1-0 bits in the TMxCTL register to 10. This configures the TIMERx signal as an input signal with its polarity determined by PULSE. If PULSE is set (= 1), an active high width pulse waveform is measured at the TIMER_Ix signal. If PULSE is cleared (= 0), an active low width pulse waveform is measured at the TIMERx_I signal. The internally-clocked timer is used to determine the period and pulse width of externally-applied rectangular waveforms. The period and width registers are read-only in WDTH_CAP mode. The period and pulse width measurements are with respect to a clock frequency of PCLK ÷ 2.

Figure 17-5 shows a flow diagram for WDTH_CAP mode. In this mode, the timer resets words of the count in the TMxCNT register value to 0x0000 0001 and does not start counting until it detects the leading edge on the TIMERx_I signal.

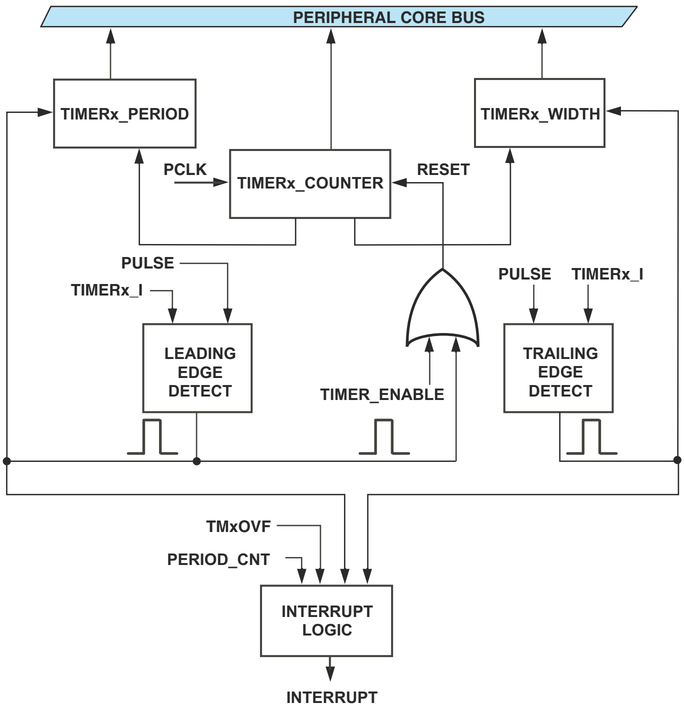

Figure 17-5. Timer Flow Diagram – WDTH_CAP Mode

When the timer detects a first leading edge, it starts incrementing. When it detects the trailing edge of a waveform, the timer captures the current value of the count register (= TMxCNT ÷ 2) and transfers it into the TMxW width registers. At the next leading edge, the timer transfers the current value of the count register (= TMxCNT ÷ 2) into the TMxPRD period register.

The count registers are reset to 0x0000 0001 again, and the timer continues counting until it is either disabled or the count value reaches 0xFFFF FFFF.

In this mode, programs can measure both the pulse width and the pulse period of a waveform. To control the definition of the leading edge and trailing edge of the TIMERx_I signal, the PULSE bit in the TMxCTL register is set or cleared. If the PULSE bit is cleared, the measurement is initiated by a falling edge, the count register is captured to the WIDTH register on the rising edge, and the period register is captured on the next falling edge.

The PRDCNT bit in the TMxCTL register controls whether an enabled interrupt is generated when the pulse width or pulse period is captured. If the PRDCNT bit is set, the interrupt latch bit (TIMxIRQ) gets set when the pulse period value is captured. If the PRDCNT bit is cleared, the TIMxIRQ bit gets set when the pulse width value is captured.

If the PRDCNT bit is cleared, the first period value has not yet been measured when the first interrupt is generated. Therefore, the period value is not valid. If the interrupt service routine reads the period value anyway, the timer returns a period value of zero. When the period expires, the period value is loaded in the TMxPRD register.

A timer interrupt (if enabled) is also generated if the count register reaches a value of 0xFFFF FFFF. At that point, the timer is disabled automatically, and the TIMxOVF status bit is set, indicating a count overflow. The TIMxIRQ and TIMxOVF bits are sticky bits, and programs must explicitly clear them. The WDTH_CAP timing is shown in Figure 17-6.

The first width value captured in WDTH_CAP mode is erroneous due to synchronizer latency. To avoid this error, programs must issue two NOP instructions between setting WDTH_CAP mode and setting TIMxEN.

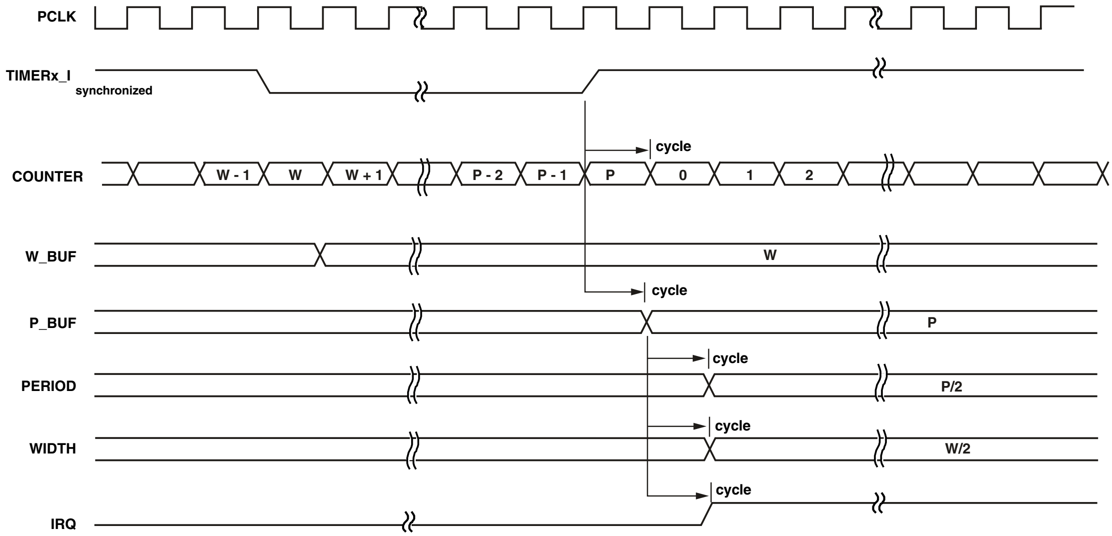

Figure 17-6. WDTH_CAP Timing (Period Count = 1)

External Event Watchdog Mode (EXT_CLK)

Figure 17-7 shows a flow diagram for EXT_CLK mode. To enable EXT_-CLK mode, set the TIMODE1-0 bits in the TMxCTL register to 11 in the TMxCTL register. This samples the TIMERx_I signal as an input. Therefore, in EXT_CLK mode, the TMxCNT register should not be read when the counter is running.

The operation of the EXT_CLK mode is as follows:

1. Program the TMxPRD period register with the value of the maximum timer external count.
2. Set the TIMxEN bits. This loads the period value in the count register and starts the countdown.
3. When the period expires, an interrupt, (TIMxIRQ) occurs.

After the timer is enabled, it waits for the first rising edge on the TIMERx_I signal. The rising edge forces the count register to be loaded by the value (0xFFFF FFFF – TMxPRD). Every subsequent rising edge increments the count register. After reaching the count value 0xFFFF FFFE, the TIMxIRQ bit is set and an interrupt is generated. The next rising edge reloads the count register with (0xFFFF FFFF – TMxPRD) again.

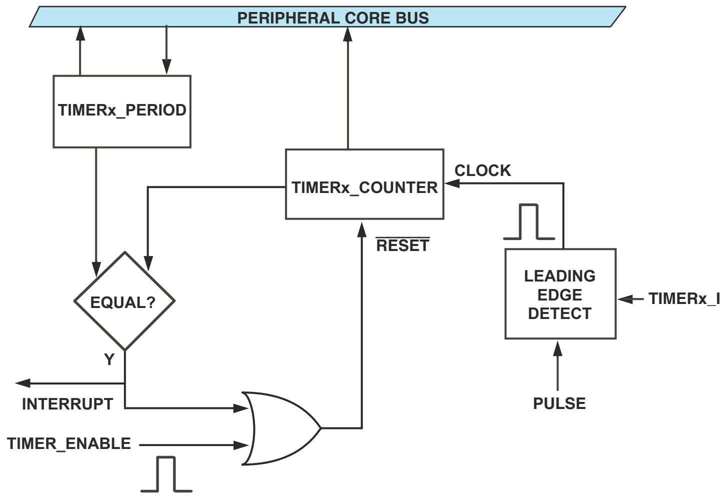

Figure 17-7. Flow Diagram EXT_CLK Mode

The EXT_CLK timing is shown in Figure 17-8.

The configuration bit, PRDCNT, has no effect in this mode. Also, TIMxOVF is never set and the width register is unused.

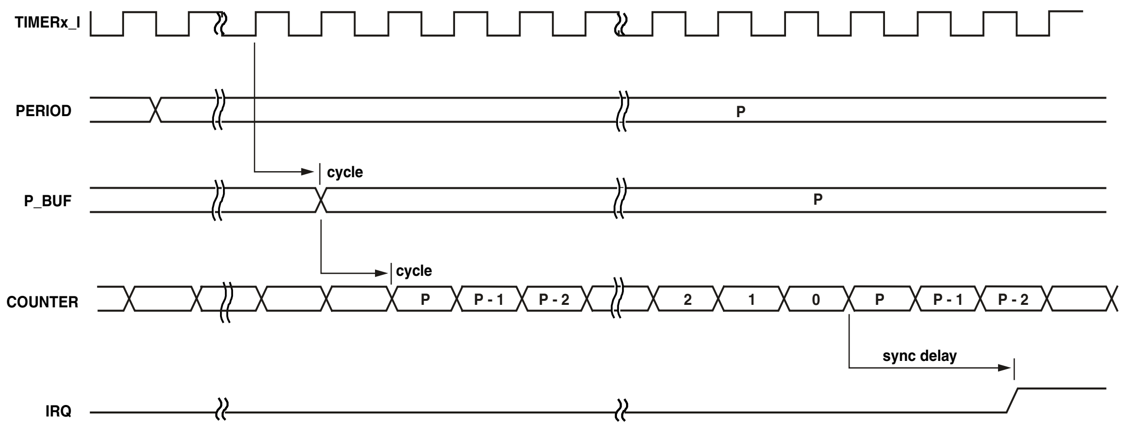

Figure 17-8. EXT_CLK Timing

Interrupts

Table 17-5 provides an overview of timer interrupts.

Table 17-5. Overview of Timer Interrupts

Default Programmable Interrupt	Sources	Masking	Service
GPTMR0I = P2I GPTMR1I = P10I	Timer Expire Timer Overflow	IRQEN-bit (TMxCTL)	RW1C to TMxSTAT + RTI instruction

Sources

The timer module drives one interrupt signal, `GPTIMERxI`.

Each timer generates a unique interrupt request signal. A common register latches these interrupts so that a program can determine the interrupt source without reference to the timer's interrupt signal. The timers can generate interrupts under the conditions described in the following sections.

PWM_OUT Mode

Depends on the PRDCNT bit setting as follows.
1 = Set at the end of period
0 = Set at the end of width

WDTH_CAP Mode

Depends on the PRDCNT bit setting as follows.
1 = Set at the end of period
0 = Set at the end of width

EXT_CLK Mode

Set after Period expires and PCLK is running

PWM_OUT Mode

Set for programming errors initialized with:
Period < Width or
Period == Width or
Period == 0

WDTH_CAP Mode

Set if the counter wraps, error condition.

Masking

The GPTMR0I and GPTMR1I signals are routed by default to programmable interrupt. To service the GPTMR0I signal, unmask (set = 1) the P2I bit in the IMASK register. To service the secondary GPTMR1I, unmask (set = 1) the P10IMSK bit in the LIRPTL register. For example:

```
bit set IMASK P2I;      /* unmasks P2I interrupt */
bit set LIRPTL P10IMSK;  /* unmasks P10I interrupt */
```

To enable a timer's interrupt, set the IRQEN bit in the timer's configuration (TMxCTL) register With the IRQEN bit cleared, the timer does not set its interrupt latch (TIMxIRQ) bits. To poll the TIMxIRQ bits without generating a timer interrupt, programs can set the IRQEN bit while leaving the timer's interrupt masked.

Service

The TMSTAT register contains an interrupt latch bit (TIMxIRQ) and an overflow/error indicator bit (TIMxOVF) for each timer.

With interrupts enabled, ensure that the interrupt service routine (ISR) clears the TIMxIRQ latch before the RTI instruction to assure that the interrupt is not serviced erroneously. In external clock (EXT_CLK) mode, the latch should be reset at the very beginning of the interrupt routine so as not to miss any timer event.

These sticky bits are set by the timer hardware and may be watched by software. They need to be cleared in the TMSTAT register by software explicitly.To clear, write a one to the corresponding bit in the TMSTAT register as shown in Listing 17-1.

Listing 17-1. Clearing Sticky Bits

```
TMR0_ISR:
ustat2=TIM0IRQ;
dm(TM0STAT)=ustat2;  /* RW1C the Timer0 bit */
r10=dm(TM0CTL);      /* dummy read for write latency */
instructions;
instructions;
RTI;
```

Interrupt and overflow bits may be cleared simultaneously with timer enable or disable.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

Timers Effect Latency

After the timer registers are configured the effect latency is 3 PCLK cycles enable and 2 PCLK cycles disable. The timer starts 3 PCLK cycles after the TIMEN bit is set.

When the timer is enabled, the count register is loaded according to the operation mode specified in the TMxCTL register. When the timer is disabled, the counter registers retain their state; when the timer is re-enabled, the counter is reinitialized based on the operating mode (Figure 17-9). The program should never write the counter value directly.

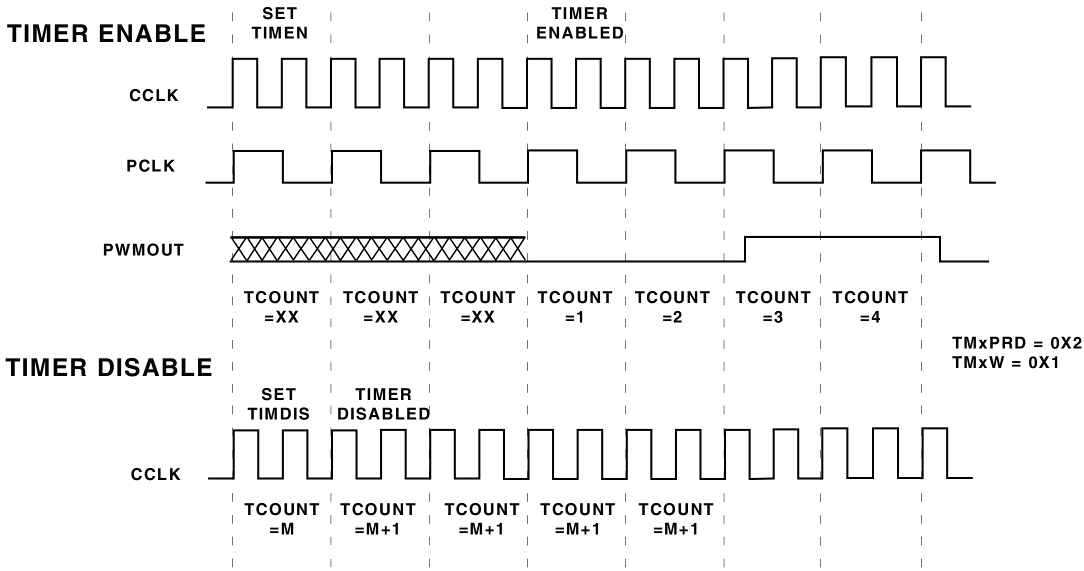

Figure 17-9. Timer PWM Enable and Disable Timing

Programming Model

The section describes which sequences of software steps are required to get the peripheral working successfully.

To enable an individual timer, set the timer's `TIMxEN` bit in the `TMSTAT` register. To disable an individual timer, set the timer's `TIMxDIS` bit in the `TMSTAT` register. To enable both timers in parallel, set all the `TIMxEN` bits in the `TMSTAT` register.

Before enabling a timer, always program the corresponding timer's configuration (`TMxCTL`) register. This register defines the timer's operating mode, the polarity of the `TIMERx` signal, and the timer's interrupt behavior. Do not alter the operating mode while the timer is running. For more information, see "Timer Control Registers (TMxCTL)" on page A-265.

PWM Out Mode

Use the following procedure to configure and run the timer in PWM out mode.

1. Reset the TIMEN bit and set the configuration mode to 01 to select PWM_OUT operation. This configures the TIMERx_0 pin as an output pin with its polarity determined by the PULSE bit.
   - The timer outputs a positive active pulse width at the TIMERx_0 pin.
   - The timer outputs a negative active pulse width at the TIMERx_0 pin.
2. Initialize the period before the width register values. Insure that the period value is greater than the width value.
3. Set the TIMEN bit. The timer performs boundary exception checks on the period and width values:
   - If (width == 0 or Period < width or period == width) both the OVF_ERR and $\overline{\text{IRQ}}$ bits are set.
   - If there are no exceptions, the width value is loaded into the counter and it starts counting.

   The timer produces PWM waveform with a period of 2 x period and a width of 2 x width.
   - When 2 x width expires, the counter is loaded with 2x(period – width) and continues counting.
   - When 2 x period expires, the counter is loaded with 2 x width value again and the cycle repeats.
   - When the width or period expires, the $\overline{\text{IRQ}}$ bit (if enabled) is set depending on the PRDCNT bit.

- When $\overline{IRQ}$ is sensed, read the status register (TMxSTAT) and perform the appropriate read-write-to-clear.

WDTH_CAP Mode

Use the following procedure to configure and run the timer in WDTH_-CAP out mode.

1. Reset the TIMEN bit and set the configuration mode to 10 to select WDTH_CAP operation. This configures the TIMERx_I pin as an input pin with its polarity determined by the PULSE bit.
   - The timer measures a positive active pulse width at the TIM-ERx_I pin.
   - The timer measures a negative active pulse width at the TIM-ERx_I pin.
2. The PRDCNT bit determines when the $\overline{IRQ}$ status bit (if enabled) is set.
   - If (PRDCNT == 1), $\overline{IRQ}$ is set when the period expires and the value is captured.
   - If (PRDCNT == 0), $\overline{IRQ}$ is set when the width expires and the value is captured.
3. Valid period and width values are set in their respective registers when $\overline{IRQ}$ is set.

   The period and width values are measured with respect to PCLK. This makes this mode coherent with the PWM_OUT mode, where the output waveforms have a period of 2 x period and a width of 2 x width.

Note that the first period value will not have been measured when the first width is measured, so it is not valid. The timer sets and returns a period value of zero in this case. When the period expires, the period value is placed into the period register. When $\overline{\text{IRQ}}$ is sensed, read the status and perform the appropriate RW1C operation.

EXT_CLK Mode

Use the following procedure to configure and run the timer in EXT_CLK out mode.

1. Reset the TIMEN bit and set the configuration mode to 11 to select EXT_CLK operation.

   This configures the TIMERx_I pin as an input pin regardless of the setting of the PULSE bit. Note that the timer always samples the rising edge in this mode. The period register is WO and the width register is unused in this mode.

2. Initialize the period register with the value of the maximum external count.

3. Set the TIMEN bit. This loads the period value in the counter and starts the count down.

   When the period expires, it is reloaded with the period value and the cycle repeats. Counter counts with each edge of the input waveform, asynchronous to PCLK.

   When the period expires, $\overline{\text{IRQ}}$ (if enabled) is set and TMR_IRQ is asserted. An external clock can trigger the Timer to issue an interrupt and wake up an idle processor.

Reads of the count register are not supported in EXT_CLK mode.

Debug Features

The following section provides information on debugging features available with the timer.

Loopback Routing

The timer support an internal loopback mode by using the SRU. For more information, see “Loopback Routing” on page 10-40. An emulation halt will not stop the timer period counter.

18 SHIFT REGISTER – ADSP-2147x

ADSP-2147x processors incorporate an 18 stage serial in, serial/parallel out Shift Register (SR). The serial in–serial out mode can be used to delay the serial data by a fixed amount of time. The serial output can also be used to cascade the shift register modules on two or more processors. The serial in–parallel out mode can be used to convert the serial data to parallel. Table 18-1 lists the shift register specifications.

Table 18-1. Shift Register Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	Yes
SRU DAI Default Routing	Yes
SRU2 DPI Required	No
SRU2 DPI Default Routing	No
Interrupt Control	N/A
Protocol	
Master Capable	N/A
Slave Capable	N/A
Transmission Simplex	N/A
Transmission Half Duplex	N/A
Transmission Full Duplex	N/A

Table 18-1. Shift Register Specifications (Cont'd)

Feature	Availability
Access Type	
Data Buffer	Yes
Core Data Access	N/A
DMA Data Access	N/A
DMA Channels	N/A
DMA Chaining	N/A
Boot Capable	N/A
Local Memory	No
Clock Operation	$f_{PCLK}/4$

Features

The following list describes the features of the shift register.

- 18-stage serial/parallel shift register.
- 18-bit parallel data latch.
- 18 parallel output signals (SR_LDO17-0) with can be three-stated.
- Serial data input (SR_SDI) and output pins (SR_SDO) allows cascading of multiple SR registers.
- SRU routing unit allows the input selection for clock and data from SPORT7-0, PCGA-B, DAI Pin buffer 8–1 or external SR pins.
- Pin buffers remain three-stated coming out of reset until configured by software as outputs.

Pin Descriptions

The pin descriptions for the shift register are described in the ADSP-2147x data sheet.

SRU Programming

To use the shift register, route the required inputs using the SRU as described in Table 18-2, taking not of the following.

- The SR_SCLK, SR_LAT, and SR_SDI inputs must come from the same source except in the cases:
  - where SR_SCLK comes from PCGA/B then SPORT0–7 generates the SR_LAT and SR_SDI signals or
  - where SR_SCLK and SR_LAT come from PCGA/B then SPORT0–7 generates the SR_SDI signal.
- Configure CKRE = 1 (SPCTL register) when using SPORT0–7 as a source of SR_SCLK_I, SR_LAT_I, and SR_DAT_I signals.

Table 18-2. SR SRU Connections

Shift Register Source	DAI Connection	Shift Register Destination
SR_SCLK_O (dedicated pin) SR_LAT_O (dedicated pin)	Group H	SR_CLK_I SR_LAT_I
SR_SDAT_O (dedicated pin)	Group I	SR_SDI_I

The shift register input pins (SR_CLK_I, SR_LAT_I, SR_SDI_I) are routed by default to the external shift register pins (SR_CLK, SR_LAT, SR_SDI).

Register Overview

The processor contains registers that are used to control the shift register.

- **Control Register (SR_CTL).** Used to clear/reset the shift register in software, select the data source for the SR_SDO pin out of the 18 bits of the register, and to enable parallel data output. Complete bit descriptions can be found at "Shift Register Control Register" on page A-208.
- **Clock Routing Register (SRU_CLK_SHREG).** Configures the clock source. For more information, see "Destination Control Signal Register (SR_CLK_SHREG)" on page A-152.
- **Data Routing Register (SRU_DAT_SHREG).** Configures the data source. For more information, see "Group I – Shift Register Serial Data Routing Register (ADSP-2147x)" on page A-153.

Clocking

The shift register requires two clock inputs: SR_SCLK_I for the serial shift register and SR_LAT_I for the latch. The source of these clocks is selectable out of many sources such as the SPORTs, PCGA/B, DAI pin buffers 8–1, or dedicated SR_SCLK and SR_LAT input pins. The data is shifted on the rising edge of the SR_SCLK_I and the data from the shift register is transferred to the latch on rising edge of the SR_LAT_I. If both clocks are connected together, the shift register is always one clock pulse ahead of the latch.

Functional Description

The Shift Register module consists of an 18-stage serial shift register, 18-bit latch, and three-state output buffers. Three-state buffers are implemented in I/O buffers. The shift register and latch have separate clocks.

Data is shifted on the positive-going transitions of the SR_SCLK_I input. The data in each flip-flop is transferred to the respective latch on a positive-going transition of the SR_LAT_I input. The shift register has a serial data input (SR_SDI_I) and a serial data output (SR_SDO) for cascading.

A common active low asynchronous reset (SR_CLR_I) is provided for 18-bit shift register and for 18-bit latch. As shown in the Figure 18-1, the latch has 18 parallel outputs to drive three-state output buffers. Data in the latch appears at the output whenever the output enable input (SR_LDOE_I) is high.

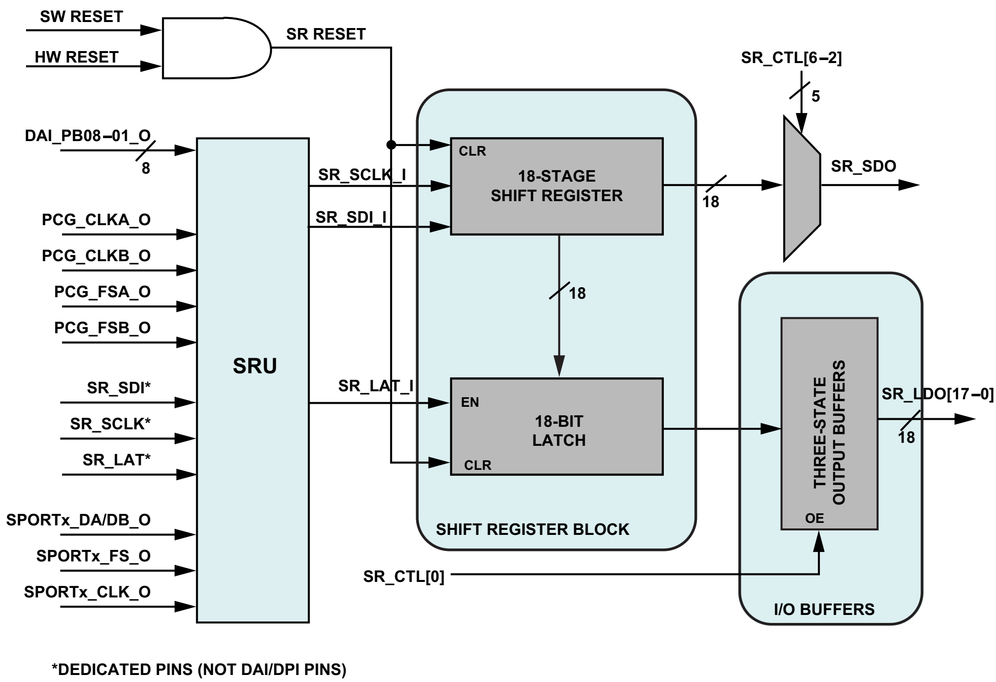

Figure 18-1. Shift Register Block Diagram

The SR_CLR_I signal is derived from an dedicated pin (SR_CLR), and a software programmable reset (SR_SW_CLR bit in the SR_CTL register). If either of these two signals goes low, then SR_CLR_I goes low. The serial data output (SR_SDO) can be selected from any one of the 18-bit register's outputs. Selection of the source is provided through software using the SR_CTL register. A common active low asynchronous reset (SR_CLR_I) is provided for the shift register and for the latch.

Operating Modes

This section describes the two operation modes used by the shift register.

Serial Data Output

The shift register outputs serial data on the SR_SDO pins based on the SR_SDO_SEL bits in the SR_CTL register. These bits select which serial data of the 18-bit stream are moved to the serial output. By default if all the SR_SDO_SEL bits are cleared, the LSB data is output. This mode is for useful if multiple SR registers need to be cascaded.

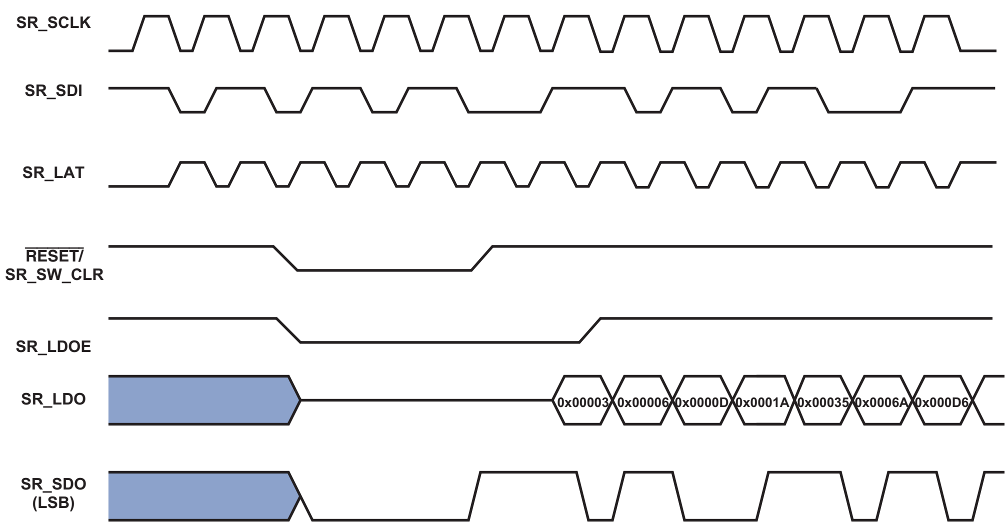

Figure 18-2. Shift Register Timing

Parallel Data Output

If the SR_LDOE bit in the SR_CTL register is set, the output stage of the parallel data latch is enabled. Data in the latch appears at the output whenever this bit is set.

The data in each flip-flop is transferred to the respective latch on a positive-going transition of the SR_LAT_I input. If both clocks are connected together, the shift register is always one clock pulse ahead of the latch.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

Shift Register Effect Latency

After the SR register is configured, the maximum effect latency is 2 PCLK cycles.

Programming Model

Since the SR_CTL, SRU_CLK_SHREG, and SRU_DAT_SHREG register signals come from the peripheral clock domain (PCLK) to the SR_SCLK_I and

SR_LAT_I domain, there are timing violations for one SR_SCLK_I period. To avoid this program the following registers in the order listed.

1. The SRU_CLK_SHREG, and SRU_DAT_SHREG registers.
2. The SR_CTL register.
3. Drive the SR_SCLK_I, SR_LAT_I, and SR_SDI_I input signals.
4. The SR_CTL, SRU_CLK_SHREG, and SRU_DAT_SHREG registers are in PCLK domain. There may be timing violations for signals crossing PCLK domain to the SR_SDCLK_I and SR_LAT_I domain. To avoid this first program SR_CTL, SRU_CLK_SHREG, and SRU_DAT_SHREG registers and then drive on SR_SDCLK_I, SR_LAT_I, and SR_SDI_I.

19 REAL-TIME CLOCK – ADSP-2147x/ADSP-2148x

The ADSP-2147x processors contain a real-time clock (RTC) which provides a set of digital watch features to the processor, including time of day, alarm, and stopwatch countdown. It is typically used to implement either a real-time watch or a life counter. The RTC specifications are shown in Table 19-1.

Table 19-1. RTC Specifications

Feature	**Availability**
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	No
SRU DAI Default Routing	N/A
SRU2 DPI Required	No
SRU2 DPI Default Routing	N/A
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	No
Transmission Simplex	N/A
Transmission Half Duplex	N/A
Transmission Full Duplex	N/A
Access Type	
Data Buffer	N/A

Table 19-1. RTC Specifications (Cont'd)

Feature	Availability
Core Data Access	N/A
DMA Data Access	N/A
DMA Channels	N/A
DMA Chaining	N/A
Boot Capable	No
Local Memory	No
Clock Operation	f_{PCLK}

Features

The RTC interface has the following features.

- Provides a 1 Hz clock with Second, Minute, Hour and Day Counter (0 to 32767 days).
- Alarm Feature available with time of day interrupt.
- Operates on a dedicated supply from an external 3.3 V battery.
- Stopwatch function available.
- Standard two pin interface with external 32.768 kHz crystal, 6 pF capacitor on each pin and 100 MΩ resistor between the pins.
- RTC Power switches to that of I/O Supply when chip is powered on, saving battery life.
- Calibration Feature available to correct time once a day; application can use `RTCXTALIN` pin to determine calibration settings.

Pin Descriptions

The pins used for the real-time clock are described in the ADSP-2147x and the ADSP-2148x SHARC Processor data sheets.

Clocking

The RTC timer has a 32.768 kHz crystal external to the processor. An internal clock divider scales the crystal clock down to a 1 Hz reference clock (RTCLKOUT pin) which triggers all RTC counters. The RTC interface is clocked internally with PCLK.

For clock power management refer to Chapter 23, Power Management.

Register Overview

This section provides basic information on the RTC control and status registers only. Complete bit information can be found at "Real-Time Clock Registers" on page A-116.

The RTC_CTL, RTC_STAT, and RTC_INITSTAT registers are user registers located in the RTC core voltage domain and can be directly accessed over the peripheral bus.

Control Register (RTC_CTL). Controls RTC functions and the RTC interrupt enable functions.

Status Register (RTC_STAT). Reports RTC status and interrupt status.

Initialization Status Register (RTC_INITSTAT). Reports the register status of the 1 Hz domain.

The RTC_CLOCK, RTC_ALARM, RTC_INIT, and RTC_STPWTCH are 1 Hz registers located in the RTC I/O voltage domain and are accessed indirectly by shadow registers in 1 Hz ticks. These register (except RTC_INIT) do not have default reset settings.

Clock Count Register (RTC_CLOCK). Used to read or write the current time and is updated every second.

Alarm Count Register (RTC_ALARM). Used to control the alarm functions.

Stopwatch Count Register (RTC_STPWTCH). Used to have the count-down value for the stopwatch function.

Initialization Register (RTC_INIT). Used for calibration purpose and for RTC power-down.

Functional Description

The RTC provides a set of digital watch features to the SHARC processor. It uses an external 32.768 kHz crystal with external capacitors and provides Second, Minute, Hour and Day counts along with an Alarm and Stopwatch feature. The RTC operates on a dedicated external 3.3 V Lithium coin cell which is never powered off. A block diagram of the RTC is shown in Figure 19-1.

Power Supply Partitioning

The RTC logic is partitioned between the processor core supply voltage and RTC I/O supply voltage. The RTC functions on a separate power island on the RTC supply voltage. When the core supply voltage is absent, interrupts are ignored.

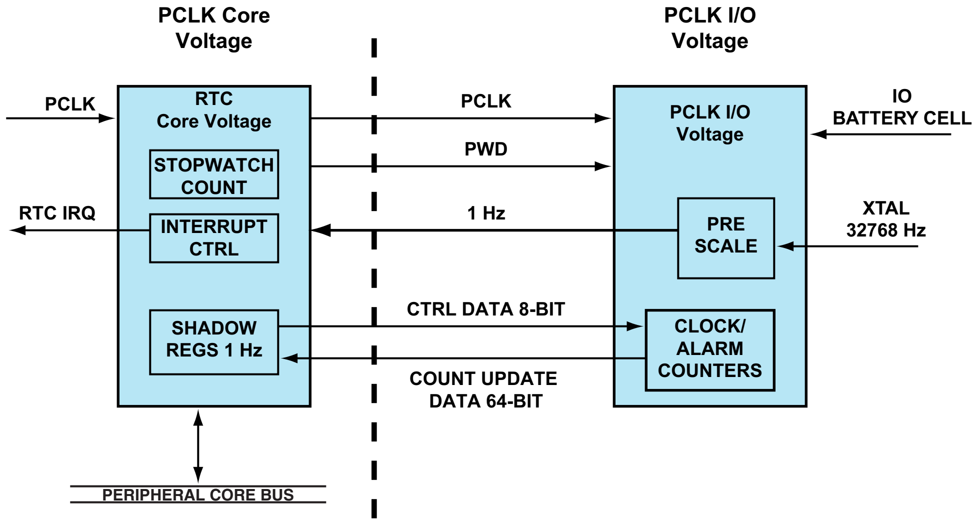

Figure 19-1. RTC Block Diagram

When the processor's I/O supply is above a certain threshold, the RTC switches to the I/O supply to conserve battery power.

Battery power supply can operate the RTC when the I/O voltage if turned off.

The RTC is partitioned into two blocks. The counting and clock function is provided in the I/O voltage domain (V_{DDEXT}), while the control and access function and the user interface are provided in the core voltage domain. The RTC I/O operates on a dedicated power supply provided by the external 3.3 V (nominal) lithium coin cell. The RTC also has the ability to switch to the I/O supply (V_{DDEXT}). The RTC core operates on the nominal core voltage supply (V_{DDINT}). The interface between both blocks is provided by a set of level shifters. The partitioning at chip level is shown in Figure 19-1.

Battery Life

To increase the battery life of the external 3.3V cell, maximum functionality is kept in the RTC core voltage which runs off the chip supply with only basic clock circuitry inside the RTC IO voltage. This means:

- The seconds, minutes, hours and days counters reside inside the RTC IO voltage.
- The alarm register and comparators reside inside the RTC IO voltage. This allows programs to power down the rest of the chip without the alarm being reset.
- The programmable interface registers, through which the application reads or writes the current time and alarm settings, are part of the RTC core voltage. In order to set the current time and/or alarm, software writes into the shadow registers in the RTC core voltage are performed. The data is then transferred into the corresponding register in the RTC IO voltage by hardware.
- The RTC core voltage runs primarily on the processor's peripheral clock while the RTC IO voltage runs primarily on a self generated 1 Hz clock. The synchronization circuitry sits inside the RTC core voltage.
- The stopwatch circuitry is inside the RTC core voltage and operates on a 1Hz clock, level shifted from the RTC IO voltage.

Digital Watch Counters

The primary function of the RTC is to maintain an accurate day count and time of day. The RTC accomplishes this by means of four counters:

- 60-second counter
- 60-minute counter
- 24-hour counter
- 32768-day counter

The RTC increments the 60-second counter once per second and increments the other three counters when appropriate. The 32768-day counter increments each day at midnight (23 hours, 59 minutes, 59 seconds). Interrupts can be issued periodically, either every second, every minute, every hour, or every day. Each of these interrupts can be independently controlled. The RTC block diagram is shown in Figure 19-2.

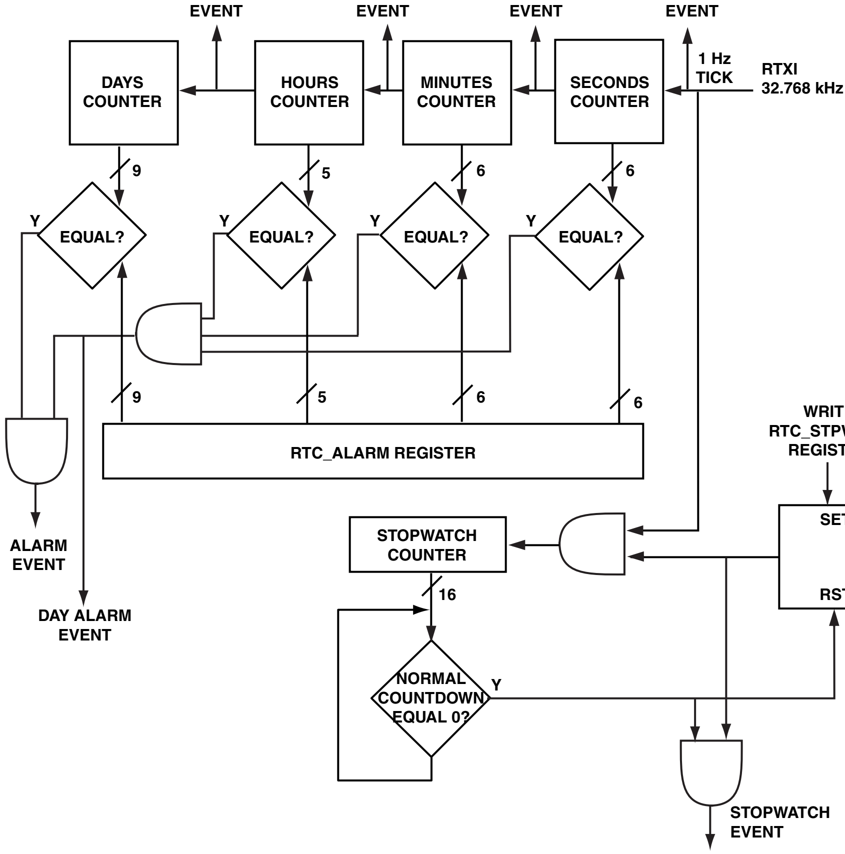

Figure 19-2. RTC Block Diagram

The RTC provides a set of digital watch features. The internal oscillator generates a 32768 Hz signal using the crystal which is scaled down to 1Hz and used to clock the second, minute, hour and day counters. The 32768 day counter increments each day at midnight (during the change from 23:59:59). The counter operates on the RTC supply (either the external battery or I/O supply) and is active irrespective of the status of the processor core supply (V_{DDINT}). When the processor core and I/O supply are valid:

- the current time is updated every second into the RTC clock register (RTC_CLOCK)
- interrupts can be issued periodically every second, every minute, every hour or every day

Each of the interrupts can be independently controlled, described in "Interrupts" on page 19-14.

It is the responsibility of the program to set the correct time by a software write into the RTC_CLOCK register. Once set, the counters maintain time as long as the RTC supply is valid.

Writes to the 1 Hz Registers

Writes to the RTC 1 Hz registers are synchronized to the 1 Hz RTC clock. When setting the time of day, do not factor in the delay when writing to the RTC 1 Hz registers. The most accurate method of setting the RTC is to monitor the seconds (1 Hz) event flag or to program an interrupt for this event and then write the current time to the RTC status register (RTC_STAT) in the interrupt service routine (ISR). The new value is inserted ahead of the incremented value. Hardware adds one second to the written value (with appropriate carries into minutes, hours and days) and loads the incremented value at the next 1 Hz tick, when it represents the then-current time.

Writes posted at any time are properly synchronized to the 1 Hz clock. Writes complete at the rising edge of the 1 Hz clock. A write posted just before the 1 Hz tick may not be completed until the 1 Hz tick one second later.

Reads From the 1 Hz Registers

There is no latency when reading 1 Hz registers, as the values come from the shadow registers. The shadows are updated and ready for reading by the time any RTC interrupts or event flags for that second are asserted. Once the internal core logic completes its initialization sequence after PCLK starts, there is no point in time when it is unsafe to read the 1 Hz for synchronization reasons. They always return coherent values, although the values may be unknown.

Operating Modes

The following sections provide information on the operating modes available to the real-time clock.

Alarm

The RTC provides two alarm features, programmed with the RTC alarm register (RTC_ALARM). The first is a time of day alarm (hour, minute, and second). When the alarm interrupt is enabled, the RTC generates an interrupt each day at the time specified. The second alarm feature allows the application to specify a day as well as a time. When the day alarm interrupt is enabled, the RTC generates an interrupt on the day and time specified. The alarm interrupt and day alarm interrupt can be enabled or disabled independently.

Day Alarm

The second alarm feature allows the application to specify a day as well as a time. When the day alarm interrupt is enabled, the RTC generates an interrupt on the day and time specified. The alarm interrupt and day alarm interrupt can be enabled or disabled independently.

Stopwatch

The RTC stopwatch count register (RTC_STPWTCH) contains the countdown value for the stopwatch. The stopwatch counts down seconds from the programmed value and generates an interrupt (if enabled) when the count reaches 0. The counter stops counting at this point and does not resume counting until a new value is written to RTC_STPWTCH. Once running, the counter may be overwritten with a new value. This allows the stopwatch to be used as a watchdog timer with a precision of one second.

The stopwatch can be programmed to any value between 0 and (2^{16} – 1) seconds, which is a range of 18 hours, 12 minutes, and 15 seconds. Typically, software should wait for a 1 Hz tick, then write the RTC_STPWTCH register. One second later, RTC_STPWTCH changes to the new value and begins decrementing. Because the register write occupies nearly one second, the time from writing a value of N until the stopwatch interrupt is nearly N + 1 seconds. To produce an exact delay, software can compensate by writing N – 1 to get a delay of nearly N seconds. This implies that a delay of 1 second with the stopwatch cannot be achieved. Writing a value of 1 immediately after a 1 Hz tick results in a stopwatch interrupt nearly two seconds later. To wait one second, software should just wait for the next 1 Hz tick.

Calibration for Accuracy

To guard against the possibility of long term (> 1 day) errors, the RTC provides a calibration feature using 4 bits of the RTC_INIT register (not available for the stopwatch function).

This is a simple a time correction at the end of every day (when the clock register changes from a Day:Hour:Min:Sec value of XXX:23:59:59 to YYY:00:00:00). It functions by adding or subtracting an integer number of seconds (maximum of 7) from the start of the next day, to correct accumulated time error over the course of the previous day. The number of seconds that are added or subtracted is defined in the RTC_INIT register, CALIB field.

As an example, if there is a –50 ppm error in the 1Hz frequency, this translates into 86400 × 50 ppm seconds (=+4.32 seconds) error at the end of the day. That is at 00:00:00, RTC time is 4.32 seconds ahead of actual time. The RTC can correct this by adding 4 seconds (if 4 is the value written into the calibration register) to the time at 00:00:00. Therefore, from 23:59:59, the timer counter jumps directly to 00:00:04, (there is no 00:00:00 to 00:00:03 time occurrences). At the instant it jumps to 00:00:04, the error reduces to +0.32 seconds over the course of the day, which is only 3.7 ppm.

As a second example, if there is a +50 ppm error in the 1Hz frequency, this also translates into 86400 × 50ppm seconds (= –4.32 seconds) error at the end of the day which in this case the time has to be subtracted. This is corrected in the RTC by counting 00:00:00 to 00:00:03 twice, so that the time is effectively subtracted. As soon as the RTC reaches 00:00:04, the error reduces to –0.32 seconds over the course of the previous day and accumulated error is minimized.

When the RTC is powered up for the first time, the calibration values are written once to ensure proper functionality. (If they are not to be used, write 0000). These register bits are sticky, which means that once set, they retain their value irrespective of the status of core supply.

The addition or subtraction of time can only be in integer multiples of seconds. Zero to seven seconds can be added or subtracted using 4 bits. The MSB indicates addition (0) or subtraction (1). The three LSBs indicate number of seconds (0 – 7 represented by their binary 3 bit equivalents). Because the clock runs at a time period of one second (@ 1 Hz frequency) 0.25 or 0.5 second resolution is not possible.

The calibration technique introduces a guard band for alarm by definition. In case the alarm is set within the duration of the time (Day:00:00:00 to Day:00:00:06) corrected by the calibration register, then it occurs at the nearest corrected time. This is shown in the following two examples.

If the CALIB value in the RTC_INIT register is 0101, the RTC clock jumps from 23:59:59 to 00:00:05 due to calibration. If the alarm is set to 00:00:01, it occurs at the RTC time 00:00:05.

If the CALIB value in the RTC_INIT register is 1101, then the RTC clock counts from 00:00:00 to 00:00:04 twice and then moves to 00:00:05. If the alarm is set to 00:00:01, it occurs at the RTC time 00:00:05.

For calibration purposes the RTCLKOUT pin drives the 1 Hz clock.

Accuracy

In order to perform calibration on the bench, use the RTXI pin and check the ppm deviation from 32.768 kHz. This ppm error is the same as in the internal 1Hz clock (RTCLKOUT pin) and the calibration register should be updated with the corresponding values as explained in "Calibration for Accuracy" above.

Note that total accuracy is <= +/– 35 ppm, <+/–1.5 minutes per month of error, inclusive of any inaccuracies of the RTC input crystal at room temperature. This is achieved with a crystal of +/– 10 ppm error at 25°C. A crystal error of +/–20 ppm translates into a maximum inaccuracy of +/–45 ppm.

Interrupts

The RTC has one interrupt that is programmable through the programmable interrupt priority control register (see Appendix 2, Interrupt Control). The RTCI source bit is used to connect the RTC interrupt to the peripheral interrupt inputs of the core. Table 19-2 provides and overview of RTC interrupts.

Table 19-2. RTC Interrupt Overview

Default Programmable Interrupt	Sources	Masking	Service
RTC not connected by default	Second Minute Hour Day Alarm Daily alarm 1Hz clock fail	RTC_CTL	Read to clear RTC_STAT + RTI instruction

Sources

The RTC module generates in total 8 local interrupts which are grouped into 4 counter status, 2 alarm status and 2 system status interrupts. All eight signals are logically ORed into 1 RTC interrupt signal which must be routed into a programmable interrupt. The RTC port can generate interrupts under the conditions described in the following sections.

RTC Counter

These conditions are based on the RTC counter.

- Per second
- Per minute
- Per hour

- Per day
- Alarm
- Daily alarm

1 Hz Register Write Completion

Completion of a write to any 1 Hz registers (RTC_CLOCK, RTC_INIT, RTC_ALARM and RTC_STPWTCH).

Emulation

The module allows programs to enable/disable interrupts for debug purposes.

1 Hz Clock Errors

The module generates interrupts on 1 Hz clock failures.

Masking

The RTCI signal is not routed by default to programmable interrupts.To service the RTCI, unmask (set = 1) any programmable interrupt bit in the IMASK/LIRPTL register.

Interrupts can be individually unmasked using the RTC interrupt control register (RTC_CTL). To identify clock errors in the 1Hz domain, programs need to unmask the CKFAIL_INTEN bit in the RTC_CTL register.

Service

In the service routine the RTC_STAT register should be read to identify the cause of the interrupt. While reading the status register the RTC automatically clears the respective status bit ensuring that the cause has been cleared before ending the routine.

Note that the pending RTC interrupt is cleared whenever all enabled and set bits in the RTC_STAT register are read, or when all bits in the RTC_CTL register corresponding to pending events are cleared.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

Real-Time Clock Effect Latency

After the RTC registers are configured the effect latency is 2 PCLK cycles.

Programming Model

The following sections provide basic programming steps for the RTC interface.

1 Hz Register Write Latency

Writes of the alarm, clock, stopwatch and initialization registers is performed in a two step sequence:

1. The desired values are programmed into a shadow register in the processor's core voltage domain and operating on the its peripheral clock (PCLK).
2. The contents of the shadow register are synchronized onto the contents of the RTC's internal clock register which operates on the 1 Hz clock in the I/O voltage domain.

To ensure that writes between the core voltage and RTC voltage domain are properly synchronized, all write commands should be issued immediately after a seconds' event in the RTC_STAT register.

This two step sequence results in a write latency of up to 1 second. While the write sequence is ongoing, the write pending (WR_PEND) bit is set in the RTC_STAT register and is cleared by hardware when the process is complete. Resetting or powering down the peripherals while a write is in progress, (WR_PEND bit is set) is forbidden. Subsequent writes to the same register before completion of the previous write are ignored.

Do not attempt write to the RTC_CLOCK, RTC_ALARM or RTC_SWTCH registers when the RTC oscillator is powered down or when the RTC_BUSDIS bit is set.

During initialization, after a write of the RTC_INIT register, make sure that the WR_PEND bit is cleared before attempting writes to other registers.

Power-Up, Power-Down and Reset

A programmable register bit is provided to power down the RTC. Here power-down is interpreted as a crystal oscillator disable, which would reduce power dissipation to only leakage current. Once set or reset, this bit retains its value unless changed, irrespective of the status of core supply.

The inclusion of the power-down bit (RTC_PDN) as well as the possibility that the RTC may not be used in certain applications introduces specific constraints on the power-up and reset behavior of the RTC. These are described below.

1. When the RTC is powered-up for the first time, it remains in an undefined state until the core powers-up and the corresponding power-down bit in the RTC_INIT register is written by software. Programs should clear RTC_PDN if the RTC function is desired and set if it is not.

2. After clearing the RTC_PDN bit the application has to wait at least until the first seconds' event before it writes the timer and alarm registers. This is because the oscillator has a startup time before the clock is generated.

   This sequence applies only to the first time the RTC supply (battery or I/O) is connected. Once the RTCPDN bit is set or reset, its value is retained as long as RTC supply (battery or I/O) is valid.

3. After the RTC supply is connected for the first time and the RTC_PDN bit has been cleared, the application is free to power-up and power -down the core supply any number of times without loss of RTC functionality (provided the RTC supply—battery or I/O, is valid). Conversely, if the RTC_PDN bit has been set, then the RTC oscillator remains disabled irrespective of the status of the core supply.

4. The current status of the RTC power-down is updated by hardware into the initialization status register (RTC_INITSTAT) register. This is useful when the rest of the processor wakes up from power-down and needs to know the status of the RTC.

5. Whenever the processor core wakes up from power-down, the values of the RTC_CLOCK, RTC_ALARM and RTC_SWTCH registers is zero until the first seconds' event after power-up. At the first seconds' event, an arbitrary value is uploaded into these registers. To put them in a defined state software must write the desired value into these registers.In case the RTC_CLOCK and RTC_ALARM have been set before core power-down and subsequent power-up, their values are valid throughout, but can be read by the program only after the first seconds' event after power-up.

Status Flags

The unknown values in the registers at power up can cause event flags to set before the correct value is written into each of the registers. By catching the 1 Hz clock edge, the write to RTC_CLOCK can occur a full second before the write to RTC_ALARM. This would cause an extra second of delay between the validity of RTC_CLOCK and RTC_ALARM, if the value of the RTC_ALARM out of reset is the same as the value written to RTC_CLOCK. Wait for the writes to complete on these registers before using the flags and interrupts associated with their values.

The following is a list of flags along with the conditions under which they are valid:

- Seconds (1 Hz) Event flag – Always set on the positive edge of the 1 Hz clock and after shadow registers have updated after waking from power-down. This is valid as long as the RTC 1 Hz clock is running. Use this flag or interrupt to validate the other flags.

- Write Complete – always valid.

- Write Pending Status – always valid.
- Minutes Event flag – valid only after the second field in RTC_STAT is valid.
- Hours Event flag – valid only after the minute field in RTC_STAT is valid.
- 24 Hours Event flag – valid only after the hour field in RTC_STAT is valid.
- Stopwatch Event flag – valid only after the RTC_SWCNT register is valid.
- Alarm Event flag – valid only after the RTC_STAT and RTC_ALARM registers are valid.
- Day Alarm Event flag – same as alarm.

Writes posted together at the beginning of the same second take effect together at the next 1 Hz tick. The following sequence is safe and does not result in any spurious interrupts from a previous state.

1. Wait for 1 Hz tick.
2. Write new values for RTC_CLOCK, RTC_ALARM, and/or RTC_SWCNT.
3. Write 1s to clear the RTC_CLOCK flags for Alarm, Day Alarm, Stopwatch, and/or per-interval.
4. Write new value for RTC_CTL with Alarm, Day Alarm, Stopwatch, and/or per-interval interrupts enabled.
5. Wait for 1 Hz tick.
6. New values have taken effect simultaneously.

Debug Features

The following section provides information on debugging features available with the real time clock (RTC).

Emulation Considerations

An emulation halt can optionally mask all RTC interrupts by setting the EMU_INTDIS bit in RTC_CTL register.

20 WATCHDOG TIMER – ADSP-2147x, ADSP-2148x

The ADSP-2147x and ADSP-2148x processors include a 32-bit watchdog timer (WDT) that can be used to implement a software watchdog function. The timer can improve system reliability by forcing the processor to a known state through generation of a system reset if the timer expires before being reloaded by software. The WDT specifications are shown in Table 20-1.

Table 20-1. Watchdog Timer Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	No
SRU DPI Required	No
SRU DPI Default Routing	No
Interrupt Control	No
Protocol	
Master Capable	N/A
Slave Capable	N/A
Transmission Simplex	N/A
Transmission Half Duplex	N/A
Transmission Full Duplex	N/A
Access Type	
Data Buffer	No
Core Data Access	N/A

Table 20-1. Watchdog Timer Specifications (Cont'd)

Feature	Availability
DMA Data Access	N/A
DMA Channels	N/A
DMA Chaining	N/A
Boot Capable	N/A
Local Memory	N/A
Clock Operation	WDTCLKIN

Features

The following list provides a brief description of the watchdog timer's features.

- Programmable time out period – with about 1 second with 12 MHz clock.
- Time out resets the DSP and asserts the external reset ($\overline{\texttt{WDTRSTO}}$ pin). DSP is reset internally to the chip upon WDT time out.
- WDT has its own clock (`WDT_CLKIN`) that is independent from the SHARC `CLKIN` and any other clock derived from `CLKIN`.
- An internal oscillator to provide the clock input. This internal oscillator provides a 2 MHz (typical frequency) clock.
- Status bit available for the processor to read which is cleared on hardware reset assertion – it is not cleared on WDT generated reset.

- Programmable trip counter which allows programs to set the number of times the WDT can expire before the $\overline{\text{WDTRSTO}}$ signal is asserted continuously.
- WDT space is locked and can be accessed only after unlocking the space using commands.

Pin Descriptions

The pins used for the watchdog timer are described in the ADSP-2147x and ADSP-2148x data sheets.

Register Overview

The following sections provide brief descriptions of the primary registers used to program the timers. For more information, see “Peripheral Timer Registers” on page A-264.

Control Register (WDTCTL). The control register is a 32-bit system memory-mapped register used to configure the watchdog timer. Any writes made by the Software to the Register will keep it enabled. Only an External Hardware Reset can disable WDT.

Count Register (WDTCNT). Holds the 32-bit unsigned count value. The WDTCNT register must always be accessed with 32-bit read/writes.

Current Count Status Register (WDTCURCNT). Contains the current count value of the watchdog timer. Reads to WDTCURCNT return the current count value.

Status Register (WDTSTATUS). Contains the watchdog timer status information. This register is not cleared by the WDT generated reset.

Trip Register (WDTTRIP). Sets the number of times that the WDT can expire before the $\overline{\text{WDTRSTO}}$ pin is continually asserted until the next time hardware reset is applied.

Unlock Register (WDTUNLOCK). Protects the WDT configuration space (WDTCTL, WDTCNT, WDTCURCNT and WDTTRIP registers) against accidental writes from the processor core.

Clock Select Register (WDTCLKSEL). Selects one of the 2 clock sources for WDTCLK, an external source (external clock connected to WDT_CLKIN or a ceramic resonator connected between WDT_CLKIN and WDT_CLKO) or from internal oscillator. For more information, see "Clocking" on page 20-4.

Internal oscillator is only supported on ADSP-2147x processor models.

Clocking

The WDT provides three options for the clock source.

1. An external clock source can be provided on WDT_CLKIN pin.
2. A ceramic resonator connected between the WDT_CLKIN pin and the WDT_CLKO pin combined with internal circuitry to generate a clock. The ceramic resonator should be either a CERALOCK CST-CR4M00G53-R0 (4 MHz) or CERALOCK CSTCC2M00G56-R0 (2 MHz).
3. An internal RC oscillator. This internal RC oscillator provides a 2 MHz (typical frequency) clock. This feature is only applicable for ADSP-2147x processors.

Functional Description

The watchdog timer is used to supervise the stability of the system software. Software initializes the 32-bit count value of the timer, and 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 reload the timer, has stopped running due to an external event or software error. When used in this way, software reloads the watchdog timer in a regular manner so that the downward counting timer never expires. An expiring timer then indicates that system software might be out of control.

The watchdog timer resets both the core and the internal peripherals. After an external reset, the WDT must be disabled by default. Software must be able to determine if the watchdog was the source of the hardware reset by interrogating a status bit in the watchdog timer control register.

As shown in Figure 20-1 the clock source for the watchdog timer can be selected from either the internal RC oscillator or from an external oscillator.

ⓘ The expired timer performs a software reset (reset core and the peripherals). For more information, see "Processor Reset" on page 24-3.

After an external reset, the WDT must be disabled by default. Software must be able to determine if the watchdog was the source of the hardware reset by interrogating a status bit in the watchdog timer control register.

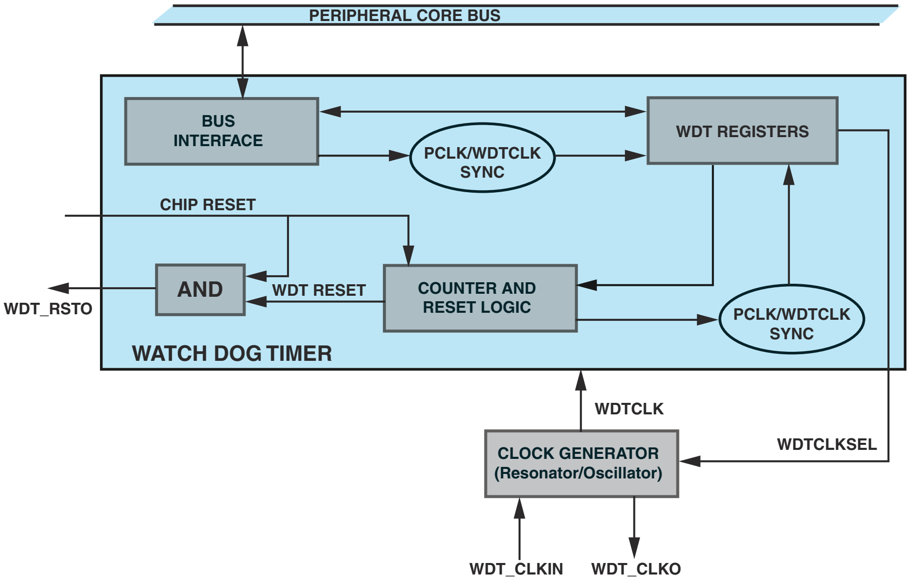

Figure 20-1. Watchdog Timer Block Diagram

Operating Mode

The WDT operates in trip count mode as described below.

Trip Count

The WDT contains a software programmable trip counter register that sets the number of times that the timer can expire before the `WDTRSTO` pin is continually asserted (until the next time hardware reset is applied). The trip counter is not cleared by the WDT generated reset. This gives software the ability to count the number of WDT generated resets using the `CURTRIPVAL` bits in the `WDTTRIP` register.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

Watchdog Timers Effect Latency

After the WDT registers are configured the effect latency is 2 PCLK cycles enable and 2 PCLK cycles disable.

Programming Model

If enabled, the 32-bit watchdog timer counts downward every WDT_CLKIN cycle. When it becomes 0, the system is reset. The counter value can be read through the 32-bit WDTCURCNT register. The WDTCURCNT register cannot, however, be written directly. Rather, software writes the watchdog period value into the 32-bit WDTCNT register before the watchdog is enabled. Once the watchdog is started, the period value cannot be altered.

To start the watchdog timer:

1. Unlock the WDT configuration registers by writing the unlock "command" value to the WDTUNLOCK register. Select the WDTCLK by programming the WDTCLKSEL bit.

   By default, the resonator output is WDTCLK.

2. Set the trip counter value for the watchdog timer by writing to the WDTTRIP register.

3. Set the count value for the watchdog timer by writing the count value into the watchdog period register (WDTCNT). Since the watchdog timer is not yet enabled, the write to the WDTCNT registers automatically pre-loads the WDTCURCNT register as well. Note that sufficient time must be provided for the write to the WDTCURCNT register to occur (2.5 WDTCLK cycles max.), before enabling WDT.

4. Enable the watchdog timer in WDTCTL.

5. Lock the WDT configuration registers by writing to the WDTUNLOCK register. The watchdog timer begins counting down, decrementing the value in the WDTCURCNT register every WDT_CLKIN cycle. If software does not serve the watchdog in time, WDTCURCNT continues decrementing until it reaches 0 and the system is reset.

   The counter now reloads the WDTCURCNT value from WDTCNT and keeps decrementing. Additionally, the WDRO latch bit in the WDTSTATUS register is set and can be interrogated by software. This occurs up to the number of times programmed in the WDTTRIP register. When WDTTRIP expires, WDT holds the $\overline{\text{WDTRSTO}}$ asserted.

6. To prevent the watchdog from expiring, software serves the watchdog by unlocking the WDT configuration space and performing dummy writes to the WDTCURCNT register address in time. The values written are ignored, but the write command cause the WDTCURCNT register to be reloaded from the WDTCNT register. If the watchdog is enabled with a zero value loaded to the counter, WDT expires immediately and resets the system. The WDRO bit of the watchdog control register is also set.

Debug Features

The following section provides information on debugging features available with the watchdog timer.

Emulation Considerations

An emulation halt stops the WDT counter. The WDT resumes counting after being released from emulation halt. Single stepping is not supported for WDT in emulation mode.

21 UART PORT CONTROLLER

The universal asynchronous receiver/transmitter (UART) is a full-duplex peripheral compatible with the PC-style, industry-standard UART. The interface specifications are shown in Table 21-1.

Table 21-1. UART Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	No
SRU DAI Default Routing	N/A
SRU2 DPI Required	Yes
SRU2 DPI Default Routing	Yes
Interrupt Control	Yes
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

Table 21-1. UART Specifications (Cont'd)

Feature	Availability
DMA Channels	2
DMA Chaining	Yes
Boot Capable	No
Local Memory	No
Max Clock Operation	$f_{PCLK}/16$

Features

The UART converts data between serial and parallel formats. The serial format follows an asynchronous protocol that supports various word lengths, stop bits, and parity generation options. The UART primary features are listed below Figure 21-1 on page 21-6 shows the functional block.

- Compatible with the RS-232 and RS-485 Standards
- Data packing support for efficient memory usage
- Full duplex DMA operation
- Multiprocessor communication using 9 bit addressing
- Autobaud detection support
- The UART includes interrupt handling hardware

The UARTs do not support MODEM functionality.

SRU Programming

The SRU (signal routing unit) needs to be programmed in order to connect the UART signals to the output pins or connect the output of the transmitter to the receiver. The UART signals need to be routed as shown in Table 21-2.

Table 21-2. UART SRU2 Signal Connections

UART0 Source	DPI Connection	UART0 Destination
UART0_TX_O	Group A	UART0_RX_I
UART0_TX_O	Group B	
UART0_TX_PBEN_O	Group C	

Register Overview

The processor provides a set of PC-style, industry-standard control and status registers for the UART. These memory-mapped IOP registers are byte-wide registers that are mapped as half-words with the most significant byte zero-filled.

Transmit Control Register (UART0TXCTL). Global control for TX core or DMA operation.

Receive Control Register (UART0RXCTL). Global control for RX core or DMA operation.

Line Control Register (UART0LCR). Controls the format of the data character frames. It selects word length, number of stop bits and parity.

Divisor Latch High/Low Register (UART0DLL/UART0DLH). Characterize the UART bit rate. The divisor is split into the divisor latch low byte (`UART0DLL`) and the divisor latch high byte (`UART0DLH`).

Mode Control Register (UART0MODE). Controls packing and address modes.

Interrupt Enable Control Register (UART0IER). Enables interrupt requests from system handling.

Line Status Register (UART0LSR). Returns status of controls format of the data character frames as overrun or framing errors and break interrupts.

Transmit Status Register (UART0TXSTAT). Returns status of DMA operation.

Receive Status Register (UART0RXSTAT). Returns status of DMA operation and Rx errors.

Interrupt Identification Status Register (UART0IIR). Provides status of all interrupts and combines them into one channel.

Clocking

The fundamental timing clock of the UART module is peripheral clock/16 (PCLK/16).

The bit rate is characterized by the peripheral clock (PCLK) and the 16-bit divisor. The divisor is split into the UART divisor latch low byte register (UARTDLL) and the UART divisor latch high byte register (UARTDLH). These registers form a 16-bit divisor. The baud clock is divided by 16 so that:

- Divisor = 1 when UARTDLL = 1 UARTDLH = 0
- Divisor = 65,535 when UARTDLL = UARTDLH = FF

The 16-bit divisor formed by the UARTDLH and UARTDLL registers resets to 0x0001, resulting in the highest possible clock frequency by default. The UARTDLH and UARTDLL registers can be programmed by software before or after turning on the clock.

Table 21-3 provides example divide factors required to support most standard baud rates.

Table 21-3. UART Baud Rate Examples With 100 MHz PCLK

Baud Rate	Divisor Latch (DL)	Actual	% Error
2400	2604	2400.15	0.006
4800	1302	4800.31	0.007
9600	651	9600.61	0.006
19200	326	19,171.78	0.147
38400	163	38,343.56	0.147
57600	109	57,339.45	0.452
115200	54	115,740.74	0.469
921,600	7	892,857.14	3.119
6,250,000	1	6,250,000	–

Careful selection of PCLK frequencies, that is, even multiples of desired baud rates, can result in lower error percentages.

The clock to this module may be shut off for power savings. For more information, see “Disabling Peripheral Clocks” on page 23-11.

Functional Description

The UART supports multiprocessor communication using 9-bit address detection. This allows the units to be used in multi-drop networks using the RS-485 data interface standard. The UART has its own set of control and status registers (Figure 21-1).

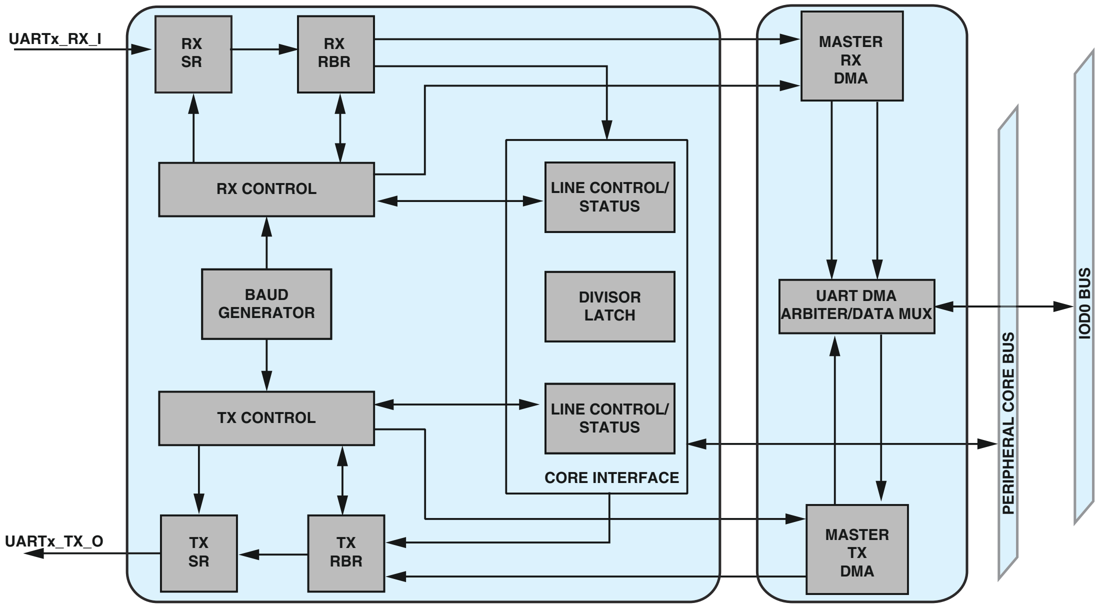

Figure 21-1. UART Functional Block Diagram

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 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. For more information, see "DMA Transfers" on page 21-14.

Either one of the peripheral timers can be used to provide a hardware-assisted autobaud detection mechanism for use with the UART. See "Autobaud Detection" on page 21-21.

Serial Communication

The UART follows an asynchronous serial communication protocol with these options:

- 5 – 8 data bits
- 1 or 2 stop bits
- None, even, or odd parity
- Baud rate = PCLK/(16 × divisor), divisor value can be from 1 to 65,536

All data words require a start bit and at least one stop bit. With the optional parity bit, this creates a 7 to 12-bit range for each word. The format of received and transmitted character frames is controlled by the line control register (UARTLCR). Data is always transmitted and received least significant bit (LSB) first.

Figure 21-2 shows a typical physical bit stream measured on the transmit pin.

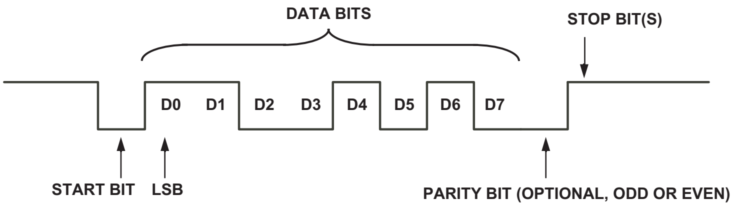

Figure 21-2. Bit Stream on the Transmit Pin

Transmit and receive channels are both buffered. The UARTTHR register buffers the transmit shift register (UARTTSR) and the UARTRBR register buffers the receive shift register (UARTRSR). The shift registers are not directly accessible by software.

Operating Modes

The packed and unpacked UART operation modes are described in the following sections.

Data Packing/Unpacking

The UART provides packed and unpacked modes of data transfer to and from the internal memory of the processors. This mode is set using the UARTPACK bit (bit 0) in the UARTMODE register. In unpacked mode, the data word is appended to the left with 24 zeros during transmission or reception. In packed mode, two words of data are transmitted or received with their corresponding higher bytes filled with zeros. For example, consecutive data words 0xAB and 0xCD are packed as 0x00CD 00AB in the receiver, and 0x00CD 00AB is transmitted as two words of 0xAB and 0xCD successively from the transmitter. Packing is available in both I/O and DMA modes. A control bit, UARTPKSYN, can be used to re synchronize the packing. For information on using the UART for DMA transfers, see "DMA Transfers" on page 21-14.

The packed feature is provided to use the internal memory of the processor in a more efficient manner.

Note that in packed mode, both the transmitter and receiver operate with an even number of words. A transmit-buffer-empty or receive-buffer-full interrupt is generated only after an even number of words are transferred.

Programs must use care when using the packing feature in 9-bit mode.

Data Transfer Types

The UART is capable of transferring data using both the core and DMA. Note that data packing is available using both data transfer types. For more information, see "Data Packing/Unpacking" on page 21-8.

Serial Shift Registers

The UART contains two serial shift registers described below.

Output Shift Register

The data is moved to the internal transmit shift register (UARTTSR) where it is shifted out at a baud rate equal to PCLK/(16 × Divisor) with start, stop, and parity bits appended as required.

Input Shift Register

The number of stop bits is always assumed to be 1. After detection of the start bit, the received word is shifted into the receive shift register (UARTRSR) at a baud rate of PCLK/(16 × Divisor). After the appropriate number of bits (including stop bit) is received, the data are updated and the UARTRSR register is transferred to the UART receive buffer register (UARTRBR).

Buffers

The UART contains a single data buffer register for transmission and reception. These buffers are described in the following sections.

Transmit Buffer

A write to the UART transmit holding register (UARTTHR) initiates the transmit operation. All data words begin with a 1-to-0 transition start bit. This 32-bit write only register uses only 18 bits. The other bits are filled with zeros during writes. In no-pack mode (default), only the lower byte is used—all other bits are zero filled. Note that data is transmitted and received by the least significant bit (LSB) first followed by the most significant bits (MSBs).

Receive Buffer

The receive operation uses the same data format as the transmit configuration, except that shown in Figure 21-4. After the transfer of the received word to the UARTRBR buffer and the appropriate synchronization delay, the data ready status flag (UARTDR) is updated.

A sampling clock equal to 16 times the baud rate samples the data as close to the midpoint of the bit as possible. Because the internal sample clock may not exactly match the asynchronous receive data rate, the sampling point drifts from the center of each bit. The sampling point is synchronized again with each start bit, so the error accumulates only over the length of a single word. A receive filter removes spurious pulses of less than two times the sampling clock period.

> Because of the destructive nature of reading this register, a shadow register is provided for reading the contents of the corresponding main register. For more information, see "Debug Features" on page 21-24.

Buffer Status

The transfer of data from the UARTTHR register to the transmit shift register sets the transmit holding register empty status flag (UARTTHRE) in the UART line status register (UARTLSR).

After the appropriate number of bits (including stop bit) is received, the status is updated in the UART line status register (UARTLSR).

Buffer Packing

In packing mode, both the high and low bytes are used (Figure 21-3 on page 21-11). This mode is set using the UARTPACK bit in the UARTMODE register. In unpacked mode, the data word is appended to the left with 24 zeros during transmission or reception.

In packed mode, two words of data are transmitted or received with their corresponding higher bytes filled with zeros. For example, consecutive data words 0xAB and 0xCD are packed as 0x00CD 00AB in the receiver, and 0x00CD 00AB is transmitted as two words of 0xAB and 0xCD successively from the transmitter.

Packing is available in both core and DMA modes. A control bit, UARTPKSYN, can be used to re synchronize the packing. For information on using the UART for DMA transfers, see "DMA Transfers" on page 21-14.

The packed feature is provided to use the internal memory of the processor in a more efficient manner. In packed mode, both the transmitter and receiver operate with an even number of words. A transmit-buffer-empty or receive-buffer-full interrupt is generated only after an even number of words are transferred.

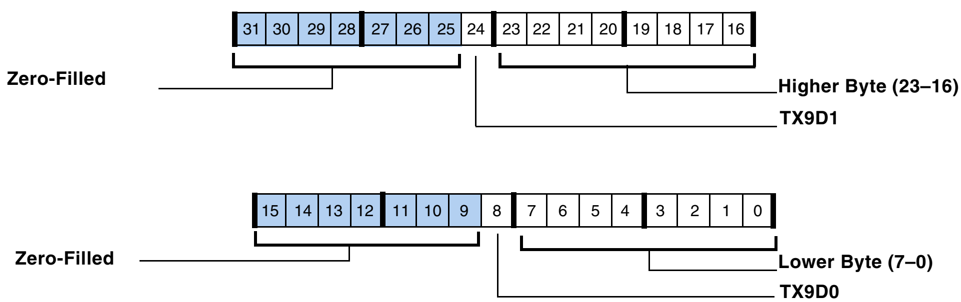

Figure 21-3. UART0 Transmit Holding Register (Packing Enabled)

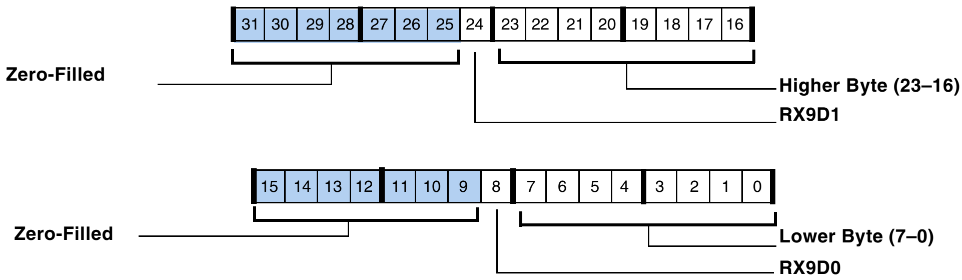

Figure 21-4. UART0 Receive Buffer Register

9-Bit Transmission Mode

The TX9D bits are the ninth bit in 9-bit transmission mode. A write to the UART transmit holding register (UARTTHR) initiates the transmit operation and reads from this address return the UARTRBR register. To select 9-bit transmission mode in the transmitter, set the UARTTX9 bit in the UART0MODE register. The 9-bit data (TX9D + 1 byte) can be directly written to the UART0THR buffer—either in packed or unpacked format. The UART transmitter transmits the TX9D bit instead of the parity bit. During 9-bit transmission mode, the parity select controls and the word length select do not have any effect.

For the receiver, set the UARTRX9 bit in the UART0MODE register. Set the address enable bit (UARTAEN) to enable address detection. If the received ninth bit is high, the received word is shifted from the RSR register to the UART_RBR buffer which generates an address detection interrupt. Read the UART0RBR buffer to find out if the device is being addressed. If the device is being addressed, the address enable (UARTAEN bit) should be cleared to allow further data and address bytes to be read into the receive buffer.

In 9-bit mode, the address detect interrupt can be generated whenever the receiver gets an address word, irrespective of the packing mode. This helps programs respond to an address word immediately. The program is expected to take into account these features when using packed mode.

During the 9-bit transmission mode parity has to be calculated in software to detect errors. The reception may be stopped when the receiver receives another address which is different from its own.

Programs must use care when using the packing feature in 9-bit transmission mode.

Flushing the Buffer

The UART receive and transmit buffers are flushed by disabling the UART receive and transmit ports.

Core Transfers

Core transfers move data to and from the UART by the processor core. To transmit a character, load it into the UART0THR register. Received data can be read from the UARTRBR register. The processor must write and read one character at time.

To prevent any loss of data and misalignments of the serial data stream, the UART line status register (UARTLSR) provides two status flags for handshaking—UART0THRE and UARTDR.

The UART0THRE flag is set when the UART0THR register is ready for new data and cleared when the processor loads new data into the UART0THR register. Writing this register when it is not empty overwrites the register with the new value and the previous character is never transmitted.

The UARTDR flag signals when new data is available in the UARTRBR register. This flag is cleared automatically when the processor reads from this register. Reading the UARTRBR register when it is not full returns the previously received value. When the UARTRBR register is not read in time, newly received data overwrites the UARTRBR register and the overrun (UART0E) flag is set.

With interrupts disabled, these status flags can be polled to determine when data is ready to move. Note that because polling can be processor-intensive, it is not typically used in real-time signal processing environments.

DMA Transfers

The UART interface support both standard and chained DMA. However, unlike the serial ports, programs cannot insert a TCB in an active chain using the UART.

In the UART, 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 normal DMA or DMA chaining. Software can write up to two words into the UART0THR register before enabling the UART clock. As soon as the UART DMA engine is enabled, those two words are sent. See also "Functional Description" on page 3-23.

To perform DMA transfers, the UART has a special set of receive and transmit registers. These registers are listed in "Standard DMA Parameter Registers" on page 3-4.

No additional buffering is provided in the UART DMA channel, so the latency requirements are the same as core transfers. However, the latency is determined by the bus activity and arbitration mechanism and not by the processor loading and interrupt priorities.

DMA through the UART is started by setting up values in the DMA parameter registers and then writing to the transmit and receive control registers, enabling the module using the UARTEN bits (in the UART0TXCTL and UARTRXCTL registers) and enabling DMA using the UARTDEN bits. A DMA can be interrupted by resetting the UARTDEN bit in the control register. A DMA request that is already in the pipeline completes normally.

UART DMA Group Priority

The UART module has two DMA channels, one for reads and the second for writes. The two channels are grouped together. When both channels have data ready, the read channel always wins with fixed priority (which is the first arbitration stage). The winning channel requests the DMA bus arbiter to get control of the peripheral DMA bus (2nd stage of arbitration).

The I/O processor considers the two DMA channels as a single group and therefore one arbitration request. For more information, see "Peripheral DMA Arbitration" on page 3-36.

DMA Chaining

DMA chaining is enabled by setting the UARTCHEN bit in the transmit and receive control registers. When chaining is enabled at the end of a current DMA, the next set of DMA parameters are loaded from internal memory and a new DMA starts. The index of the memory location is written in the chain pointer register. DMA parameter values reside in consecutive memory locations as shown in Table 3-17 on page 3-16. Chaining ends when the chain pointer register contains address 0x00000 for the next parameter block.

Interrupts

Table 21-4 provides an overview of UART interrupts.

Table 21-4. Overview of UART Interrupts

Default Programmable Interrupt	Sources	Masking	Service
DPII = P14I	DMA complete Core buffer service Address detection	DPI_IMASK_RE UART0IER	RTI instruction
	RX overrun error RX parity error RX frame error		ROC from UART0LSR + RTI instruction DMA: ROC from UART0RX-STAT + RTI instruction
Separate UART0RXI and UART0TXI not connected by default		UART0IER	

Sources

The UART0 module generates 10 local interrupts which are grouped into five system status, four error status and on3 transmit DMA interrupt. Nine signals are logically ORed into one `UART0RXI` output interrupt signal which is routed by default as a DPI interrupt into the core IVT address map. The transmit DMA signal is routed into the `UART0TXI` interrupt which is also routed by default as a DPI interrupt into the core IVT address map.

The UART has two interrupt outputs referred to as the `UART0RXI` and `UART0TXI` interrupts. In core mode the buffer service and line status are mapped into the single `UART0RXI` interrupt.

Separate interrupt lines are provided for error signals. Line error handling can be configured completely independently from the receive/transmit setup. The UART port can generate interrupts as described in the following sections.

Core Buffer Service Request

When DMA is disabled the processor core may read from the UARTTHR buffer or write to the UARTTHR buffer. An interrupt is generated when the receive buffer is full (UARTRBFIE) or transmit buffer is empty (UARTTBEIE). A transmitter empty interrupt is generated if both the TSR + THR registers (UARTTXFIE) are empty.

Address Detection

Generate a receive interrupt when an address is detected in 9-bit mode (UARTADI).

DMA Complete

For DMA, the transmit interrupt is generated when a DMA in transmit mode is complete whereas the receive interrupt is generated when a receive DMA is complete or when a receive error occurs. For information on using the UART for DMA transfers, see "DMA Transfers" on page 21-14, and Appendix 2, "Interrupt Control".

Chained DMA

For chained DMA, if the PCI bit is cleared (= 0), the DMA complete interrupt is generated only after the entire chained DMA access is complete. If the PCI bit is set (= 1), then a DMA interrupt is generated for each TCB.

Line Status Error

The receive error interrupt is generated for the following cases.

- On a receive overrun error
- On a receive parity error
- On a receive framing error
- On a break error interrupt

Masking

The DPI signal is routed by default to a programmable interrupt. To service the DPI, unmask (set = 1) the P14I bit in the IMASK register. For the DPI system interrupt controller the DPI_IMASK_RE register must be unmasked. For example:

```
bit set IMASK P14I;          /* unmasks P14I interrupt */
ustat1=dm(DPI_IMASK_RE);     /* set UART TX Int */
bit set ustat1 UART0_TX_INT;
dm(DPI_IMASK_RE)=ustat1;
```

The separate UART0RXI and UART0TXI signals are not routed by default to programmable interrupt. To service the UART0TXI/UART0RXI, unmask (set = 1) any programmable interrupt bit in the IMASK/LIRPTL register.

With the UART enabled (UARTEN bit, UART0TXCTL/UART0RXCTL) the UART line status interrupt is unmasked by setting UARTLSIE bit (UARTIER register). The UARTRBFIE and UARTTBEIE bits (UARTIER register) are set to enable core buffer service request

When the UARTTBEIE bit is set in the UARTIER register for core transmit transfers, the UART module immediately issues an interrupt.

With UART DMA enabled (UARTDEN bit, UART0TXCTL/UART0RXCTL) the UART uses dedicated DMA channels for receive and transmit operations.

Service

The following sections describe interrupt servicing for the UART.

The DPI_INT interrupt is automatically cleared when DPI_IRPTL is read, for the MISCBxI, UART0RXI and UART0TXI for DMA mode only interrupts. Further, the UART0RXI interrupt for core mode must be cleared in the ISR by following the UART interrupt acknowledge mechanism (see IIR section).

Core Buffer Service Request

When initiating the transmission of a string, no special handling of the first character is required. Let the interrupt service routine (ISR) load the first character from memory and write it to the UARTTHR register in the normal manner. Accordingly, the UARTTBEIE bit should be cleared if the string transmission has completed. Alternatively, UART writes and reads can be accomplished by ISRs.

Note that the UART0LSR status register should always be read before the receive buffer register (UART0RBR). Address detection (9 bit UART) can also generate the interrupt both in I/O and DMA modes. To check for the 9th bit being high, programs should read the UART0RBRSH (shadow) register.

The UART interrupt identification register (UARTIIR) reflects the UART interrupt status (see Table 21-4) by bundling all UART0RXI interrupt sources to a single interrupt channel and servicing them all by the same software routine. The transmit interrupt request is cleared by writing new data to the UART0THR register or by reading the UART0IIR register.

Please note the special role of the UARTIIR register read in the case where the service routine does not want to transmit further data. If software stops transmission, it must read the UARTIIR register to reset the interrupt request. As long as the UARTIIR register reads receive service (0x04) or receive status line (0x06) indicating that another interrupt of higher priority is pending, the transmit service (0x01) latch cannot be cleared by reading the UARTIIR register.

Errors

The UART0LSR register reports the cause of the interrupt. The error interrupts can be determined by reading the UART0LSR status register. The bits causing the interrupt must be RW1C.

The UART0RXSTAT register reports if the interrupt is due to receive DMA errors. The error interrupts can be determined by reading the UART0LSR status register. The bits causing the interrupt must be cleared through the RW1C operation. This clears the UARTERRIRQ bit in the UART0RXSTAT register.

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

UART Effect Latency

After the UART registers are configured the effect latency is 2 PCLK cycles. Note that when transmitting data the effective data on DPI pins can't be seen immediately after 2 PCLK cycles because the time which the UART takes to start driving data depends on the baud rate settings.

Programming Model

The following sections provide some programming procedures for core and DMA data transfers.

The UART allows mapping the interrupts via the DPI interrupt (default) or separately by programming UART0TXI and UART0RXI to any other programmable interrupt.

Autobaud Detection

When the baud rate of the incoming signal is not known, one of the general-purpose timers can be used in width capture mode to automatically calculate the baud rate. Do not enable the UART until the width is captured by the timer. To perform autobaud detection, use the following procedure.

1. The UART RX input signal is fed to a DPI pin buffer. This buffer is used as an input (DPI_PBENxx_I is low). The pin buffer output (DPI_PBxx_O) is routed to both inputs UART0RX_I and TIMERx_I.

2. Configure the timer in width capture mode with the timer interrupt enabled. Inside the ISR, the program should read the width of the incoming signal and disable the timer.

3. Send test data through the host device that can be used for calculating the baud rate of the incoming signal. A NULL character (0x00) can be used for this purpose.

4. The baud rate can be derived from the width of timer as follows: BAUDR = Width ÷ (8 × (number of zero data bits + 1)).

5. When using a NULL character, the number of zero data bits is 8. Programs can also send some other pattern for this purpose. Once the baud rate is calculated inside the timer ISR, the UART can be programmed with the calculated baud rate.

DMA Transfers

The following is the general procedure for transferring data using DMA.

1. Clear the global UARTTXCTL/UARTRXCTL register.
2. Configure the UART DMA parameter registers (index, modify and count).
3. Configure the UARTLCR, UARTDLL, UARTDLH, UARTIER, UARTSCR and UARTMODE registers (see the DLAB bit).
4. Enable the DMA by setting the UARTEN and DMAEN bits in the UART-TXCTL/UARTRXCTL registers.

Setting Up and Starting Chained DMA

To start a chain pointer DMA use the following steps.

1. Clear the chain pointer register
2. Initialize the chain pointer register with the address of the DMA descriptor table. Set the PCI bit if an interrupt is needed at the end of each DMA block.
3. Set up the appropriate control register to enable the UART transmitter and receiver, chain pointer, and DMA (UARTDEN, UARTEN, UARTCHEN bits). Once chain pointer DMA is enabled, the DMA engine fetches the index, modify, count, and chain pointer values from the memory address specified in the chain pointer register. Once the DMA parameters are fetched, normal DMA starts. This process is continued until the chain pointer register contains all zeros.

Notes on Using UART DMA

The following should be noted when performing DMA through the UART.

- DMA can be interrupted by resetting the UARTDEN bit, but none of the other control settings should be changed. If the UART is enabled again, then interrupted DMA can be resumed by resetting the UARTDEN bit.
- Disabling the UART by resetting the enable UARTEN bit flushes data in the transmit/receive buffer. Resetting the UART during a DMA operation is prohibited and leads to data loss.
- Do not disable chaining (UARTCHEN bit) when a chaining DMA is in progress.
- During a receive DMA, a read of the receiver buffer (UARTRBR) is not allowed. If needed, programs should read the receiver shadow buffer (UARTRBRSH).

Core Transfers

The following is the general procedure for transferring data using the core.

1. Clear the global UARTTXCTL/UARTRXCTL registers.
2. Configure the UARTLCR, UARTDLL, UARTDLH, and UARTMODE registers (see the DLAB bit).
3. Program the UARTIER registers to generate interrupt when the transmit buffer is empty and / or the receive buffer is full.
4. Enable the UART by setting the UARTEN bit in the UARTTXCTL/ UARTRXCTL registers.

5. Inside the ISR, check for the interrupt triggered and write to the transmit buffer in case of transmit buffer empty and read from the receive buffer in case of receive buffer fill event.

9-Bit Transmission and Packing Transfers

Programs should write the UARTPKSYN bit (bit 1) with a 1 each time an address is received. This starts the reception of the following data from the lower half-word of the UARTRBR register.

The address-detect interrupt is generated whenever the UART receiver receives an address, irrespective of the packing. The DR bit in the UARTLSR register can be used to discover whether the address is in the lower (DR = 0) or higher half-word (DR = 1). The LSR register must be read before reading the UARTRBR register, because the latter clears the DR bit. Reading the UARTRBR register clears both the address-detect and the data-ready interrupts. In non-packed mode, when the address-detect interrupt is generated, it means that the data is ready in the RBR buffer while in packed mode, this is not the case.

Debug Features

The following sections describe the debugging features available on the UART.

Shadow Registers

Because of the destructive nature of reading the following registers: interrupt identification (UARTIIR), line status (UARTLSR) and read buffer (UARTRBR) shadow registers are provided for reading the contents of the corresponding main registers. The shadow registers, (UARTIIRSH), (UARTLSRSH) and (UARTRBRSH) return exactly the same contents as the main register, but without changing the register's status in any way.

Shadow Buffer

Because of the destructive nature of reading the read buffer (UART0RBR) a shadow buffer is provided. The shadow buffer (UART0RBRSH) returns exactly the same contents as the main buffer, but without changing the register's status in any way.

Shadow Interrupt Registers

For more information, see "Debug Features" on page 2-15.

Loopback Routing

The UART supports an internal loopback mode by using the SRU. For more information, see "Loopback Routing" on page 10-40.

22 TWO-WIRE INTERFACE CONTROLLER

The two-wire interface (TWI) controller allows a device to interface to an inter-IC bus as specified by Philips. The TWI is fully compatible with the widely used I^2C bus standard. It is designed with a high level of functionality and is compatible with multi-master, multi-slave bus configurations. To preserve processor bandwidth, the TWI controller can be set up with transfer initiated interrupts to service FIFO buffer data reads and writes only. Protocol related interrupts are optional.The TWI specifications are shown in Table 22-1.

Table 22-1. TWI Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	No
SRU DAI Default Routing	N/A
SRU2 DPI Required	Yes
SRU2 DPI Default Routing	Yes
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	Yes
Transmission Simplex	
Transmission Half Duplex	
Transmission Full Duplex	

Table 22-1. TWI Specifications (Cont'd)

Feature	Availability
Access Type	
Data Buffer	Yes
Core Data Access	Yes
DMA Data Access	No
DMA Channels	N/A
DMA Chaining	N/A
Boot Capable	No
Local Memory	No
Max Clock Operation	400 kHz

Features

The TWI is fully compatible with the widely used I^2C bus standard. It was designed with a high level of functionality and is compatible with multimaster, multislave bus configurations. To preserve processor bandwidth, the TWI controller can be set up and a transfer initiated with interrupts only. This allows the processor to service FIFO buffer data reads and writes. Protocol-related interrupts are optional. The TWI master controller includes the features described in the list that follows.

- Simultaneous master and slave operation on multiple device systems
- Support for multimaster data arbitration
- Support for fast mode (400 KHz)
- 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 a bus lockup
- Input filter for spike suppression

The TWI moves 8-bit data externally while maintaining compliance with the I^2C bus protocol.

Pin Descriptions

Table 22-2 shows the pins for the TWI. Two bidirectional pins externally interface the TWI controller to the I^2C bus. The interface is simple and no other external connections or logic are required.

Table 22-2. TWI Pins

Internal Node	Type	Description
TWI_CLK_I	I	**TWI Clock Signal.** Serial clock input.
TWI_DATA_I	I	**TWI Data Signal.** Serial receive data input.
TWI_CLK_PBEN_O	O	**TWI Clock Signal.** This output signal is used to drive the TWI clock off chip Note since the TWI output signals must operate in open drain it should be routed to a DPI PBEN input.
TWI_DATA_PBEN_O	O	**TWI Data Signal.** This output signal is used to drive the TWI data off chip. Note that since the TWI output signals must operate in open drain it should be routed to a DPI PBEN input.

SRU Programming

The TWI signals are available through the SRU2, and are routed as described in Table 22-3.

Table 22-3. TWI DPI/SRU2 Signal Connections

TWI Source	DPI Connection	TWI Destination
	Group A	TWI_CLK_I TWI_DATA_I
TWI_CLK_PBEN_O TWI_DATA_PBEN_O	Group C	

Clocking

The fundamental timing clock of the TWI module is peripheral clock (PCLK). Serial clock frequencies can vary from 400 kHz to less than 20 kHz. The resolution of the generated clock is 1/10 MHz or 100 ns.

CLKDIV = TWI_CLOCK period ÷ 10 MHz time reference

For example, for an TWI_CLOCK 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 TWI_CLOCK with a 30% duty cycle, then CLKLOW = 17 and CLKHI = 8. Note that CLKLOW and CLKHI add up to CLKDIV.

The clock to this module may be shut off for power savings. For more information, see “Disabling Peripheral Clocks” on page 23-11.

Register Overview

This section provides brief descriptions of the major registers. For complete information see "Two-Wire Interface Registers" on page A-245.

Slave Mode Control Register (TWISCTL). 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.

Slave Mode Status Register (TWISSTAT). During and at the conclusion of slave mode transfers, the TWISSTAT 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.

Master Mode Control Register (TWIMCTL). 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.

Master Mode Status Register (TWIMSTAT). 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 offer information on the current transfer. Slave mode operation does not affect master mode status bits.

Control Timer Register (TWIMITR). Enables the TWI module and establishes a relationship between the peripheral clock (PCLK) and the TWI controller's internally-timed events. The internal time reference is derived from PCLK using the prescaled value shown below.

$$\text{PRESCALE} = f_{PCLK}/10\ \text{MHz}$$

Serial Clock Divider Register (TWIDIV). During master mode operation, the TWIDIV register values are used to create the high and low durations of the TWI_CLOCK.

FIFO Control Register (TWIFIFOCTL). The FIFO control register affects only the FIFO and is not tied in any way with master or slave mode operation.

FIFO Status Register (TWIFIFOSTAT). The fields in the TWI FIFO status register 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.

Functional Description

Figure 22-1 illustrates the overall architecture of the TWI controller.

The peripheral interface supports the transfer of 32-bit wide data and is used by the processor in the support of register and FIFO buffer reads and writes.

The register block contains all control and status bits and reflects what can be written or read as outlined by the programmer's model. Status bits can be updated by their respective functional blocks.

The FIFO buffer is configured as a 1-byte-wide, 2-deep transmit FIFO buffer and a 1-byte-wide, 2-deep receive FIFO buffer.

The transmit shift register serially shifts its data out externally off chip. The output can be controlled to generate acknowledgements or it can be manually overwritten.

The 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.

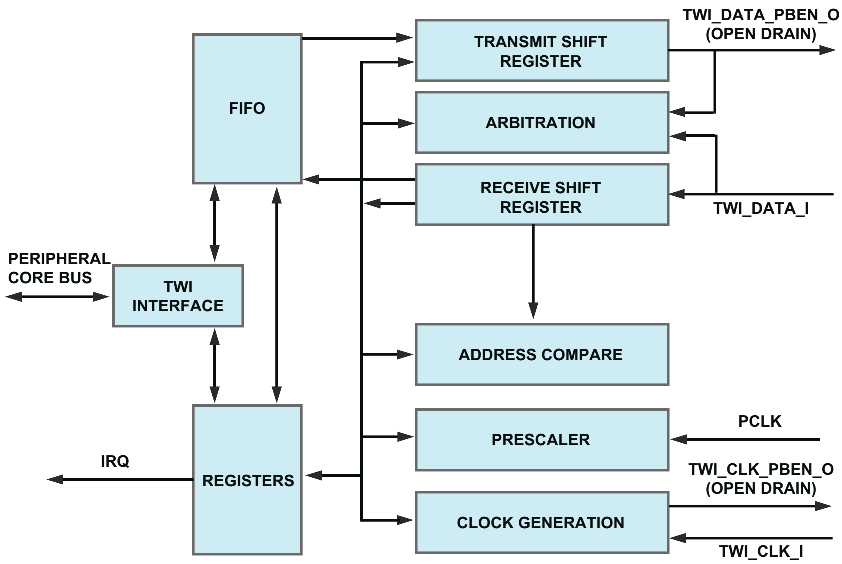

Figure 22-1. TWI Block Diagram

The address compare block supports address comparison in the event the TWI controller module is accessed as a slave.

The prescaler block must be programmed to generate a 10 MHz time reference relative to the peripheral clock. This time base is used for filtering data and timing events specified by the electrical parameters in the data sheet (see the I^2C bus specification from Philips), as well as for `TWI_CLOCK` clock generation.

The clock generation module is used to generate an external serial clock (`TWI_CLOCK`) when in master mode. It includes the logic necessary for synchronization in a multimaster clock configuration and clock stretching when configured in slave mode.

The TWI controller's clock output follows these rules:

1. Once the clock high (CLKHI) count is complete, the serial clock output is driven low and the clock low (CLKLOW) count begins.

2. Once the clock low count is complete, the serial clock line is three-stated and the clock synchronization logic enters into a delay mode (shaded area) until the TWI_CLOCK line is detected at a logic 1 level. At this time, the clock high count begins.

The TWI controller 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 illustrated in Figure 22-2.

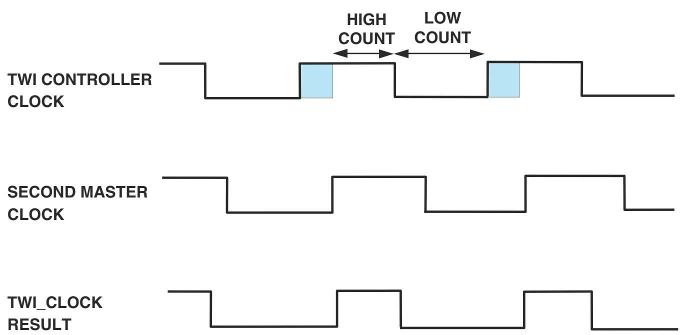

Figure 22-2. TWI Clock Synchronization

The TWI controller follows the transfer protocol of the *Philips I²C Bus Specification version 2.1* dated January 2000. A simple complete transfer is diagrammed in Figure 22-3.

S	7-BIT ADDRESS	R/W	ACK	8-BIT DATA	ACK	P

S = START
P = STOP
ACK = ACKNOWLEDGE

Figure 22-3. Standard Data Transfer

To better understand the mapping of TWI controller register contents to a basic transfer, Figure 22-4 details the same transfer as 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.

S	MADDR[6:0]	MDIR	ACK	XMITDATA8[7:0]	ACK	P

S = START
P = STOP
ACK = ACKNOWLEDGE

Figure 22-4. Data Transfer With Bit Illustration

Bus Arbitration

The TWI controller initiates a master mode transmission (TWIMEN) only when the bus is idle. If the bus is idle and two masters initiate a transfer, arbitration for the bus begins. This is illustrated in Figure 22-5.

The TWI controller monitors the serial data bus (TWI_DATA) while the TWI_CLOCK is high. If TWI_DATA is determined to be an active logic 0 level while the internal TWI controller's data is a logic 1 level, the TWI controller has lost arbitration and ends generation of clock and data. Note that arbitration is performed not only at serial clock edges, but also during the entire time TWI_CLOCK is high.

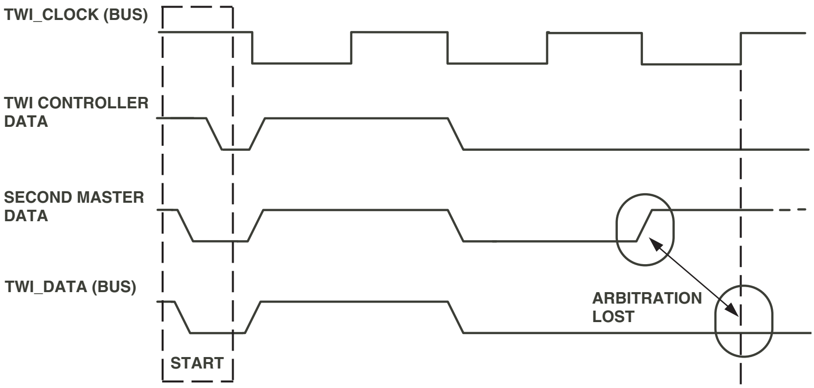

Figure 22-5. TWI Bus Arbitration

Start and Stop Conditions

Start and stop conditions involve serial data transitions while the serial clock is at 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 Figure 22-6.

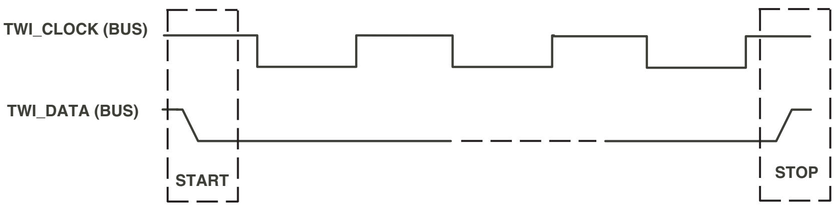

Figure 22-6. TWI Start and Stop Conditions

The TWI controller's special-case start and stop conditions include:

- TWI 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 (TWISCOMP).

- TWI 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 (TWISCOMP) and indicates a slave transfer error (TWISERR).

- TWI controller as a master-transmitter or master-receiver

  If the stop bit is set during an active master transfer, the TWI controller issues a stop condition as soon as possible to avoid any error conditions (as if data transfer count had been reached).

Operating Modes

The following sections provide information on the operation modes of the interface.

General Call Addressing

The TWI controller always decodes and acknowledges a general call address if it is enabled as a slave (TWISEN) and if general call is enabled using the TWIGCE bit. General call addressing (0x00) is indicated by the setting of the GCALL bit, and by the nature of the transfer, the TWI controller is a slave-receiver. If the data associated with the transfer is to be not acknowledged (NAKed), the TWINAK bit can be set.

If the TWI controller is to issue a general call as a master-transmitter, the appropriate address and transfer direction can be set along with loading transmit FIFO data.

Slave Mode Addressing

With the appropriate selection of 7-bit addressing using the TWISLEN bit, the corresponding number of address bits (SADDR) are referenced during the address phase of a transfer.

Master Mode Addressing

Whether enabled as a master-transmitter or master-receiver with 7-bit addressing using the TWIMLEN bit, the TWI master performs all addressing and data transfers as required. This includes generating the repeated start condition, re-transmission of the 7-bits of the first address byte, and acknowledgement and generation of a new transfer direction change (indicated by the TWIMLEN bit).

Fast Mode

Fast mode (400 kHz) uses essentially the same mechanics as standard mode (100 kHz). It is the electrical specifications and timing that are different. When fast mode is enabled using the TWIFAST bit, the following timings are modified to meet the electrical requirements.

- Serial data rise times before arbitration evaluation (t_r)
- Stop condition setup time from serial clock to serial data (t_{SUSTO})
- Bus free time between a stop and start condition (t_{BUF})

Data Transfer

The TWI uses its transmit and receive buffers for data transfer (no DMA capability). These buffers are described in the following sections.

Serial Shift Register

The TWI has both an input and output serial shifter which are described below.

Output Shift Register

The transmit shift register receives byte wide buffer data or register data (address) and serially shifts its data out externally off chip. The output can be controlled for generation of acknowledgements or can be manually over-written

Input Shift Register

The receive shift register receives its data serially from off chip. Internally the receive shift register is byte wide and data received can either be transferred to the buffer or used in an address comparison.

Buffers

The TWI has multiple receive and transmit data buffers for 8 and 16-bit data, which are described in the following sections.

Each buffer is accessed independently of the other and can be accessed simultaneously. As an example, a write to the transmit buffer could occur at the same time a receive shift register writes to the receive buffer.

8-Bit Transmit Buffer

The TWI 8-bit transmit FIFO register (`TXTWI8`) shown in Figure 22-7 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. Although peripheral bus writes are 32 bits, a write access to the `TXTWI8` register adds only one transmit data byte to the FIFO buffer. With each access, the transmit status (`TWITXS`) field in the `TWIFIFOSTAT` register is updated. If an access is performed while the FIFO buffer is full, the core waits until there is at least one byte space in the transmit FIFO buffer and then completes the write access.

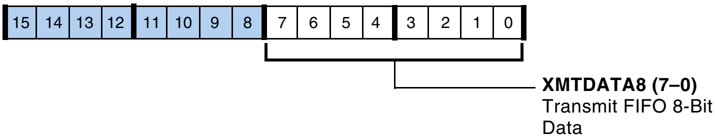

Figure 22-7. 8-Bit Transmit FIFO Register

16-Bit Transmit Buffer

The TWI 16-bit FIFO transmit register (TXTWI16) shown in Figure 22-8 holds a 16-bit data value written into the FIFO buffer. Although peripheral bus writes are 32 bits, a write access to the TXTWI16 register adds only two transmit data bytes to the FIFO buffer. To reduce interrupt output rates and peripheral bus access times, a double byte transfer data access can be performed. 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 (TWITXS) field in the TWIFIFOSTAT register is updated. If an access is performed while the FIFO buffer is not empty, the core waits until the FIFO buffer is completely empty and then completes the write access.

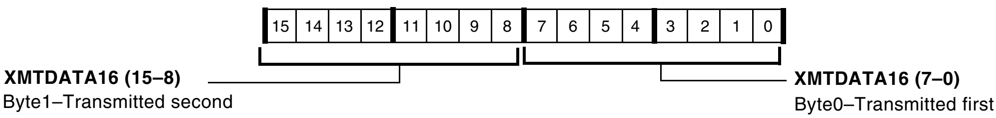

Figure 22-8. 16-Bit Transmit FIFO Register

8-Bit Receive Buffer

The TWI 8-bit FIFO receive register (RXTWI8) shown in Figure 22-9 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 32 bits, a read access to the RXTWI8 register can only access one receive data byte from the FIFO buffer. With each access, the receive status (TWIRXS) field in the TWIFIFOSTAT register is updated. If an access is performed while the FIFO buffer is empty, the core waits until there is at least one byte in the receive FIFO buffer and then completes the read access.

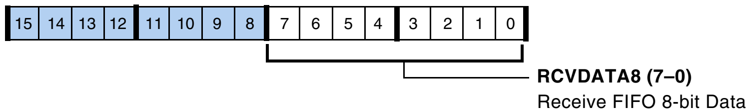

Figure 22-9. 8-Bit Receive FIFO Register

16-Bit Receive Buffer

The TWI 16-bit FIFO receive register (RXTWI16) shown in Figure 22-10 holds a 16-bit data value read from the FIFO buffer. Although peripheral bus reads are 32 bits, a read access to the RXTWI16 register can only access two receive data bytes 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.

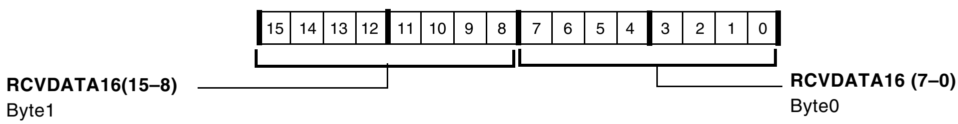

Figure 22-10. 16-Bit Receive FIFO Register

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.

Buffer Status

With each write access, the transmit status (TWITXS) field in the TWIFIFOSTAT register is updated. If an access is performed while the FIFO buffer is full, the core waits until there is at least one byte space in the transmit FIFO buffer and then completes the write access.

With each read access, the receive status (TWIRXS) field in the TWIFIFOSTAT register is updated. If an access is performed while the FIFO buffer is empty, the core waits until there is at least one byte in the receive FIFO buffer and then completes the read access.

Buffer Error

The master status register (TWIMSTAT) reports buffer writes/underflow errors (TWIWERR) or buffer read/overflow errors (TWIRERR).

Flushing the Buffer

The TWI RX/TX buffers are flushed only by setting the FIFOFLUSH bit.

Buffer Hang Disable

For more information, see “Buffer Hang Disable” on page 22-16.

Interrupts

Table 22-4 provides an overview of TWI interrupts.

Table 22-4. Overview of TWI Interrupts

Default Programmable Interrupt	Sources	Masking	Service
DPII = P14I	Master TX complete Master buffer service Slave initiative Slave complete Master error Slave error Slave overflow	DPI_IMASK_RE TWIIMASK	RW1C to TWIIRPTL + RTI instruction
Separate TWII not connected by default		TWIIMASK	

Sources

The TWI module generates eight local interrupts which are grouped into five system status and three error status interrupts. All eight signals are logically ORed into one TWI output interrupt signal which is routed by default as a DPI interrupt into the core IVT address map.

The TWI port generates interrupts as described in the following sections.

Slave Status

Transfer initiate or transfer complete.

Master Status

Transfer complete or TX/RX buffer service.

Error

Transfer error or transfer overflow.

Masking

The DPI signal is routed by default to programmable interrupt. To service the DPI, unmask (set = 1) the P14I bit in the IMASK register. For DPI system interrupt controller the DPI_IMASK_RE register must be unmasked. For example:

```
bit set IMASK P14I;        /* unmasks P14I interrupt */
ustat1=dm(DPI_IMASK_RE);     /* set TWI Int */
bit set ustat1 TWI_INT;
dm(DPI_IMASK_RE)=ustat1;
```

The separate TWII signal is not routed by default to programmable interrupts. To service the TWII, unmask (set = 1) any programmable interrupt bit in the IMASK/LIRPTL register. The local interrupts can be unmasked by setting the corresponding bits in TWIIMASK register.

Service

For the default DPI interrupt, reading the DPI_IRPTL register for a TWI service request does not clear the request. Instead, the ISR must first read the TWIIRPTL register to identify the root cause and write 1 to that bit to clear the request as shown in Listing 22-1.

The DPI_INT interrupt is automatically cleared when the DPI_IRPTL register is read for the MISCBxI, UART0TXI and UART0RXI for DMA mode only interrupts. The TWI interrupt must be cleared in the ISR by following the TWI interrupt acknowledge mechanism. For more information, see “Interrupt Latch Register (TWIIRPTL)” on page A-259.

Listing 22-1. Clearing Transmit Buffer Interrupt

```
TWI_ISR:
ustat1 = dm(TWIIRPTL);      /* read IRPTL to identify cause*/
bit TST ustat1 TWITXINT;    /* test TX buffer bit*/
IF TF jump TX_BUF;
TX_BUF:
dm(TWIIRPTL) = ustat1;      /* RW1C to clear TWI TX buffer
                               interrupt */
r0=dm(TWIMCTL);             /* dummy read*/
instruction;
rti;
```

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific).

Write Effect Latency

For details on write effect latency, see *SHARC Processor Programming Reference*.

TWI Effect Latency

After the TWI registers are configured the effect latency is 1.5 PCLK cycles minimum and 2 PCLK cycles maximum.

Programming Model

The following sections include information for general setup, slave mode, and master mode, as well as guidance for repeated start conditions.

General Setup

General setup refers to register writes that are required for both slave mode and master mode operation. General setup should be performed before either the master or slave enable bits are set.

Programs should enable the TWI controller through the TWIMITR register and set the prescale value. Program the prescale value to the binary representation of f_{PCLK}/10 MHz.

All values should be rounded up to the next whole number. The TWIEN enable bit must be set. Note that once the TWI controller is enabled, a bus busy condition may be detected.

SRU Programming Mode

Since the TWI data/clock output requires open drain connectivity following SRU programming is required regardless of master/slave mode:

Listing 22-2. TWI Pin Routing

```
SRU(LOW, DPI_PBxx_I);                 /* input buffer is low */
SRU(TWI_DATA_PBEN_O, DPI_PBENxx_I);   /* TWI data output*/
SRU(DPI_PBxx_O, TWI_DATA_I);          /* TWI data input */
```

Listing 22-3. TWI Clock Pin Routing

```
SRU2(LOW, DPI_PByy_I);                /* input buffer is low */
SRU2(TWI_CLK_PBEN_O, DPI_PBENyy_I);   /* TWI clock output*/
SRU2(DPI_PByy_O, TWI_CLK_I);          /* TWI clock input*/
```

Slave Mode

When enabled, slave mode 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 `TWISADDR` register. The appropriate 7 bits are used in determining a match during the address phase of the transfer.
2. Program the `TXTWI8` or `TXTWI16` register. These are the initial data values to be transmitted in the event the slave is addressed as a transmitter. This is an optional step. If no data is written and the slave is addressed and a transmit is required, the TWI clock is stretched and an interrupt is generated.
3. Program the `TWIFIFOCTL` register. Indicate if transmit (or receive) FIFO buffer interrupts should occur with each byte transmitted (received) or with each 2 bytes transmitted (received).
4. Program the `TWIIMASK` register. Enable bits associated with the desired interrupt sources. As an example, programming the value 0x000F results in an interrupt output to the processor when a valid address match is detected, a valid slave transfer completes, a slave transfer has an error, or a subsequent transfer has begun but the previous transfer has not been serviced.
5. Program the `TWISCTL` 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.

Table 22-5 shows what the interaction between the TWI controller and the processor might look like when the slave is addressed as a receiver.

Table 22-5. Slave Mode Setup Interaction (Slave Addressed as Receiver)

TWI Controller Master	Processor
Interrupt: TWISINIT – Slave transfer has been initiated.	Change on the next sides always. Interrupt Acknowledge: RW1C the TWIIRPTL register.
Interrupt: TWIRXS – Receive buffer has 1 or 2 bytes (according to TWIRXINT).	Read receive FIFO buffer. Change on the next sides always. Interrupt Acknowledge: RW1C the TWIIRPTL register
...	...
Interrupt: TWISCOMP – Slave transfer complete.	Read receive FIFO buffer. Acknowledge: Clear interrupt source bits.

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.

Program the `TWIDIV` register. This defines the clock high duration and clock low duration.

Master Mode Transmit

Follow these programming steps for a single master mode transmit:

1. Program the `TWIMADDR` register. This defines the address transmitted during the address phase of the transfer.

2. Program the TXTWI8 or TXTWI16 registers. 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 TWIFIFOCTL register. Indicate if transmit FIFO buffer interrupts should occur with each byte transmitted (8 bits) or with each 2 bytes transmitted (16 bits).

4. Program the TWIIMASK register. Enable the bits associated with the desired interrupt sources. For example, programming the value 0x0030 results in an interrupt output to the processor when the master transfer completes, or if a master transfer error has occurred.

5. Program the TWIMCTL 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.

Table 22-6 shows what the interaction between the TWI controller and the processor might look like using this example.

Table 22-6. Master Mode Transmit Setup Interaction

TWI Controller Master	Processor
Interrupt: TWITXINT – Transmit buffer has 1 or 2 bytes empty (according to XMTINTLEN).	Write transmit FIFO buffer. Change on the next sides always. Interrupt Acknowledge: RW1C the TWIIRPTL register.
...	...
Interrupt: TWIMCOM – Master transfer complete.	Change on the next sides always. Interrupt Acknowledge: RW1C the TWIIRPTL register

Master Mode Receive

Follow these programming steps for a single master mode transmit:

1. Program the TWIMADDR register. This defines the address transmitted during the address phase of the transfer.
2. Program the TWIFIFOCTL register. Indicate if receive FIFO buffer interrupts should occur with each byte received (8 bits) or with each 2 bytes received (16 bits).
3. Program the TWIIMASK 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 TWIMCTL register. Ultimately 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-receive, uses standard mode timing, and receives 8 data bytes before generating a stop condition.

Table 22-7 shows what the interaction between the TWI controller and the processor might look like using this example.

Table 22-7. Master Mode Receive Setup Interaction

TWI Controller Master	Processor
Interrupt: TWIRXINT – Receive buffer has 1 or 2 bytes (according to RCVINTLEN).	Read receive FIFO buffer. Change on the next sides always. Interrupt Acknowledge: RW1C the TWIIRPTL register.
...	...
Interrupt: TWIMCOM – Master transfer complete.	Read receive FIFO buffer. Change on the next sides always. Interrupt Acknowledge: RW1C the TWIIRPTL register.

Repeated Start Condition

In general, a repeated start condition is the absence of a stop condition between two transfers initiated by the same master. The two transfers can be of any direction type. Examples include a transmit followed by a receive, or a receive followed by a transmit. During a repeated start transfer, each interrupt must be serviced correctly to avoid errors. The following sections are intended to assist the programmer with service routine development.

Transmit/Receive Repeated Start Sequence

Figure 22-11 illustrates a repeated start data transmit followed by a data receive sequence.

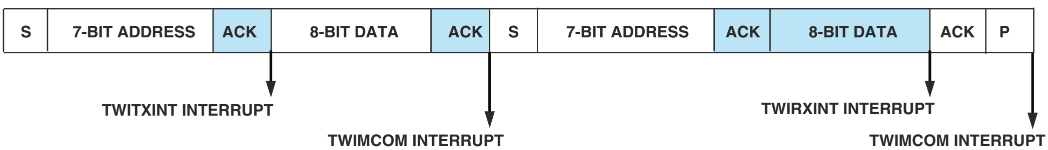

Figure 22-11. Transmit/Receive Data Repeated Start

The tasks performed at each interrupt are:

- `TWITXINT` interrupt – Generated every time the transmit FIFO has one or two byte locations available to be written. To service this interrupt, write a byte or word into the transmit FIFO registers (`TXTWI8` or `TXTWI16`). During one of these interrupts (preferably the first time), do the following:
  - Set the `TWIRSTART` bit (or earlier when `TWIMCTL` register is programmed first).
  - Set the `TWIMDIR` bit to indicate the next transfer direction is receive. This should be done before the addressing phase of the next transfer begins.

- TWIMCOM interrupt – Generated because all data has been transferred (DCNT = 0). If no errors occur, a start condition is initiated. At this time, program the following bits of TWI_MASTER_CTRL register:
    - Clear TWIRSTART (if this is the last transfer).
    - Re-program DCNT with the desired number of bytes to receive.
- TWIRXINT interrupt – Generated due to the arrival of a byte into the receive FIFO. Simple data handling is all that is required.
- TWIMCOM interrupt – Transfer is complete.

Receive/Transmit Repeated Start Sequence

Figure 22-12 illustrates a repeated start data receive followed by a data transmit sequence. The shading indicates the slave has the bus.

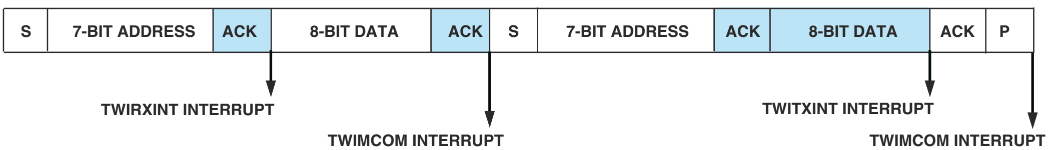

Figure 22-12. Receive/Transmit Data Repeated Start

The tasks performed at each interrupt are:

- TWIRXINT interrupt

  This interrupt is generated due to the arrival of one or two data bytes into the receive FIFO. The TWIRSTART bit should be set at this time (or earlier) and TWIMDIR should be cleared to reflect the change in direction of the next transfer. The TWIMDIR bit must be cleared before the addressing phase of the subsequent transfer begins.

- TWIMCOM interrupt

  This interrupt has occurred due to the completion of the data receive transfer. At this time the data transmit transfer begins. The TWIDCNT field should be set to reflect the number of bytes to be transmitted. Clear the TWIRSTART bit if this is the last transfer.

- TWITXINT interrupt

  This interrupt is generated when there is one or two bytes of empty space in the FIFO. Simple data handling is all that is required.

- TWIMCOM interrupt

  The transfer is complete.

Electrical Specifications

All logic complies with the electrical specification outlined in the *Philips I^2C Bus Specification version 2.1* dated January, 2000.

Debug Features

The following section provides information on debugging features available with the TWI.

Buffer Hang Disable

To support debugging buffer transfers, the processors have a buffer hang disable (BHD) bit in the TWIFIFOCTL register. When set (=1), this bit prevents the processor core from detecting a buffer-related stall condition, permitting debugging of this type of stall condition. For more information, see "Buffer Hang Disable (BHD)" on page 11-65.

Shadow Interrupt Registers

For more information, see "Debug Features" on page 2-15.

Loopback Routing

The controller supports an internal loopback mode by using the SRU. For more information, see "Loopback Routing" on page 10-40.

23 POWER MANAGEMENT

This chapter describes how to control the power use of the SHARC by controlling the clocks that run each peripheral.

Features

The following list describes the power management features.

- The PLL has various multiplier and divisor settings to generate a flexible core clock.
- Allows changes to the output clock during runtime.
- $\overline{\text{RESETOUT}}$ pin can be used for boot handshake or as a debug aid.
- Resetting the PLL is possible without performing a new power-up sequence.
- Power savings controls the shut-down of individual clocks to peripherals. For information on which peripherals are enable after reset, see "Peripherals Enabled by Default" on page 23-10.

Register Overview

Power Management Control Register (PMCTL). Governs the operation of the PLL and configures the PLL settings.

Power Management Control Register 1 (PMCTL1). This register controls the various peripheral's clocks.

Phase-Locked Loop (PLL)

The following sections describe the clocking system of the SHARC processor. This information is critical to ensure designs that work correctly and efficiently.

Functional Description

To provide the clock generation for the core and system, the processor uses an analog PLL with programmable state machine control. The PLL design serves a wide range of applications. It emphasizes embedded applications and low cost for general-purpose processors, in which performance, flexibility, and control of power dissipation are key features. This broad range of applications requires a range of frequencies for the clock generation circuitry. The input clock may be a crystal, an oscillator, or a buffered, shaped clock derived from an external system clock oscillator. The clock system is shown in Figure 23-1.

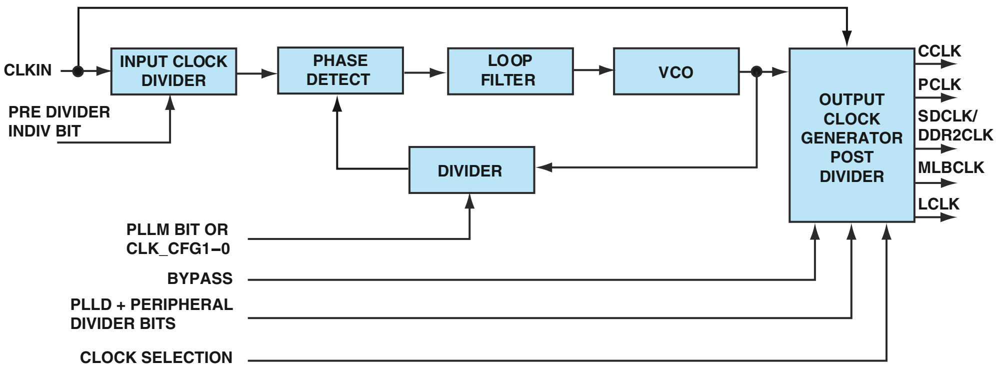

Figure 23-1. Clocking System

Subject to the maximum VCO frequency, the PLL supports a wide range of multiplier ratios of the input clock, `CLKIN`. To achieve this wide multiplication range, the processor uses a combination of programmable multipliers in the PLL feedback circuit and output configuration blocks.

The processor uses an on-chip, phase-locked loop (PLL) to generate its internal clock, which is a multiple of the CLKIN frequency. The PLL requires some time to achieve phase lock and CLKIN must be valid for a minimum time period during reset before the $\overline{\text{RESET}}$ signal can be deasserted.

For information on minimum clock setup, external crystal use, and range for any given CLKIN frequency, and for a detailed diagram along with specific equations on the derivation of VCO frequency with reference to CLKIN, see the appropriate product data sheet.

PLL Input Clock

If an external clock oscillator is used, it should NOT drive the CLKIN pin when the processor is not powered. The clock must be driven immediately after power-up; otherwise, internal gates stay in an undefined (hot) state and can draw excess current. After power-up, allow sufficient time for the oscillator to start up, reach full amplitude, and deliver a stable CLKIN signal to the processor before the reset is released. This may take several milliseconds and depends on the choice of crystal, operating frequency, loop gain and capacitor ratios. For details on timing, refer to the appropriate product data sheet.

Pre-Divider Input

This unit divides the PLL input clock by 2 if enabled (using the INDIV bit). The pre-divider input is part of the PLL loop, therefore, if a program changes the PLL input clock (affecting the VCO frequency), the PLL must be put in bypass mode before the change can take effect. This is described in "Bypass Mode" on page 23-7.

PLL Multiplier

The PLL multiplier bits are used to divide the VCO clock down to the CLKIN input (see Figure 23-1). The multiplier settings are controlled by hardware or software and based on the PLL multiplier settings below.

- Hardware—through the clock configuration pins (CLK_CFG1-0)
- Software—the hardware settings are overridden through the PLLM bits

PLLM Hardware Control

On power-up, the CLK_CFG1-0 pins are used to select core to CLKIN ratios which cannot be changed during runtime. After booting however, numerous other ratios (slowing or speeding up the clock) can be selected through software control.

For information on the internal clock to CLKIN frequency ratios supported by the various processors, see the product-specific data sheet.

PLLM Software Control

Programs control the PLL through the PMCTL register. The PLL multiplier (PLLM) bits can be configured to set a multiplier range of 0 to 63. This allows the PLL to be programmed dynamically in software to achieve a higher or slower core instruction rate depending on a particular system's requirements.

The reset value of the PLLM bits is derived from the CLK_CFG1-0 pin multiply ratio settings. This value can be reprogrammed in the boot kernel to take effect immediately after start- up.

PLL VCO

The VCO is the output stage of the PLL. It feeds the output clock generator which provides core and peripheral clocks as shown in Table 23-1. Two settings have an impact on the VCO frequency:

- The INDIV bit enables the CLKIN input pre-divider by 2.
- The PLLM bits and the CLK_CFG1-0 pins control the PLL multiplier unit.

Changing the VCO frequency requires a new condition for the PLL circuitry. Therefore, the core needs to wait a specific settling time in bypass mode before it can be released for further activities (typically 4096 CLKIN cycles).

Table 23-1. VCO Encodings

PLLM Bit Settings	VCO Frequency[1]	
	INDIV = 0	INDIV = 1
0	128x	64x
1	2x	1x
2	4x	2x
N = 3–62	2Nx	Nx
63	126x	63x

1 For operational limits for the VCO clock see the appropriate product data sheet.

Output Clock Generator

The output clock generator post-divides the VCO clock to the core ratio or peripherals ratio and synchronizes all output clocks. It is fed with the VCO clock and does not provide any feedback back to the PLL circuit.

If the DIVEN bit is set, new post divider ratios are picked up on the fly and the clocks smoothly transition to their new values within 14 core clock (CCLK) cycles.

Post divider ratio changes (PLLD, DDR2CKR, SDCKR and LCKR bits) do not require bypass mode.

The output clock generator block also controls bypass mode. For a description of the PMCTL bits, see "Power Management Registers (PMCTL, PMCTL1)" on page A-7.

Core Clock (CCLK)

The PLLD bits define the VCO output clock to core clock ratio to build the processor core clock (CCLK). The post divider can be changed any time and new division ratios are implemented on the fly.

IOP Clock (PCLK)

The peripheral clock is derived from the core clock with a fixed post divisor of 2. This clock is the master clock for most peripherals including the I/O processor (IOP).

Peripheral Clocks (SDRAM/DDR2/Link Port)

The SDRAM, DDR2, and link port derive their clocks directly from the core clock. These peripherals have a default divider (refer to PMCTL register). The DIVEN bit needs to be set whenever there is a change from the default ratio (similar to core PLLD bits).

Default PLL Hardware Settings

Table 23-2 demonstrates the internal core clock switching frequency across a range of CLKIN frequencies for the ADSP-2146x processor. The minimum operational range for any given frequency may be constrained by the operating range of the phase-locked loop. Note that the goal in selecting a particular clock ratio for an application is to provide the highest permissible internal frequency for a given CLKIN frequency. For more information on available clock rates, see the appropriate product data sheet.

Table 23-2. Selecting Core to CLKIN Ratio (ADSP-2146x)

	Typical Crystal and Clock Oscillators Inputs					
	12.500	16.667	25.000	33.333	40.000	50.000
Clock Ratios (CLK_CFG Pins)	**Core CLK (MHz)**[1]					
6:1[2]	N/A	100	150	200	240	300
16:1	200	266.66	400	N/A	N/A	N/A
32:1	400	N/A	N/A	N/A	N/A	N/A

1 For operational limits for the core clock frequency see the appropriate product data sheet.
2 For ADSP-2147x and ADSP-2148x models, the ratio is 8:1.

The application needs to ensure that the limits of the core clock frequency after booting are not exceeded. This is achieved by variation of the CLK_CFG pins or the CLKIN signal.

Operating Modes

The following sections provide information on the various options for clock operation.

Bypass Mode

Bypass mode must be used if any runtime VCO clock change is required. Setting the PLLBP bit bypasses the entire PLL circuitry. In bypass mode, the core runs at CLKIN speed. Once the PLL has settled into the new VCO frequency, (which may take 4096 CLKIN cycles) the PLLBP bit may be cleared to release the core from bypass mode. For more information, see "Back to Back Bypass" on page 23-17.

Only VCO frequency changes require bypass mode, therefore this mode is not intended as a standard operating mode.

Normal Mode

The normal mode is the regular mode and is effective if the PLLBP bit is cleared. In normal mode the PLL has locked and multiplies CLKIN to the desired VCO clock. The output clock generator post-divides and provides the clock tree to the I/O.

The change of PLL frequency can happen at any time (for example after power-up or during operation).

Clocking Golden Rules

The five rules below should be followed to ensure proper processor operation.

1. After power-up the CLK_CFG pins should not exceed the maximum core speed.
2. Software should guarantee minimum/maximum CCLK speed.
3. Software should guarantee maximum VCO clock speed.
4. Bypass requires 4096 CLKIN cycles.
5. Post-divider changes require 14 CCLK cycles.

Power-Up Sequence

The proper power-up sequence is critical to correct processor operation as described in the following sections.

PLL Start-Up

Before the PLL can start settling, the $\overline{\text{RESET}}$ signal should be asserted for several micro-seconds under the following conditions. For PLL information, see the appropriate product data sheet.

- Valid and stable core voltage (VDDINT)
- Valid and stable I/O voltage (VDDEXT and VDD_DDR2)
- Valid and stable clock input (CLKIN)

The chip reset circuit is shown in Figure 23-2. The PLL needs time to lock to the CLKIN frequency before the core can execute or begin the boot process. A delayed core reset signal ($\overline{\text{RESETOUT}}$) is triggered by a 12-bit counter after $\overline{\text{RESET}}$ transitions from low to high (approximately 400 µs for minimum CLKIN). The delay circuit is activated at the same time the PLL is triggered for settling after reset is deasserted.

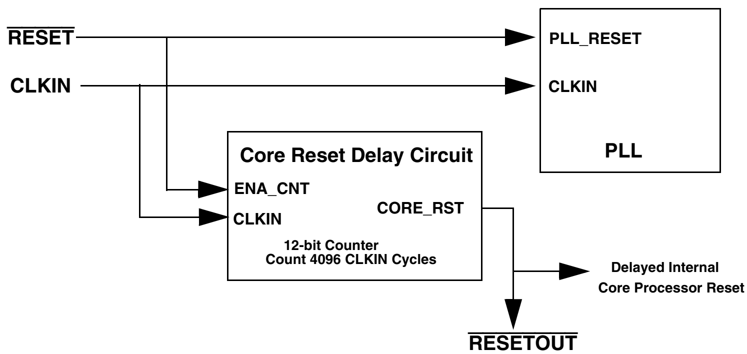

Figure 23-2. Chip Reset Circuit

After the external processor $\overline{\text{RESET}}$ signal is deasserted, the PLL starts settling. The rest of the chip is held in reset for 4096 CLKIN cycles after $\overline{\text{RESET}}$ is deasserted by an internal reset signal.

The advantage of the delayed core reset is that the PLL can be reset any number of times without having to power down the system. If there is a brownout situation, the external watchdog circuit only has to control the RESET signal. For more information on device power-up, see the appropriate product data sheet.

Peripherals Enabled by Default

When the processor is powered up the SDRAM, DDR2, S/PDIF receiver and real time clock controllers are enabled and all other peripherals disabled. If the any of the default peripherals is not to be used they can be disabled by following the steps in the sections below.

SDRAM Controller

After reset is de-asserted the SDCLK output is enabled by default. If the SDRAM interface is not used, the following bit should be configured.

In the SDCTL register, set (=1) the DSDCTL bit to disable the controller and its I/O pads.

DDR2 Controller

After reset is de-asserted the DDR2CLK output is enabled by default. If the DDR2 interface is not used, the following bits should be configured.

1. In the DDR2CTL0 register, set (=1) the DIS_DDR2CTL, DIS_DDR2CLK1 and DIS_DDR2CKE bits to disable the controller and its clocks.
2. In the DDR2PADCTL0 register (bits 9, 19 and 29) and DDR2PADCTL1 register (bits 9 and 19), set (=1) all the PWD bits to power-down the pad receivers.

S/PDIF RX Controller

If the receiver is not used, programs should disable the receiver and its digital PLL to avoid unnecessary switching. This is accomplished by setting the DIR_RESET bit in the DIRCTL register.

Real-Time Clock Controller

There is no way to disable the RTC counters from software. If a given system does not require the RTC functionality, then it may be disabled with hardware tie-offs. Tie the RTXI pin to EGND, tie the RTCVDD pin to EVDD, and leave the RTXO pin unconnected.

The RTC_INIT register is provided to power down the RTC. Power-down is interpreted as a crystal oscillator disable, which reduces I/O power dissipation to only leakage current by setting the RTC_PWD bit. Furthermore the bus logic and its level shifters between the core and I/O voltage domain can be disabled to further reduce leakage by setting RTC_BUSDIS bit. Once set or reset, this mode retains its value unless changed, irrespective of the status of core supply.

Disabling Peripheral Clocks

The PMCTL1 register (Table A-4 on page A-11) provides bits which are used to disable the individual clocks to the peripherals. Note that all clocks are enabled by default.

Routing Units

In "DAI Default Routing" on page 10-25 and "DPI Default Routing" on page 10-28 the default routing scheme after reset are listed. Programs should check if all unused inputs are tied high or low. For example the DAI pin 1 which has many default connections.

Packages Without an External Port

For reduced package sizes (88 and 100 lead) without external port connectivity it is recommended to disable the external port clock as soon as possible (since it is enabled by default) by setting the EPOFF bit in PMCTL1 register.

Example for Clock Management

The following example shows a method for using the power saving features in the SHARC processors for the SPI.

```
ustat2 = dm(PMCTL1);
bit set ustat2 SPIOFF; /* disable internal peripheral clock
                          for SPI module. */
dm(PMCTL1) = ustat2;
```

General Notes on Power Savings

The following are some additional methods for reducing power.

- The lower the operation frequency, the lower the power consumption. The core and peripherals should be operated at the lowest frequency that meets the system's requirements. Active power is proportional to the processor's core clock frequency.
- Reducing the case temperature lowers leakage power. Leakage power increases exponentially to junction temperature.
- For core pauses programs should execute the IDLE instruction. Note that an interrupt is required to release the processor from IDLE.
- Don't leave input pins floating. In some cases leakage draws current in the region of milliamps. For more information, consult the product-specific data sheet. If an external resistor is problematic, change the input to an output if possible (flag input).

Programming Models

For SDRAM programming models (AMI, SDRAM, DDR2), see "Programming Models" on page 4-155.

Post Divider

Use the following procedure and the example shown in Listing 23-1 to program or reconfigure the divider.

1. Disable any peripheral (configured with PCLK=CCLK/2). Note that the peripherals cannot be enabled when changing VCO to core clock ratio.

2. Select the PLLD divider by setting the PLLD bits (6–7) in the PMCTL register and enable the DIVEN bit.

3. Wait 15 CCLK cycles. During this time, the new divisor ratios are picked up on the fly and the clocks smoothly transition to their new values after a maximum of 14 core clock CCLK cycles.

4. Re-enable the peripherals.

Listing 23-1. Post Divider

```
ustat2 = dm(PMCTL);
bit clr ustat2 PLLBP;          /* bypass disabled*/
bit set ustat2 DIVEN|PLLD4;    /* set and enable post divisor */
dm(PMCTL) = ustat2;
lcntr = 15, do wait until lce;
wait: nop;
```

Multiplier and Post Divider Programming Model

There are two allowable procedures to program the VCO. The first method is shown in Listing 23-2.

1. Set the PLL multiplier and divisor value and enable the divisor by setting the `DIVEN` bit.
2. After one core clock cycle, place the PLL in bypass mode by setting (= 1) the `PLLBP` bit. Clear the `DIVEN` bit while placing the PLL into bypass mode.
3. Wait in bypass mode until the PLL locks (4096 `CLKIN` cycles).
4. Take the PLL out of bypass mode by clearing (= 0) the bypass bit. Clear the `DIVEN` bit while taking the PLL out of bypass mode.
5. Wait 15 core cycles before next activity.

The second method is described below and shown in Listing 23-3.

1. Set the PLL multiplier and divisor values and place the PLL in bypass mode by setting the `PLLBP` bit.
2. Wait in the bypass mode until the PLL locks (4096 `CLKIN` cycles).
3. Take the PLL out of bypass mode by clearing the bypass bit.
4. Wait for one core clock cycle.
5. Enable the divisor by setting the `DIVEN` bit.
6. Wait 15 core cycles before next activity.

Listing 23-2. VCO Programming: First Method

```
ustat2 = dm(PMCTL);
bit clr ustat2 PLLM63|PLLD16;     /* Clear the old multiplier
                                     and divider values */
bit set ustat2 DIVEN | PLLD4 |PLLM16;  /* set a multiplier of
                                        16 and a divider of 4 */
dm(PMCTL) = ustat2;
bit set ustat2 PLLBP;       /* Put PLL in bypass mode. */
bit clr ustat2 DIVEN;       /* clear the DIVEN bit */

dm(PMCTL) = ustat2;         /* The DIVEN bit should be cleared
                          while placing the PLL in bypass mode */
waiting_loop:
r0 = 4096;                 /* wait for PLL to lock at new rate
                             (requirement for VCO change) */
lcntr = r0, do pllwait until lce;
pllwait: nop;

ustat2 = dm(PMCTL);     /* Reading the PMCTL register value
                          returns the DIVEN bit value as zero */

bit clr ustat2 PLLBP;   /* take PLL out of Bypass, PLL is now at
                          new CCLK) */
dm(PMCTL) = ustat2;    /* The DIVEN bit should be cleared while
                            taking the PLL out of bypass mode */

lcntr = 15, do pllwait1 until lce;
pllwait1: nop;
```

Listing 23-3. VCO Programming: Second Method

```
ustat2 = dm(PMCTL);
bit clr ustat2 PLLM63| PLLD16;      /* Clear the old multiplier
                                       and divider values */
bit set ustat2 PLLBP | PLLD4 |PLLM16;  /* set a multiplier of
                                        16 and a divider of 4 */
dm(PMCTL) = ustat2;
waiting_loop:
r0 = 4096;                  /* wait for PLL to lock at new rate
                             (requirement for VCO change) */

lcntr = r0, do pllwait until lce;
pllwait: nop;
ustat2 = dm(PMCTL);         /* Reading the PMCTL register value
                              returns the DIVEN bit value as zero.
                              The DIVEN bit should be cleared while
                              taking the PLL out of bypass mode  */

bit clr ustat2 PLLBP;         /* take PLL out of Bypass,
                                PLL is now at new CCLK) */
dm(PMCTL) = ustat2;
bit set ustat2 DIVEN;         /* Enable the divider */
dm(PMCTL) = ustat2;
lcntr = 15, do pllwait1 until lce;
pllwait1: nop;
```

Back to Back Bypass

Use this steps and the example shown in Listing 23-4 if the application needs to re-enter the bypass mode.

1. Disable the bypass bit in the PMCTL register.
2. Wait 6 core clock cycles.
3. Enable the bypass bit.

Listing 23-4. Back to Back Bypass

```
ustat3 = dm(PMCTL);
bit clr ustat3 PLLBP;
dm(PMCTL) = ustat3;    /* PLLBP is cleared */
nop;nop;nop;nop;
ustat4 = dm(PMCTL);
bit set ustat4 PLLBP;
dm(PMCTL) = ustat4;    /* PLLBP is set */
```

24 SYSTEM DESIGN

This chapter discusses different processor reset methods, boot modes and pin multiplexing. In addition, information about high speed design is illustrated with some examples of supervisor circuits used in conjunction with the SHARC processor. These topics are located in the following sections.

- "Processor Reset" on page 24-3
- "Processor Booting" on page 24-7
- "Pin Multiplexing" on page 24-28
- "High Frequency Design" on page 24-34
- "System Components" on page 24-43

Before proceeding with this chapter it is recommended that you become familiar with the SHARC core architecture. This information is presented in *SHARC Processor Programming Reference*.

Features

The following list describes the features for reset and multiplexing.

- Five reset options: hardware, software, emulation, running reset and watchdog reset.
- Different master or slave boot mechanisms.
- Hardware and software reset for processor booting.

- Two pin multiplexing groups: core flag pins and external port pins.
- DAI/DPI units work together with multiplexing logic provides system design flexibility.

Thermal Diode

The processors incorporate thermal diode to monitor the die temperature. The thermal diode of is a grounded collector, PNP Bipolar Junction Transistor (BJT). For complete information on thermal diodes, see the "Thermal Characteristics" section of the processor-specific data sheet.

Pin Descriptions

Refer to the appropriate product data sheet for pin information, including package pinouts for the currently available package options.

Register Overview

The following registers are used for processor reset, booting, and pin multiplexing.

Software Reset Control Register (SYSCTL). Controls the software reset mechanism.

Running Reset Control Register (RUNRSTCTL). Controls the functionality of the $\overline{\text{RESETOUT}}$ pin as running reset input.

EP Control Register (EPCTL). Controls the memory chip selects for AMI on the external port memory space during boot.

EP DMA control register (DMACx). Controls the receive configuration for external boot DMA.

AMI Control Register (AMICTL1). Controls the AMI port configuration for external port boot mode.

Link Port Control Register 0 (LCTL0). Controls the receive DMA for link port mode during boot.

SPI Control Register (SPICTL). Controls the configuration for SPI as master or slave during SPI boot.

SPI DMA Control Register (SPIDMAC). Configures the SPI as receive DMA which generates an interrupt during boot.

SPI Slave Select Control Register (SPIFLGx). Controls the slave select configuration for SPI as master during SPI boot.

SPI Baudrate Register (SPIBAUD). Controls the SPICLK frequency for master mode during boot.

Processor Reset

After power-up, a $\overline{\text{RESET}}$ is required to place the processor into a known good state. Table 24-1 shows the differences between a hardware reset ($\overline{\text{RESET}}$ pin deasserted) or a software reset (setting bit 0 in the SYSCTL register) and gives an overview of the different reset methods.

Hardware Reset

All members of the SHARC processor family support the hardware reset controlled with the $\overline{\text{RESET}}$ pin. The deassertion of this pin enables the PLL and asserting it resets the PLL. In the time it takes the PLL to acquire lock (set to 4096 CLKIN cycles), the processor, internal memory, and the peripherals are held in reset. Upon completion of the 4096 CLKIN cycles, the chip is brought out of reset. This is indicated on the $\overline{\text{RESETOUT}}$ pin for the valid boot modes. For more information, see "Processor Booting" on page 24-7.

Table 24-1. Reset Function Overview

Reset Function	Hardware Reset	Software Reset	Running Reset
$\overline{\text{RESETOUT}}$ Pin	Output	Output	Input
$\overline{\text{RESETOUT}}$ Pulse	4096 CLKIN cycles asserted	2 PCLK cycles asserted	N/A
PLL	Yes	No	No
Core	Yes	Yes	Yes
Internal Memory[1]	No	No	No
Peripherals	Yes	Yes	Yes (except SDRAM/DDR2)
Booting	Yes	Yes	No
Power Management	Yes	No	No
Emulation Unit[2]	No	No	No

1 Internal memory array does not have reset. Only power up/down can change array contents, (or direct read/write by the core or DMA). However, if data exists in shadow FIFOs then that data is reset with any of the above resets. The logic outside the memory array is reset by all of the above three reset types, only the memory array contents remain unchanged.

2 There is an independent reset ($\overline{\text{TRST}}$) for the emulation interface. Enhanced Emulation (BTC) related logic is reset by the three resets types (HW Reset, SW Reset, Running Reset). Furthermore, no other part of the emulator is affected by the reset types. $\overline{\text{TRST}}$ resets the whole emulator function, including BTC.

Software Reset

In addition to the hardware reset, there is also support for a software reset, which is asserted by setting bit 0 of the SYSCTL register.

Running Reset

When running reset is asserted ($\overline{\text{RESETOUT}}$ pin acting as an input and asserted) and recognized, note the following.

- The core-PLL is NOT reset, and continues to run.

- Internal memory SRAM contents remain unaltered.
- The processor core and peripherals are reset exactly as if a Power-on (hardware) reset is asserted, except:
  - The SDRAM/DDR2 controllers continue to run and refresh as programmed.
  - The contents of external SDRAM/DDR2 are unaffected, and retain their values prior to a running reset.
  - A system boot is NOT initiated. Instead, the program counter is cleared and program execution begins from the very first location of program memory (from the reset interrupt vector table).

Running reset allows programs to:

- Execute self-modifying code that has previously overwritten existing code in external memory.
- Activate an external watchdog in cases where there is a malfunction or exception within a peripheral.
- Perform a context reset of the processor sufficient to restore the state, (in cases where a complete boot is not required).

The `RUNRSTCTL` register is reset only on assertion of a hardware reset, software reset, emulator reset, or by writing to the appropriate bits of the `RUNRSTCTL` register via software.

For emulation reset, see *SHARC Processor Programming Reference*, “JTAG” chapter.

System Considerations

It is important that an external 10 kΩ pull-up resistor is placed on the $\overline{\text{RESETOUT}}$ pin if it is intended to be used as an input for initiating a running reset as shown in Figure 24-1.

It is also extremely important to ensure that an external device, such as a micro controller, does not drive this signal during or after coming out of a power-on or hard-reset.

Figure 24-1 shows the active state of the pin during and after $\overline{\text{RESET}}$. The processor is actively driving this pin as an output. If the system uses an external host or micro controller to control running reset, ensure that the external device waits until the processor driver has been internally disabled (by writing to the RUNRSTCTL register) before actively driving this signal at $\overline{\text{RESET}}$. Connect the $\overline{\text{RESETOUT}}$ pin to an open-drain pin on the host side, or use an external three-state buffer.

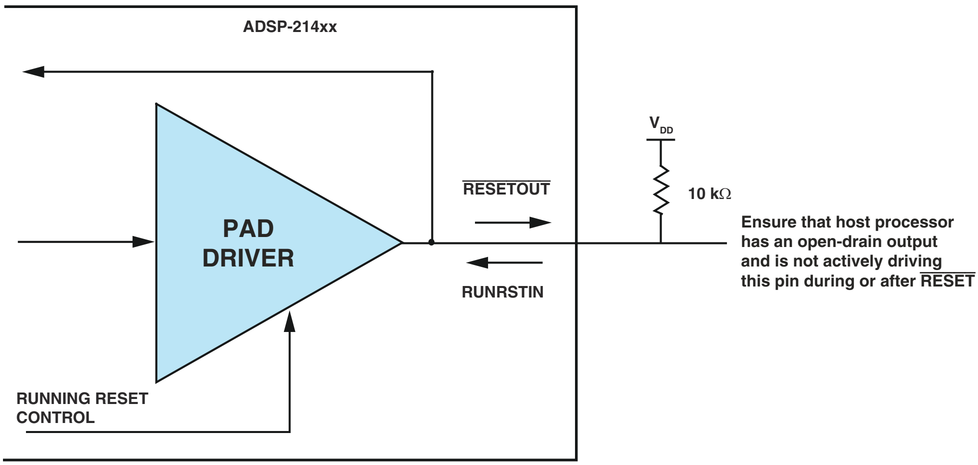

Figure 24-1. RESETOUT Pin Multiplexed with RUNRSTIN

There are several possible methods that can be used to implement running reset. The following illustrates one example of a running reset implementation involving an SHARC processor and a host processor.

External Host

In an AVR (audio-video receiver) system, a host microcontroller may communicate with the processor using the serial peripheral interface (SPI) or, if no SPI pins are available on the host device, it can use spare flag I/Os to connect with the SPI of a SHARC as shown in Figure 24-2. In this case, the host implements the SPI protocol on the port pins.

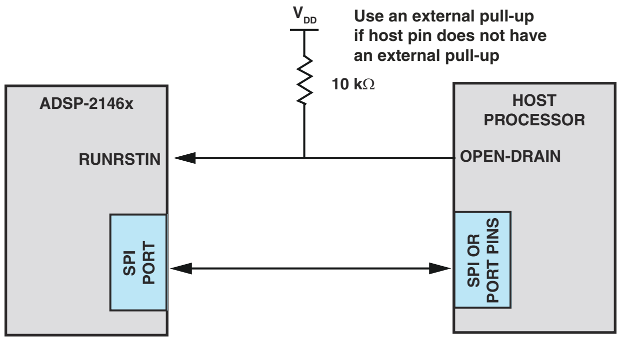

Figure 24-2. Example System Interface With an External Host

Processor Booting

When a processor is initially powered up, its internal SRAM is undefined. Before actual program execution can begin, the application must be loaded from an external non-volatile source such as flash memory or a host processor. This process is known as *bootstrap loading* or *booting* and is automatically performed by the processor after power-up or after a hard or software reset.

> ⓘ The SHARC processors don't have an on-chip boot ROM which controls the boot scenario. Instead a hard coded DMA uploads the boot kernel into the core before the application is booted.

Boot Mechanisms

In order to ensure proper device booting, the following hardware mechanisms are available on the processor.

- Peripheral boot configuration pins configure which peripheral boot stream is activated after power-up. Depending on model and package selection, 2 or 3 boot configuration pins are available. Refer to the product-specific datasheet for more information.
- Peripheral control and DMA parameter settings define the DMA channel which is started after $\overline{\text{RESETOUT}}$ is asserted based on the boot configuration pins.
- Peripheral interrupt is enabled after reset for the boot peripheral.
- During kernel load the core is put in IDLE. After the interrupt is generated the core jumps to reset location and starts kernel execution.

External Port Booting

The ADSP-214xx processors allow booting through the external port. The boot setting is configured through the boot configuration pins.

The asynchronous memory interface (AMI) supports an 8-bit user boot called AMI boot. Only the $\overline{\text{MS1}}$ signal is used for AMI (FLASH/EEPROM) booting. Table 24-2 shows the bit settings for AMI boot. These bits are described in detail in "AMI Control Registers (AMICTLx)" on page A-27.

After $\overline{\texttt{RESETOUT}}$ deasserted, the processor starts to drive:

- `ADDR23-0`
- Chip select $\overline{\texttt{MS1}}$ to the EPROM/FLASH
- $\overline{\texttt{AMI\_RD}}$ strobe with 23 `SDCLK` cycle wait states
- Read input data 7–0

The `ACK` pin is disabled during external port booting.

The received data streams of 4x8-bit data words are packed by the `AMIRX` buffer into 32-bit words least significant bit (LSB) first, and passed through the DMA's 6 deep external port buffer `DFEP0` into the internal memory (Figure 24-3).

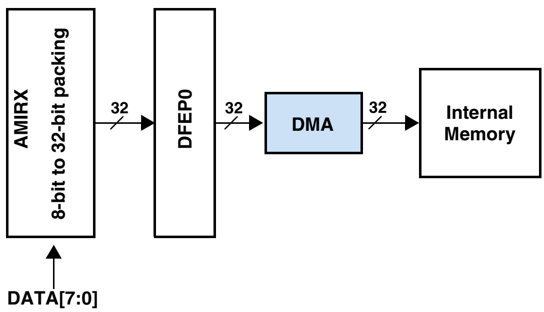

Figure 24-3. External Port Data Packing

The external port DMA channel 0 (`DMAC0`) is used when downloading the boot kernel information to the processor. At reset, the DMA parameter registers are initialized to the values listed in Table 24-4.

In this configuration, the loader kernel is read via DMA from the FLASH. If the application needs to speed-up read accesses, programs should change the wait states (`WS` bits, see Table 24-2) in the kernel file. After the kernel is executed, the new wait state settings are applied and processor booting continues.

Table 24-2. AMICTL1 Boot Settings (0x5C1)

Bit	Name	Setting
0	AMIEN	AMI enable (set = 1)
2–1	BW	Bus width = 8-bit (00)
3	PKDIS	Packing, 8-bit to 32-bit (cleared = 0)
4	MSWF	Most significant word first (cleared = 0)
5	ACKEN	ACK pin disabled (cleared = 0)
10–6	WS	23 wait state cycles = 10111
13–11	HC	Bus hold cycle at the end of write access = 000
16–14	IC	No bus idle cycle = 000
17	FLSH	Buffer holds data (cleared = 0)
20–18	RHC	Read hold cycle at the end of read access = 000
21	PREDIS	Disable Predictive Reads (cleared = 0)

Table 24-3. EPCTL Boot Settings (0xF0)

Bit	Name	Setting
0	B0SD	No SDRAM bank 0 (cleared = 0)
1	B1SD	No SDRAM bank 1 (cleared = 0)
2	B2SD	No SDRAM bank 2 (cleared = 0)
3	B3SD	No SDRAM bank 3 (cleared = 0)
5–4	EPBR	Rotating priority core vs. DMA (11)
7–6	DMAPR	Rotating priority EPDMA ch0 vs. EPDMA ch1 (11)
10–8	FRZDMA	No DMA freezing (00)
14–12	FRZCR	No core freezing (00)
18–15	DATE	No pack mode (0000) (ADSP-2147x/2148x only)
21–19	FRZSP	No SPORT DMA freezing (000)

Table 24-4. DMAC0 Boot Settings (0x1000001)

Bit	Name	Setting
0	DMAEN	DMA enabled (set = 1)
1	TRAN	Write to internal memory (cleared = 0)
2	CHEN	No DMA chaining (cleared = 0)
3	DLEN	No delay line DMA (cleared = 0)
4	CBEN	No circular DMA (cleared = 0)
5	DFLSH	Disabled (cleared = 0)
7	WRBEN	Disabled (cleared = 0)
8	OFCEN	Disabled (cleared = 0)
9	TLEN	Disabled (cleared = 0)
12	INTIRT	Disabled (cleared = 0)
17–16	DFS	Status (cleared = 00)
20	DMAS	Status (cleared = 0)
21	CHS	Status (cleared = 0)
22	TLS	Status (cleared = 0)
23	WBS	Status (cleared = 0)
24	EXTS	External access pending (set = 1)
25	DIRS	Status (cleared = 0)

Table 24-5. Parameter Initialization for External Port Boot

Parameter Register	Initialization Value	Comment
IIEP0	0x92000	Start of block 0 (IVT_START_ADDRESS 2 Column)
IMEP0	0x1	
ICEP0	0x180	384 × 32-bit transfers
EIEP0	0x4000000	External memory select 1 start address

Table 24-5. Parameter Initialization for External Port Boot (Cont'd)

Parameter Register	Initialization Value	Comment
EMEP0	0x1	
ECEP0	0x180	384 × 32-bit transfers

SPI Port Booting

The SHARC processors support booting from a host processor using SPI slave mode or booting from an SPI flash, SPI PROM, or a host processor via SPI master mode. Both SPI boot modes (master and slave) support 8-, 16-, or 32-bit SPI devices. For bit settings, see the product-specific processor data sheet.

In both (master and slave) boot modes, the LSBF format is used and SPI mode 3 is selected (clock polarity and clock phase = 1). Both SPI boot modes use default routing with the DPI pin buffers For more information, see "DPI Default Routing" on page 10-28.

Master Boot Mode

In master boot mode, the processor initiates the booting operation by:

1. Activating the SPICLK signal and asserting the SPI_FLG0_0 signal to the active low state.

2. Writing the read command 0x03 and 24-bit address 0x000000 to the slave device as shown in Figure 24-4.

Master boot mode is used when the processor is booting from an SPI-compatible serial PROM (16-bit address), serial FLASH (24-bit address), or slave host processor (no address). The specifics of booting from these devices are discussed individually.

SPI master booting uses the default bit settings shown in Table 24-7.

Table 24-6. SPIDMAC Master/Slave Boot Settings (0x7)

Bit	Setting	Comment
SPIDEN	Set (= 1)	SPI DMA
SPIRCV	Set (= 1)	SPI receive
INTEN	Set (= 1)	SPI interrupt
SPICHEN	Cleared (= 0)	SPI DMA chaining
FIFOFLSH	Cleared (= 0)	FIFO flush
INTERR	Cleared (= 0)	SPI DMA error interrupts

Table 24-7. SPICTL Master Boot Settings (0x5D06)

Bit	Setting	Comment
SPIEN	Set (= 1)	SPI enabled
SPIMS	Set (= 1)	Master device
MSBF	Cleared (= 0)	LSB first
WL	10	32-bit SPI receive shift register word length
DMISO	Cleared (= 0)	MISO enabled
SENDZ	Set (= 1)	Send zeros
SPIRCV	Set (= 1)	Receive DMA enabled
CLKPL	Set (= 1)	Active low SPI clock
CPHASE	Set (= 1)	Toggle SPICLK at the beginning of the first bit

The SPI DMA channel is used when downloading the boot kernel information to the processor. At reset, the DMA parameter registers are initialized to the values listed in Table 24-8.

Table 24-8. Parameter Initialization Values for SPI Master Boot

Parameter Register	Initialization Value	Comment
SPIBAUD	0x64	SPICLK = f_{PCLK}/100
SPIFLG	0xFE01	SPI_FLG0_O used as slave-select
IISPI	0x92000	Start of block 0 (IVT_START_ADDRESS 2 Column)
IMSPI	0x1	32-bit data transfers
CSPI	0x180	384 × 32-bit transfers

Master Read Command

The transfer is initiated by the transferring the necessary header information on the MOSI pin (consisting of the read opcode and the starting 24-bit address of the block to be transferred, which is usually all zeros). The read opcode is fixed as 0xC0 (LSBF format) and is 8 bits long. The 8-bits that are received following the read opcode should be programmed to 0xA5. If the 8-bits are different from 0xA5 the master boot transfer is aborted. The transfer continues until 384 x 32-bit words have been transferred which may correspond to the loader program (just as in the slave boot mode).

The loader tool automatically includes the SPI master header information (0xA5).

1. Default state of SPICLK signal high (out of reset).
2. Deasserting the SPI_FLG0_O signal (chip select) to the active low state and toggling the SPICLK signal.
3. Reading the read command 0x03 (MSBF format to match the LSBF format) and address 0x00 from the slave device.

Unlike previous SHARC processors, the MOSI pin (DPI pin 01) is three-stated for SPI master boot mode during reset. It is recommended to either leave the SPICLK signal (DPI3) floating or add an external pull-up resistor. The chip select signal going low a bit before this erroneous rising edge may lead to boot failure as the read command may not be recognized by the FLASH device properly. The is because the reset configuration of the SPI port for SPI master boot is CLKPL = 1 which means that the SPICLK signal should be HIGH when idle.

For more detailed information on the SPI master read command refer to the tools Loader manual.

Slave Boot Mode

In slave boot mode, the host processor initiates the booting operation by activating the SPICLK signal and asserting the $\overline{\text{SPI\_DS\_I}}$ signal to the active low state. The 256-word kernel is loaded 32 bits at a time, through the SPI receive shift register (RXSR). To receive 256 instructions (48-bit words) properly, the SPI DMA initially loads a DMA count of 0x180 (384) 32-bit words, which is equivalent to 0x100 (256) 48-bit words.

Note that for SPI slave boot $\overline{\text{SPI\_DS\_I}}$ should only be asserted after $\overline{\text{RESETOUT}}$ has deasserted.

When in SPI slave booting mode, the SPI_DS_I input signal is controlled by the SPI host to initiate the boot transfers as shown in Table 24-9.

Since the SPI host initiates the transfers, a handshake between master and slave is required for synchronization. One possible solution is to use the slave's SPI_MISO_O signal as handshake signal. If a pause is required, the slave transmits zeros or ones to the master. Another solution is to connect this signal to the master's flag input to generate an interrupt for the same purpose.

Table 24-9. SPICTL Slave Boot Settings (0x4D22)

Bit	Setting	Comment
SPIEN	Set (= 1)	SPI enabled
SPIMS	Cleared (= 0)	Slave device
MSBF	Cleared (= 0)	LSB first
WL	10, 32-bit SPI	Receive shift register word length
DMISO	Set (= 1) MISO	MISO disabled
SENDZ	Cleared (= 0)	Send last word
SPIRCV	Set (= 1)	Receive DMA enabled
CLKPL	Set (= 1)	Active low SPI clock
CPHASE	Set (= 1)	Toggle SPICLK at the beginning of the first bit

The SPI DMA channel is used when downloading the boot kernel information to the processor. At reset, the DMA parameter registers are initialized to the values listed in Table 24-10.

Table 24-10. Parameter Initialization for SPI Slave Boot

Parameter Register	Initialization Value	Comment
SPIDMAC	0x0000 0007	Enable receive, interrupt on completion
IISPI	0x92000	Start of block 0 (IVT_START_ADDRESS 2 Column)
IMSPI	0x1	32-bit data transfers
CSPI	0x180	384 × 32-bit transfers

SPI Boot Packing

In all SPI boot modes, the data word size in the shift register is hardwired to 32 bits. Therefore, for 8- or 16-bit devices, data words are packed into the shift register to generate 32-bit words least significant bit (LSB) first, which are then shifted into internal memory. The relationship between the 32-bit words received into the RXSPI register and the instructions that need to be placed in internal memory is shown in the following sections.

For more information about 32- and 48-bit internal memory addressing, see the "Memory" chapter in *SHARC Processor Programming Reference*.

As shown in Figure 24-4, two words shift into the 32-bit receive shift register (RXSR) before a DMA transfer to internal memory occurs for 16-bit SPI devices. For 8-bit SPI devices, four words shift into the 32-bit receive shift register before a DMA transfer to internal memory occurs.

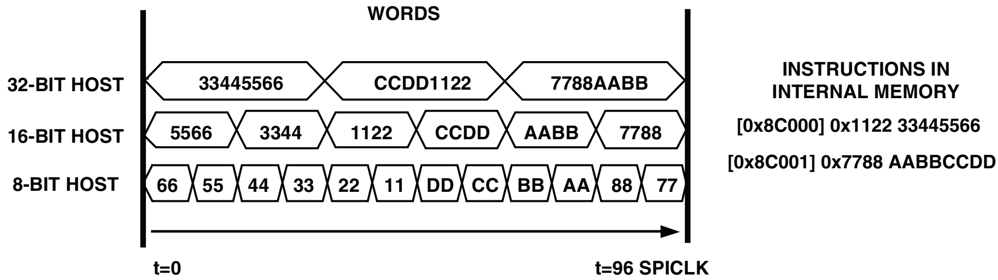

Figure 24-4. Instruction Packing for Different Hosts

When booting, the processors expect to receive words into the RXSPI register seamlessly. This means that bits are received continuously without breaks. For more information, see "Core Transfers" on page 16-25. For different SPI host sizes, the processor expects to receive instructions and data packed in a least significant word (LSW) format.

Figure 24-4 shows how a pair of instructions are packed for SPI booting using a 32-, 16-, and an 8-bit device. These two instructions are received as three 32-bit words.

The following sections examine how data is packed into internal memory during SPI booting for SPI devices with widths of 32, 16, or 8 bits.

32-Bit SPI Packing

Figure 24-5 shows how a 32-bit SPI host packs 48-bit instructions executed at PM addresses PMaddr0 and PMaddr1. The 32-bit word is shifted to internal program memory during the 256-word kernel load.

The following example shows a 48-bit instruction executed:

```
[PMaddr0] 0x112233445566
[PMaddr1] 0x7788AABBCCDD
```

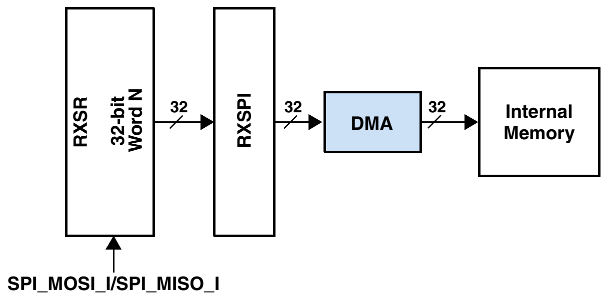

Figure 24-5. 32-Bit SPI Master/Slave Packing

The 32-bit SPI host packs or prearranges the data as:

SPI word 1=	0x33445566
SPI word 2 =	0xCCDD1122
SPI word 3 =	0x7788AABB

The initial boot of the 256-word loader kernel requires a 32-bit host to transmit 384 x 32-bit words. The SPI DMA count value of 0x180 is equal to 384 words.

16-Bit SPI Packing

Figure 24-6 shows how a 16-bit SPI host packs 48-bit instructions at PM addresses PMaddr0 and PMaddr1. For 16-bit hosts, two 16-bit words are packed into the shift register to generate a 32-bit word. The 32-bit word shifts to internal program memory during the kernel load.

The following code shows a 48-bit instruction executed:

```
[PMaddr0] 0x112233445566
[PMaddr1] 0x7788AABBCCDD
```

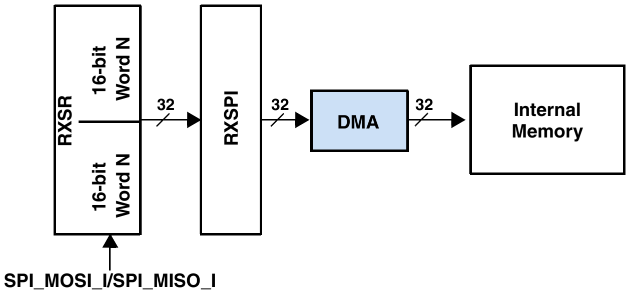

Figure 24-6. 16-Bit SPI Master/Slave Packing

The 16-bit SPI host packs or prearranges the data as:

SPI word 1 =	0x5566
SPI word 2 =	0x3344
SPI word 3 =	0x1122
SPI word 4 =	0xCCDD
SPI word 5 =	0xAABB
SPI word 6 =	0x7788

The initial boot of the 256-word loader kernel requires a 16-bit host to transmit 768 16-bit words. Two packed 16-bit words comprise the 32-bit word. The SPI DMA count value of 0x180 is equivalent to 384 words. Therefore, the total number of 16-bit words loaded is 768.

8-Bit SPI Packing

Figure 24-7 shows how an 8-bit SPI host packs 48-bit instructions executed at PM addresses PMaddr0 and PMaddr1. For 8-bit hosts, four 8-bit words pack into the shift register to generate a 32-bit word. The 32-bit word shifts to internal program memory during the load of the 256-instruction word kernel.

The following code shows a 48-bit instruction executed:

```
[PMaddr0] 0x112233445566
[PMaddr1] 0x7788AABBCCDD
```

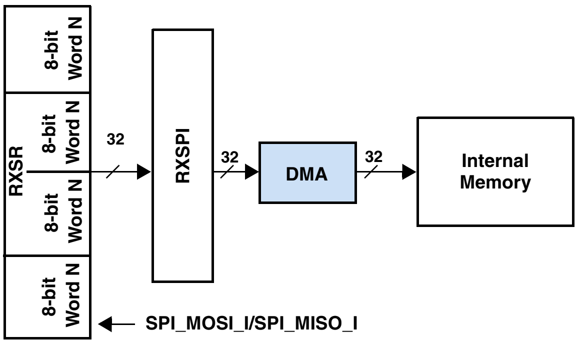

Figure 24-7. 8-Bit SPI Slave Packing

The 8-bit SPI host packs or prearranges the data as:

SPI word 1 =	0x66	SPI word 7 =	0xDD
SPI word 2 =	0x55	SPI word 8 =	0xCC
SPI word 3 =	0x44	SPI word 9 =	0xBB
SPI word 4 =	0x33	SPI word 10 =	0xAA
SPI word 5 =	0x22	SPI word 11 =	0x88
SPI word 6=	0x11	SPI word 12=	0x77

The initial boot of the 256-word loader kernel requires an 8-bit host to transmit 1536 x 8-bit words. The SPI DMA count value of 0x180 is equal to 384 words. Since one 32-bit word is created from four packed 8-bit words, the total number of 8-bit words transmitted is 1536.

Link Port Booting

Booting is supported through link port 0. The BOOT_CFGx values for selecting link port boot are located in the product-specific data sheet.

The booting procedure is the same as any other boot mode. The acknowledge signal (LACK0) is asserted at $\overline{\text{RESET}}$ since the link port is configured as a receiver. The host initiates the transfer by toggling the link port clock (LCLK0). Boot data is shifted in 8-bits every clock cycle through the LDAT0x pins. The received data streams of 4 x 8-bit is packed by the 2 deep RXLB0 buffer into 32-bit words, least significant bit (LSB) first, and passed into the internal memory (Figure 24-8). Once the DMA is completed, a link port 0 interrupt (P1I) occurs. If BOOT_CFGx is programmed for link port boot, P1I is programmed as the link port 0 interrupt at reset and the interrupt is unmasked at reset. Otherwise, P1I is programmed as the $\overline{\text{SPIHI}}$ interrupt at reset.

For link port boot, LCLK0 should only be asserted after $\overline{\text{RESETOUT}}$ has deasserted.

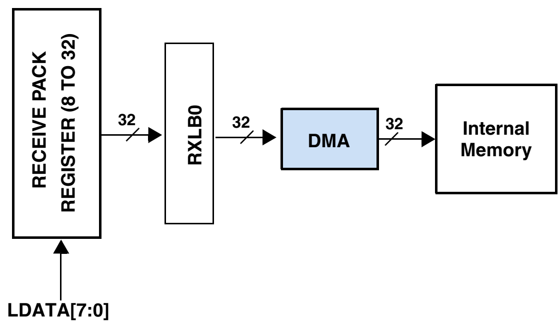

Figure 24-8. Link Port Data Packing

Table 24-11 shows the link port control settings after reset.

Table 24-11. LCTL0 Boot Settings (0x403)

Bit	Name	Setting
0	LEN	Link port enabled (set = 1)
1	LDEN	DMA enabled (set = 1)
2	LCHEN	DMA Chaining (cleared = 0)
3	LTRAN	Receive operation (cleared = 0)
7	BHD	Buffer hang disabled (cleared = 0)
8	LTRQ_MSK	LP transmit request mask (cleared = 0)
9	LRRQ_MSK	LP receive request mask (cleared = 0)
10	DMACH_IRPT_MSK	LP DMA channel interrupt unmask (P1I) (set = 1)
11	LPIT_MSK	LP Invalid transmit mask (cleared = 0)
12	TXFR_DONE_MSK	External transfer done interrupt mask (cleared = 0)

The DMA parameters for the Link Port0 channel are configured as shown in Table 24-12.

Table 24-12. Parameter Initialization for Link Boot

Parameter Register Elf splitter	Initialization Value	Comment
IILP0	0x92000	Start of block 0 (IVT_START_ADDRESS 2 Column)
IMLP0	0x1	32-bit data transfers
ICLP0	0x180	384 × 32-bit transfers

Kernel Boot Time

This section illustrates the minimum required booting time for the kernels (provided by the tools). There are five timing windows which describe together the entire boot process shown in the list below and Table 24-13.

1. $\overline{\text{RESET}}$ to $\overline{\text{RESETOUT}}$ (core is in reset)
2. $\overline{\text{RESETOUT}}$ to chip select boot source (activate the boot DMA)
3. Load Kernel DMA (256 words)
4. Load application (user dependent)
5. Load IVT (256 words)

Table 24-13. Boot Times

Boot Mode	$\overline{\text{RESET}}$ to $\overline{\text{RESETOUT}}$	$\overline{\text{RESETOUT}}$ to Boot Chip Select	Kernal DMA (256 Words)	Comment
SPI Master	4096 CLKIN	1 PCLK	(I/O × PCLK ÷ 100 + 4 × PCLK) × N	N=384, 768 or 1536 for I/O = 32, 16 or 8
SPI Slave	4096 CLKIN	Host drives signal	(I/O × PCLK ÷ 100 + 2 × PCLK) × N	N=384, 768 or 1536 for I/O = 32, 16 or 8
External Port	4096 CLKIN	5 PCLK	24 × SDCLK × 1536	
Link Port0	4096 CLKIN	Host drives signal	LCLK0 × 1536	

The complete time for booting can be estimated by adding all 5 timing windows. Loading Kernel and Loading IVT both have the same size, however the default access time (wait states) for the IVT loading can be changed in the kernel by the user.

ROM Booting

There are two access types (modes) available for ROM booting: secured and non secured modes which are described below.

Secured ROM (hardware security switch = 1). In this mode:

- BOOTCFG2-0 pins are ignored.
- Emulation is enabled only when the user enters a valid key.
- IIVT is placed into the internal ROM. It can be changed to the internal RAM by setting IIVT bit of SYSCTL register.
- Code always executes from internal ROM.

Non Secured ROM (hardware security switch = 0). In this mode:

- BOOTCFG2-0 pins select the booting modes.
- Emulation is always enabled.
- IIVT is placed into the internal RAM except for the case where BOOTCFG2-0 = 011.

Programming Model

This section describes the operation of the boot process. This process is accomplished using the default loader kernel (Visual DSP Tools) to generate the boot stream. For more details, refer to the loader source files.

Running Reset

Using the SPI protocol with additional control words and commands, running reset can become an addition command from the host or from the processor as described in the following procedure.

1. The host initiates a running reset by informing the processor over the command interface.
2. The processor receives the command and completes any unfinished work which may also include writing to the RUNRSTCTL register.
3. Wait at least 5 CCLK cycles to ensure that the pin is configured as an input.
4. When the processor is ready to accept the running reset, it signals the host over the command interface.
5. The host drives the running reset input into the processor.

Running The Boot Kernel

The following sections provide information on the use of the boot kernal with the SHARC processors.

Loading the Boot Kernel Using DMA

1. At reset, the processor is hardwired (using the boot configuration pins) to load 256 x 48-bit instruction words via a DMA starting at IVT_START_ADDRESS.
2. The sequencer is put into IDLE until the boot interrupt occurs.

Executing the Boot Kernel

1. The DMA completes (counter zero) and the interrupt associated with the peripheral that the processor is booting from is activated.
2. The processor jumps to the applicable interrupt vector location and executes the RTI instruction located there (only).

If using your own loader kernel, you must ensure that the RTI instruction points to the IVT location of the boot peripheral.

Loading the Application

1. Once the kernel is executed (initialization of some core and external peripheral registers and such as AMI or SDRAM), the kernel prepares a DMA for further data.
2. After this the DMA starts and the core waits in IDLE until an interrupt is generated.
3. The kernel then reads the header data from a memory scratch location, decodes the header and configures a loop which loads all of the header's corresponding data.
4. Step 3 is repeated until all headers are executed.

Loading the Application's Interrupt Vector Table

1. The last header is recognized by the kernel indicating that booting has nearly finished.
2. The kernel prepares a 256 x 48-word DMA starting at `IVT_START_ADDRESS`.

   This overrides the kernel with the application's IVT. However, the application needs to temporarily include the RTI instruction at the peripheral interrupt address, allowing a return from interrupt. Moreover, the last instruction in the final routine is a jump (db) including an IDLE.
3. The RTI instruction overrides the IVT address where user code is stored.

While both DMA types ("Loading the Boot Kernel Using DMA" and "Loading the Application's Interrupt Vector Table") seem similar, loading the kernel is accomplished using hardware while loading the IVT is accomplished using software.

It is very important to match the dedicated kernel to the dedicated boot type (for example SPI kernel and SPI boot type) in the elf-loader property page. If this is not done, the RTI instruction (in "Loading the Application's Interrupt Vector Table") will not be placed at the correct address. This causes execution errors.

Starting Program Execution

The processed interrupt returns the sequencer to the reset location by performing the two following steps.

1. Overriding the RTI instruction with user code.
2. Starting program execution from the reset location.

For other details relating to processor booting, see the boot loader source files that ship with the CrossCore or VisualDSP++ tools.

Memory Aliasing in Internal Memory

The boot loader takes advantage of memory aliasing which is essential to understand the boot mechanisms. For information on memory aliasing, see *SHARC Processor Programming Reference*, "Memory" chapter.

During the boot process, word packing (for example 8 to 32-bit) is performed over the SPI. In other words, the kernel is not loaded directly with 256 x 48-bit words, instead it is loaded with 384 x 32-bit 'packed words' (2-column access). The same physical memory for instruction boot is loaded via DMA in normal word (NW) 2 column. However, after booting the same physical memory region is fetched by the sequencer in NW 3-column. For example the loader kernel itself has a NW 2 columns count of 256 x 3/2 = 384 words but the kernel is executed with 256 instruction fetches.

Note that the interrupt vector table addresses are defined as:

IVT_Start_Addr = 0x8C000 and IVT_End_Addr = 0x8C0FF.

Pin Multiplexing

The SHARC processors provide extensive functionality using a low pin count (reducing system cost). They do this through extensive use of pin multiplexing. The following sections provide information on this feature. Although the processors have the efficient and flexible DAI and DPI routing options, there are also I/O pins which are shared by some peripherals. The following sections discusses these options.

> ⓘ On the ADSP-2146x processors the AMI and DDR2 interfaces are completely independent (not multiplexed). Only the AMI controller address/memory selects and data pins are shared and therefore all pins discussed in this section refer to the AMI controller. Therefore, the naming conventions DATAx and ADDRx used below refer to the AMI_DATAx and AMI_ADDRx pins for ADSP-2146x processors.

Core FLAG Pins Multiplexing

This module also includes the multiplexers of the FLAG0-3 pins shown in Figure 24-9. The FLAG0-2 pins can act as core FLAGS0-2 or $\overline{\text{IRQ0-2}}$, or a memory select $\overline{\text{MS2}}$ (FLAG2 pin) and the FLAG3 pin can act as a core FLAG3 or the TMREXP signal of the core timer or as a memory select $\overline{\text{MS3}}$.

> ⓘ Flag pins (FLG3-0) are connected as input after reset.

If more than four flags are required, they can multiplexed using the external port pins in the SYSCTL register or the DPI pins in the DPI registers.

For a detailed flag description refer to *SHARC Processor Programming Reference*. Table 24-14 provides information on FLAG function based on the settings of the memory select enable, the flag timer expired and the FLAG2 interrupt bits in the system control register.

Table 24-14. Flag 3–2 Truth Table (SYSCTL Register)

MSEN Bit	TMREXPEN Bit	IRQ2EN Bit	FLAG3 Function	FLAG2 Function
0	0	0	FLAG3	FLAG2
0	0	1	FLAG3	$\overline{\text{IRQ2}}$
0	1	0	TMREXP	FLAG2
0	1	1	TMREXP	$\overline{\text{IRQ2}}$
1	0	0	$\overline{\text{MS3}}$	$\overline{\text{MS2}}$

Backward Compatibility

The FLAG/$\overline{\text{IRQ}}$ (0, 1, 2, 3) pins retain their old functionality and programming. No changes are required for old programs. The select lines for multiplexes are controlled by the SYSCTL register. For more information, see “System Control Register (SYSCTL)” on page A-5.

External Port Pin Multiplexing

Various peripherals use the external port for off-chip communication. These peripherals use multiplexed I/O pins and have the (functions) shown:

- External Port (AMI/SDRAM/DDR2)
- PDAP (input)
- FLAGs (I/O)
- PWM channels (output)

Backward Compatibility

The multiplexing scheme is not backward compatible with previous SHARC processors. On previous SHARC processors only the external port data pins are multiplexed. With the ADSP-214xx processors, address and data pins of the external port are multiplexed.

Multiplexed External Port Pins

The external port address and data pins are used to multiplex the external port interface with other peripherals. Table 24-15 provides the pin settings.

Table 24-15. EPDATA Truth Table (SYSCTL Register)

EPDATA	ADDR23–8	ADDR7–0	DATA7–0
000	ADDR23–0		DATA7–0
001	Reserved		
010	Reserved		
011	FLAGS/PWM15–0[1]	FLAGS15–0	
100	Reserved		
101	PDAP (DATA + CTRL)		FLAGS7–0
110	Reserved		
111	Three-state all pins		

1 These signals can be FLAGS or PWM or mix of both but can be selected only in groups of four. Their functionality is decided by the bits FLAGS/PWM_SEL of SYSCTL register.

Table 24-15 shows the following options.

- The FLAG15-0 signals can be mapped to the lower set of eight data pins (DATA7-0) and lower eight address pins (ADDR7-0) such that FLAG7-0 signals map to DATA7-0 and FLAG15-8 signals map to ADDR7-0 respectively (EPDATA = 011).

- FLAGS/PWM can be mapped (in groups of four) to the upper 16 address pins (ADDR23-8) such that
  FLAG3-0/PWM3-0 signals map to ADDR11-8,
  FLAG7-3/PWM7-3 signals map to ADDR15-12,
  FLAG11-8/PWM11-8 signals map to ADDR19-16,
  FLAG15-12/PWM15-12 signals map to ADDR23-20 respectively (EPDATA = 011).
- PDAP data/control can be completely moved to external port address pins (ADDR23-0). In this mode,
  PDAP_DATA19-0 input signals are mapped to ADDR23-4,
  PDAP_HOLD input signal is mapped to ADDR3,
  PDAP_CLK input signal is mapped to ADDR2, and
  PDAP STROBE output signal is mapped to ADDR0.

Parallel Connection of Flag Pins via External Port and DPI Pins

The various external port multiplexing (shown in Figure 24-9 on page 24-33) and DPI routing options allow situations where the flag direction paths from the core to the external port or DPI pins operates in parallel as described below.

For FLAG3–0

- In output mode, if the same flag is mapped to both external port pins and FLAG3-0 pins, then the output is driven to both pins.
- In input mode, if the same flag is mapped to both external port pins and FLAG3-0 pins, then the input from external port pins has priority.

For FLAG15–4

- In output mode, if the same flag is mapped to both external port pins and DPI pins, then the output is driven from both pins.

- In input mode, if the same flag is mapped to both external port pins and DPI pins, then the input from the external port pins has priority.
- In input mode, if the same flags are mapped to both the upper AMI (ADDR23-8) and lower AMI (ADDR7-0, DATA7-0) pins, then the input from lower AMI pins have priority.

The FLAG15-4 SRU2 connections are shown in Table 24-16.

Table 24-16. FLAG SRU2 Signal Connections

FLAG Source	DPI Connection	FLAG Destination
	Group A	FLAG15–4_I
FLAG15–4_O	Group B	
FLAG15-4_PBEN_O	Group C	

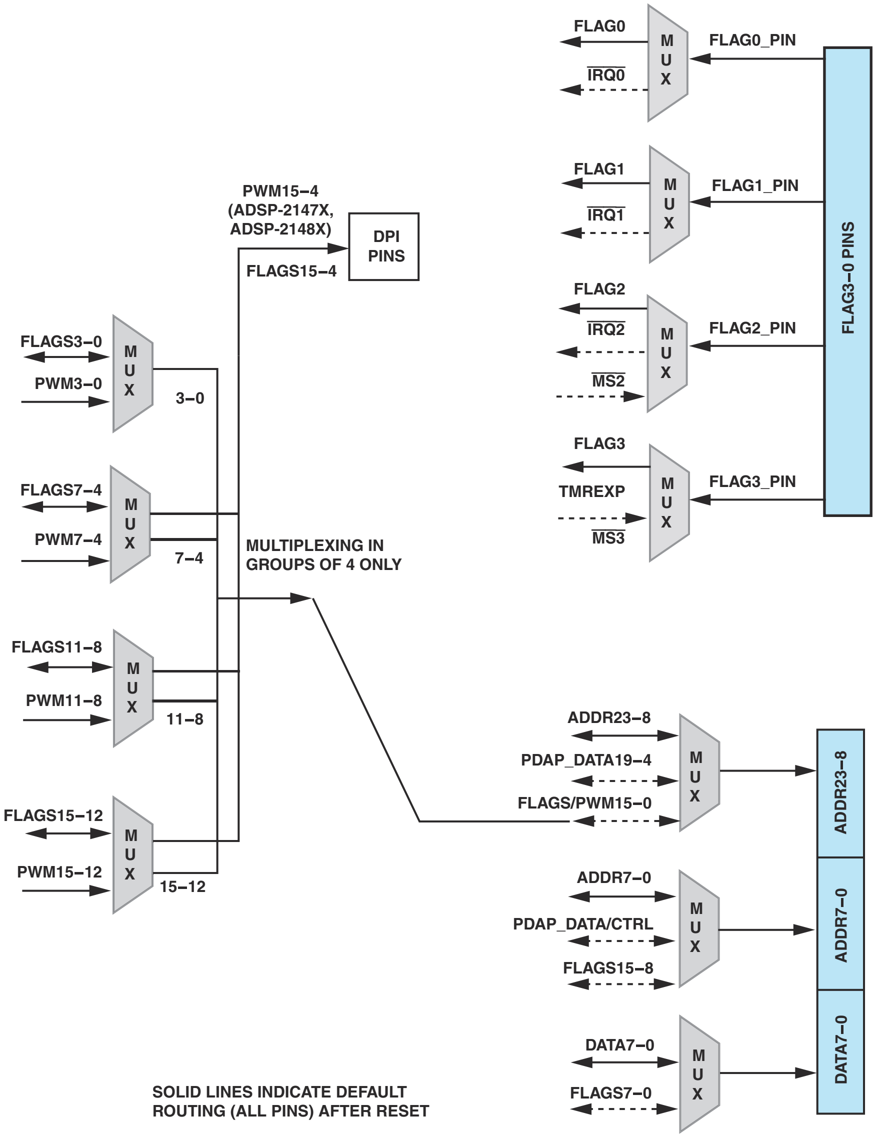

Figure 24-9. Pin Multiplexing

Processor Identification Register

The SHARC processor models have a memory-mapped version control register, `REVPID`, shown in Figure 24-10 and described in Table 24-17, that identifies the processor model and its silicon revision number.

For processor identification via the JTAG instruction (`IDCODE`) see *SHARC Processor Programming Reference*.

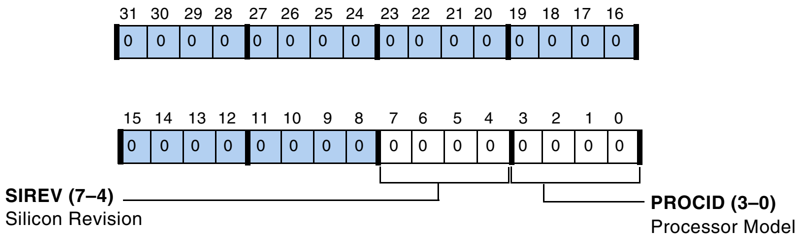

Figure 24-10. REVPID Register

Table 24-17. REVPID Register Bit Descriptions (RO)

Bit	Name	Description
3–0	PROCID	**Processor Model.** The processor model is shown below. 0101 = ADSP-2146x SHARC products 0110 = ADSP-2148x SHARC products 0111 = ADSP-2147x SHARC products
7–4	SIREV	**Silicon Revision.** For the silicon revision number bits (7–4), refer to the processor-specific anomaly sheet.

High Frequency Design

Because the processor must be able to operate at very high clock frequencies, signal integrity and noise problems must be considered for circuit board design and layout. The following sections discuss these topics and suggest various techniques to use when designing and debugging target systems.

Circuit Board Design

The processor is a CMOS device. It has input conditioning circuits which simplify system design by filtering or latching input signals to reduce susceptibility to glitches or reflections.

The following sections describe why these circuits are needed and their effect on input signals.

Clock Input Specifications and Jitter

The clock input signal must be free of ringing and jitter. Clock jitter can easily be introduced into a system where more than one clock frequency exists. Jitter should be kept to an absolute minimum. High frequency jitter on the clock to the processor may result in abbreviated internal cycles.

Keep the portions of the system that operate at different frequencies as physically separate as possible. The clock supplied to the processor must have a maximum rise time and must meet or exceed a high and low voltage of V_{IH} and V_{IL}, respectively.

Refer to the appropriate product data sheet for exact specifications.

RESETOUT

Circuit boards should have a test pad for the $\overline{\texttt{RESETOUT}}$ pin This pin can be used as handshake signal for booting or as clock out (`CLKIN` frequency) for a debug aid to verify the processor is active and running.

Input Pin Hysteresis

Hysteresis (shown in Figure 24-11) is used on all SHARC input signals. Hysteresis causes the switching point of the input inverter to be slightly above 1.4 V (VT) for a rising edge (VT+) and slightly below 1.4 V for a falling edge (VT–). The value of the hysteresis is approximately ± 100 mV.

Refer to the appropriate product data sheet for exact specifications.

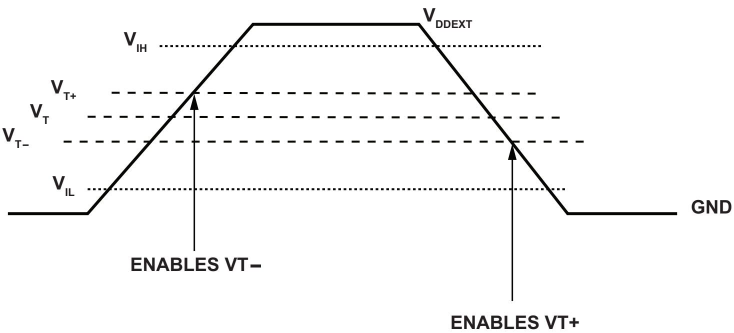

Figure 24-11. Input Pin Hysteresis

The hysteresis is intended to prevent multiple triggering of signals that are allowed to rise slowly, as might be expected for example on a reset line with a delay implemented by an RC input circuit. Hysteresis is not used to reduce the effect of ringing on processor input signals with fast edges, because the amount of hysteresis that can be used on a CMOS chip is too small to make a difference. The small amount of hysteresis allowed is due to restrictions on the tolerance of the V_{IL} and V_{IH} TTL input levels under worst-case conditions.

Clock and Control Signal Transitions

All clocks and control signals MUST transition between V_{IL} and V_{IH} (or V_{IH} and V_{IL}) in a monotonic manner.

Pull-Up/Pull-Down Resistors

The pin descriptions in the product-specific data sheets includes recommendations on how to handle pins on interfaces that are disabled or for unused pins on interfaces that are enabled. Generally, if internal pull-ups (IPU) or pull-downs (IPD) are included, the pins can be left floating. Any pin that is output only can always be left floating.

If internal pull-ups and pull-downs are not included or disabled, pins can normally still be floated with no functional issues for the device. However, this may allow additional leakage current.

Although the recommendations normally indicate using external pull-up resistors, pull-down resistors can also be used. The leakage is the same whether pull-ups or pull-downs are used. Connections directly to power or ground can be used only if the pins can be guaranteed to never be configured as outputs.

Memory Select Pins

When the multiplexed memory selects, $\overline{\texttt{MS3-2}}$, are enabled as outputs, the pull-up resistors are automatically enabled. For example, if $\overline{\texttt{MS2}}$ and $\overline{\texttt{MS3}}$ are used, they require that stronger external pull-up resistors are connected. For more details on resistor values, refer to the product-specific data sheet.

Edge-Triggered I/O

It is recommended that GPIO output pins that are used to drive an edge-sensitive signal like an interrupt ($\overline{\texttt{IRQ2-0}}$, DAI/DPI pins) have series termination resistors to prevent glitches on the signal transitions. It is equally important that GPIO inputs that are edge-sensitive be driven from sources that have series termination resistors. The values for the series resistor can be determined by simulating with the IBIS models. These models can be found on the Analog Devices web site.

Asynchronous Inputs

The processor has several asynchronous inputs such as $\overline{\texttt{IRQ2-0}}$, FLAG3-0, ACK and the DAI/DPI pins and reset inputs $\overline{\texttt{RESET}}$, $\overline{\texttt{TRST}}$, running reset which can be asserted in arbitrary phase to the reference clocks. The processor synchronizes the reset inputs to the CLKIN input while the peripheral inputs are synchronized to the PCLK prior to recognizing them.

The delay associated with recognition is called the synchronization delay. Any asynchronous input must be valid prior to the recognition point in a particular cycle. If an input does not meet the setup time on a given cycle, it may be recognized in the current cycle or during the next cycle.

To ensure recognition of an asynchronous input, it must be asserted for at least one PCLK cycle plus setup and hold time, except for $\overline{\text{RESET}}$, which must be asserted for at least four CLKIN processor cycles. The minimum time prior to recognition (the setup and hold time) is specified in the appropriate product data sheet.

Decoupling and Grounding

Designs should use an absolute minimum of four bulk capacitors (2 × 10 µF for V_{DDINT} and 2 × 10 µF for V_{DDEXT}). Furthermore a minimum of 20 × 10nF ceramic bypass capacitors (5 per chip corner for V_{DDINT} and V_{DDEXT}).

Capacitors type, value and placement is critical—especially for floating point computations, which draw more power. If the bulk/bypass capacitors are insufficient, the power rails may drop, causing errors. Therefore sufficient capacitor backup is important.

Circuit Board Layout

This section gives recommendations to physical layouts for high speed designs.

- Place the oscillator close to the destination.
- Place the series termination close to the clock source.

For trace routing:

- Place a GND plane below the oscillator and buffer.
- Place a solid GND reference plane under the clock traces.
- Do not route the digital signals near or under the clock sources.

ESD/EOS Protection Circuits

For applications that must protect the core against fast transients (automotive or others), it is recommended that designs use three serially-connected diodes to protect the nominal core supply line. The 3-diode stack compensates for the ~–2 mV/°C temperature coefficient of each diode. Note that it is important that the selected diode have a fast turn-on time (ideally <1 ns) and low ON resistance under high-current, forward-biased operation. Additionally, a Schottky diode (Figure 24-12) should be added in parallel to handle any negative transient voltage spikes.

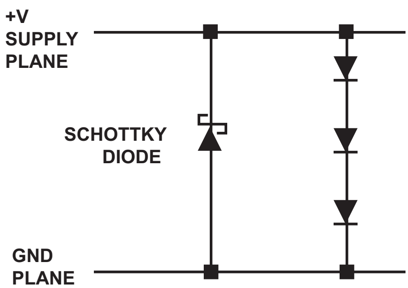

Figure 24-12. Schottky Diode

Other Recommendations and Suggestions

- Use more than one ground plane on the PCB to reduce crosstalk. Be sure to use lots of vias between the ground planes. One V_{DD} plane for each supply is sufficient. These planes should be in the center of the PCB.
- To reduce crosstalk, keep critical signals such as clocks, strobes, and bus requests on a signal layer next to a ground plane away from, or lay out these signals perpendicular to, other non-critical signals.
- If possible, position the processors on both sides of the board to reduce area and distances.
- To allow better control of impedance and delay, and to reduce crosstalk, design for lower transmission line impedances.
- Experiment with the board and isolate crosstalk and noise issues from reflection issues. This can be done by driving a signal wire from a pulse generator and studying the reflections while other components and signals are passive.

The capacitors should be placed close to the package as shown in Figure 24-13. The decoupling capacitors should be tied directly to the power and ground planes with vias that touch their solder pads. Surface-mount capacitors are recommended because of their lower series inductances (ESL) and higher series resonant frequencies.

Connect the power and ground planes to the processor's power supply pins directly with vias, do not use traces. The ground planes should not be densely perforated with vias or traces as this reduces their effectiveness. In addition, there should be several large tantalum capacitors on the board.

Designs can use either bypass placement case shown in Figure 24-13, or combinations of the two. Designs should try to minimize signal feedthroughs that perforate the ground plane.

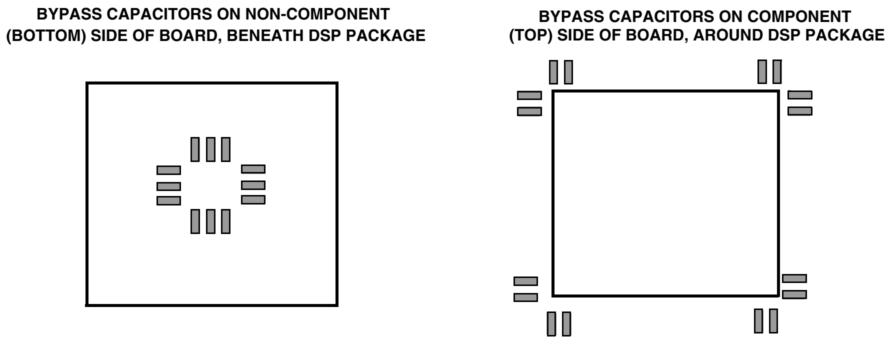

Figure 24-13. Bypass Capacitor Placement

EZ-KIT Lite Schematics

The EZ-KIT Lite® evaluation system schematics are a good starting reference. Because the EZ-KIT Lite board is for evaluation and development, extra circuitry is provided in some cases. Read the EZ-KIT Lite board schematic carefully, because sometimes a component is not populated and sometimes it has been added to make it easier to access. The design database for the SHARC processor EZ-KIT Lite boards is available online and contains all of the electronic information required for design, layout, fabrication, and assembly:

`ftp://ftp.analog.com/pub/tools/Hardware/Reference_Designs`

Oscilloscope Probes

When making high speed measurements, be sure to use a “bayonet” type or similarly short (< 0.5 inch) ground clip, attached to the tip of the oscilloscope probe. The probe should be a low capacitance active probe with 1 pF or less of loading. The use of a standard ground clip with four inches of ground lead causes ringing to be seen on the displayed trace and makes the signal appear to have excessive overshoot and undershoot.

A 1 GHz or better sampling oscilloscope is needed to see the signals accurately.

Recommended Reading

The text *High-Speed Digital Design: A Handbook of Black Magic* is recommended for further reading. This book is a technical reference that covers the problems encountered in state-of-the-art, high frequency digital circuit design. It is also an excellent source of information and practical ideas. Topics covered in the book include:

- High-Speed Properties of Logic Gates
- Measurement Techniques
- Transmission Lines
- Ground Planes and Layer Stacking
- Terminations
- Vias
- Power Systems
- Connectors
- Ribbon Cables
- Clock Distribution
- Clock Oscillators

High-Speed Digital Design: A Handbook of Black Magic, Johnson & Graham, Prentice Hall, Inc., ISBN 0-13-395724-1.

High-Speed Signal Propagation: Advanced Black Magic, Johnson & Graham, Prentice Hall, Inc., ISBN 0-13-084408-X.

System Components

This section provides some recommendations for other components to use when designing a system for your processor.

Power Management Circuits

The the ADSP-2147x and ADSP-2148x SHARC processors require a minimum of two power supplies. The ADSP-2146x processors, requires an additional supply for its DDR2 interface. The power consumption numbers are available in the respective product data sheet or EE-notes.

Refer to the following link for more information on the ADPxxxx series power supplies:

http://www.analog.com/en/power-management/switching-regulators-integrated-fet-switches/products/index.html

Supervisory Circuits

It is important that a processor (or programmable device) have a reliable active $\overline{\text{RESET}}$ that is released once the power supplies and internal clock circuits have stabilized. The $\overline{\text{RESET}}$ signal should not only offer a suitable delay, but it should also have a clean monotonic edge. Analog Devices has a range of microprocessor supervisory ICs with different features. Features include one or more of the following.

- Power-up reset
- Optional manual reset input
- Power low monitor
- Backup battery switching

The part number series for reset and supervisory circuits from Analog Devices are as follows.

- ADM69x
- ADM70x
- ADM80x
- ADM1232
- ADM181x
- ADM869x

A simple power-up reset circuit is shown in Figure 24-14 using the ADM809-RART reset generator. The ADM809 provides an active low $\overline{\text{RESET}}$ signal whenever the supply voltage is below 2.63 V. At power-up, a 240 ms active reset delay is generated to give the power supplies and oscillators time to stabilize.

Another part, the ADM706TAR, provides power on $\overline{\text{RESET}}$ and optional manual $\overline{\text{RESET}}$. It allows designers to create a more complete supervisory circuit that monitors the supply voltage. Monitoring the supply voltage allows the system to initiate an orderly shutdown in the event of power failure. The ADM706TAR also allows designers to create a watchdog timer that monitors for software failure. This part is available in an 8-lead SOIC package. Figure 24-15 shows a typical application circuit using the ADM706TAR.

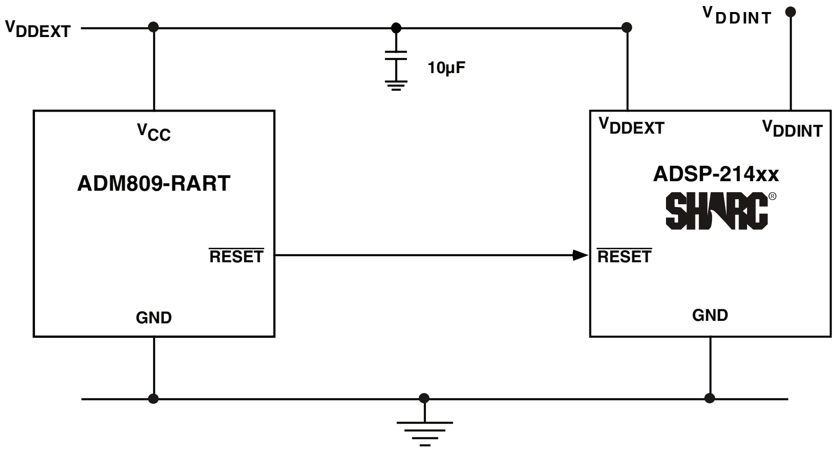

Figure 24-14. Simple Reset Generator

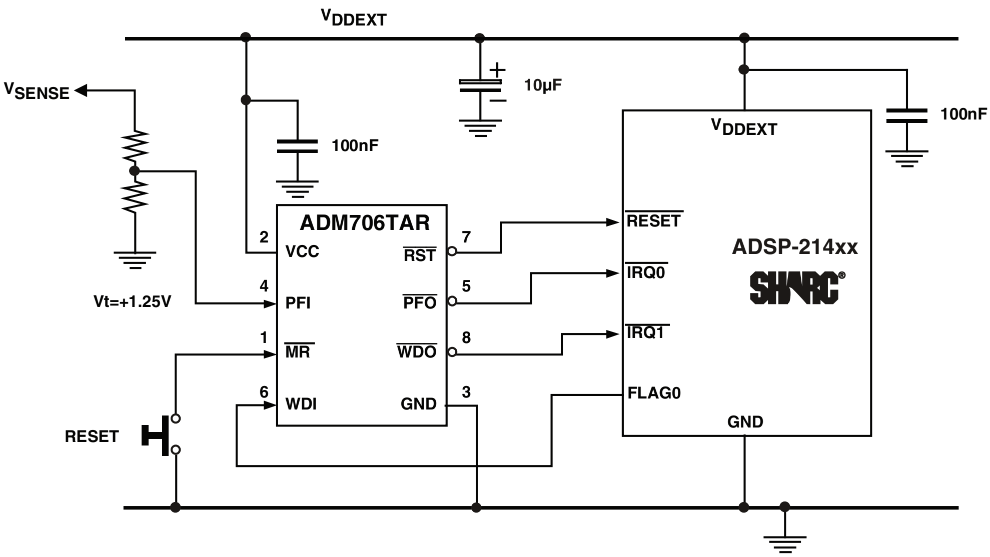

Figure 24-15. Reset Generator and Power Supply Monitor

Definition of Terms

Booting

When a processor is initially powered up, its internal SRAM and many other registers are undefined. Before actual program execution can begin, the application must be loaded from an external non-volatile source such as flash memory or a host processor. This process is known as bootstrap loading or booting and is automatically performed by the processor after power-up or after a software reset.

Boot Kernel

The boot kernel is an executable file which schedules the entire boot process. The temporary location of the kernel resides in the processor's Interrupt vector location (IVT). The IVT typically has a maximum size of 256 x 48 words. After booting, the kernel overwrites this area.

These kernel files (DXE, ASM) are supplied with the CrossCore or VisualDSP++ development tools for all boot modes. For more information on the kernels, refer to the tools documentation

Boot Master/Slave

How a processor boots is dependent on the peripheral used. See "Processor Booting" on page 24-7.

Boot Modes

The boot mode is identified by the `BOOT_CFGx` pins that are used in the boot process.

No Boot Mode

In legacy mode, the processor does not boot. Instead, it starts fetching instructions directly from external memory. The SHARC ADSP-214xx processors onwards do not support this mode.

ROM Boot Mode

For BOOT_CFGx pins = 011, the processor executes from internal ROM. Only specific versions of the processors support this mode.

A REGISTER REFERENCE

The SHARC processors have general-purpose and dedicated registers in each of their functional blocks. The register reference information for each functional block includes bit definitions, initialization values, and memory-mapped addresses (for I/O processor registers). Note that this appendix only contains information for the control and or status registers. All peripheral DMA parameter (IOP) registers (for example index, modify, count, chain pointer) are listed and described in Chapter 3, I/O Processor.

This reference does not include core related control and status registers. These registers are described in the *SHARC Processors Programming Reference*. The registers are grouped under the following headings.

When writing programs, it is often necessary to set, clear, or test bits in the processor's registers. While these bit operations can all be done by referring to the bit's location within a register or (for some operations) the register's address with a hexadecimal number, it is much easier to use symbols that correspond to the bit's or register's name. For convenience and consistency, Analog Devices supplies a header file that provides these bit and registers definitions. CrossCore Embedded Studio provides processor-specific header files in the `SHARC/include` directory. An #include file is provided with VisualDSP++ tools and can be found in the `VisualDSP/214xx/include` directory.

Overview

The I/O processor's registers are accessible as part of the processor's memory map. "Register Listing" on page B-1 lists the I/O processor's memory-mapped registers and provides a brief description of each register.

Since the I/O processor registers are memory-mapped, the processor's architecture does not allow programs to directly transfer data between these registers and other memory locations, except as part of a DMA operation. To read or write I/O processor registers, programs must use the processor core registers.

The register names for I/O processor registers are not part of the processor's assembly syntax. To ease access to these registers, programs should use the header file containing the registers' symbolic names and addresses.

Register Diagram Conventions

The register drawings in this appendix provide "at-a-glance" information about specific registers. They are designed to give experienced users basic information about a register and its bit settings. When using these registers, the following should be noted.

- In cases where there are multiple registers that have the same bits (such as serial ports), one register drawing is shown and the names and addresses of the other registers are simply listed. Also, depending on peripheral (such as ASRC), if two different ASRC ports are programmed in the same register, one peripheral is defined with a x the other with a y index.
- The bit descriptions in the figures are intentionally brief, containing only the bit mnemonic, location, and function. More detailed information can be found in the tables that follow the register drawings and in the chapters that describe the particular module.
- Shaded bits are reserved.
- The CrossCore or VisualDSP++ tools suite contains the complete listing of registers in a header file, `def214xx.h`.
- "Register Listing" on page B-1 provides a complete list of user accessible registers, their addresses, and their state at reset.

Bit Types and Settings

There are several bit types used in SHARC registers. These are described in Table A-1. In general, control register bits are read-write (RW) and status register bits are read-only (RO). In exceptional cases, bit types are shown in the "Bit" column in parenthesis where for example a RO bit is used in a control register or for read-write-one-to-clear (RW1C) bits.

Also note that the setting after reset (default setting) of most bits is 0 (cleared). In cases where this is not true, this is shown in the "Description" column in parenthesis.

Table A-1. Bit Type Definitions

Bit Type	Description	Usage
RW	Read-Write	RW bits are used primarily in control registers and DMA parameter and count registers.
RO	Read-Only	RO bits are used primarily in status registers and Shadow registers/buffers for debug aid.
ROC	Read-Only-to-Clear	ROC bits are similar to RO bits however the read changes the status bits. Can be used for example to acknowledge interrupt signals properly which clears the request logic. These bits are sticky.
WOC	Write-Only-to-Clear	WOC bits are used primarily in data FIFOs. Each write automatically updates the FIFO status.
RW1C	Read-Write-1-to-Clear	RW1C bits are sticky bits used primarily in status registers for correct interrupt acknowledge (some peripherals). These bits are sticky and their status is only cleared after a write.
RW1S	Read-Write-1-to-Set	RW1S bits need to be set for an action and hardware clears them by itself. An example is the PLL DIVEN bit or the SDRAM power-up sequence start bit.

Many registers have reserved bits. When writing to a register, programs should not change the register's reserved bits. For example:

Change bit 22 and bit 25 only:

```
ustat1 = dm(IOP_register);    /* read */
bit set ustat1 BIT22;         /* modify */
bit clr ustat1 BIT25;         /* modify*/
dm(IOP_register)=ustat1;      /* write */
```

If reading reserved bits, the read value is the last written value to these bits or the reset value of these bits.

System and Power Management Registers

The registers described in the following sections are used to control system wide operations and power management.

System Control Register (SYSCTL)

The SYSCTL register configures memory use, interrupts, and many aspects of pin multiplexing. (For more information, see "Pin Multiplexing" on page 24-28.) Bit descriptions for this register are shown in Figure A-1 and described in Table A-2.

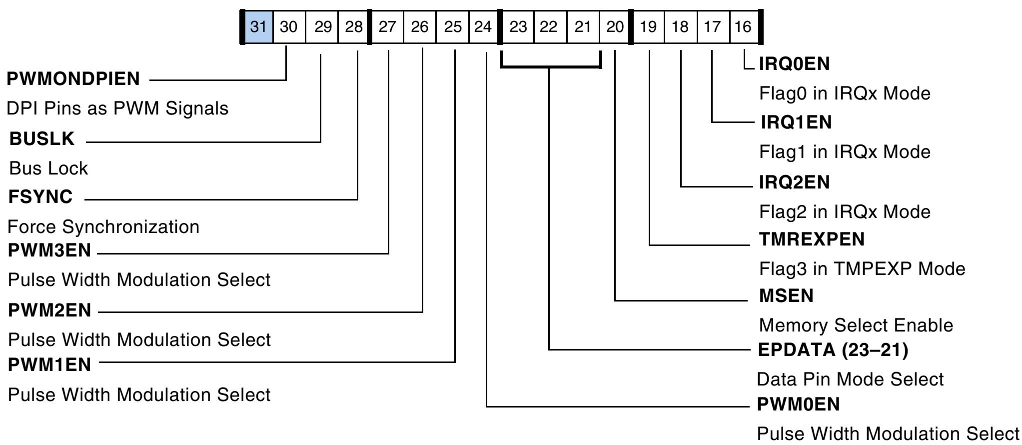

Figure A-1. SYSCTL Register

Table A-2. SYSCTL Register Bit Descriptions (RW)

Bit	Name	Description
15–0	The bits are used for controlling core function. Refer to *SHARC Processor Programming Reference*.	
16	IRQ0EN	**Flag0 Interrupt Mode.** 0 = Flag0 pin is a general-purpose I/O pin. Permits core writes. 1 = Flag0 pin is allocated to interrupt request $\overline{\text{IRQ0}}$.
17	IRQ1EN	**Flag1 Interrupt Mode.** 0 = Flag1 pin is a general-purpose I/O pin. Permits core writes. 1 = Flag1 pin is allocated to interrupt request $\overline{\text{IRQ1}}$.
18	IRQ2EN	**Flag2 Interrupt Mode.** 0 = Flag2 pin is a genera-purpose I/O pin. Permits core writes. 1 = Flag2 pin is allocated to interrupt request $\overline{\text{IRQ2}}$.
19	TMREXPEN	**Flag Timer Expired Mode.** 0 = Flag3 pin is a general-purpose I/O pin. Permits core writes. 1 = Flag3 pin output is timer expired signal (TMREXP).
20	MSEN	**Memory Select Enable.** Selects between FLGx/$\overline{\text{AMI\_MSx}}$/$\overline{\text{IRQx}}$ or TMREXP. Together with bits 19–18 generate a truth table. Detailed modes of programming for these bits are given in “Core FLAG Pins Multiplexing” on page 24-28. 0 = FLAG/$\overline{\text{IRQx}}$ pins are selected 1 = Enables FLAG2 and 3 ($\overline{\text{IRQ2}}$ and TIMEXP) as $\overline{\text{MS2}}$ and 3
23–21	EPDATA	**AMI Mode Select.** Selects between multiplexed AMI, Flags, PWM and PDAP interfaces on the AMI bus. For detailed programming modes for these bits, see “Multiplexed External Port Pins” on page 24-30.
24	PWM0EN	**Pulse Width Modulation Select.** When set (=1), enables PWM3–0. For more information, see “Pin Multiplexing” on page 24-28. Reserved for ADSP-2147x and ADSP-2148x.
25	PWM1EN	**Pulse Width Modulation Select.** When set (=1), enables PWM7–4. For more information, see “Pin Multiplexing” on page 24-28.
26	PWM2EN	**Pulse Width Modulation Select.** When set (=1), enables PWM11–8. For more information, see “Pin Multiplexing” on page 24-28.

Table A-2. SYSCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
27	PWM3EN	**Pulse Width Modulation Select.** When set (=1), enables PWM15–12. For more information, see "Pin Multiplexing" on page 24-28.
28	FSYNC	**Force Synchronization of the Shared Memory Bus** (**ADSP-2146x only**). 0 = Do not force synchronization of multiple processors arbitration in the system. 1 = Force synchronization of multiple processor arbitration in the system. Used for synchronization of DSPs in a multiple processor system, after reprogramming of PLL to another clock ratio. Clear this bit after sync achieved (allow enough time for the PLL to settle and lock to new ratio).
29	BSLK	**Bus Lock Request** (**ADSP-2146x only**). Requests bus lock where the processor maintains bus master control if set, (=1) or does not request bus lock (normal bus master control) if cleared (=0).
30	PWMOND-PIEN	**Enable PWM Signals on the DPI Pins.** Enables the PWM signals on DPI pins. When this bit is set (=1), the flags (4–15) which are routed to the DPI pins can be used as PWM signals. Applicable only for ADSP-2148x and ADSP-2147x processors.
31	Reserved	

Power Management Registers (PMCTL, PMCTL1)

The following sections describe the registers associated with the processors power management functions.

The PMCTL register, shown in Figure A-2 is a 32-bit memory-mapped register. This register contains bits to control phase lock loop (PLL) multiplier and divider (both input and output) values, PLL bypass mode, and clock control for enabling peripherals (see Table A-3 on page A-8). This register also contains status bits, which keep track of the status of the CLK_CFG pins (RO). The reset value of PMCTL is dependent on the CLK_CFG pins (bits 5–0 and 17–16).

The PMCTL1 register, shown in Figure A-3 and described in Table A-4, contains the bits for shutting down the clocks to various peripherals and selecting one of the three FIR/IIR/FFT accelerators.

Writes to this register have an effect latency of two PCLK cycles.

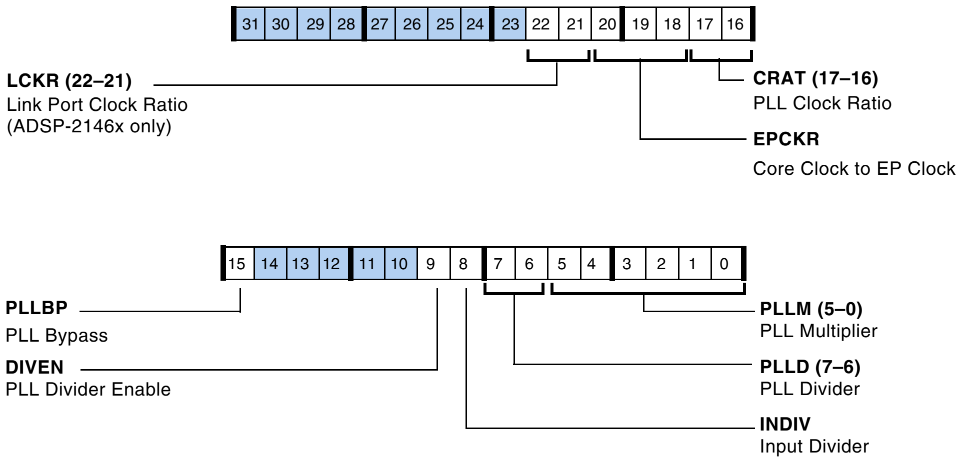

Figure A-2. PMCTL Register

Table A-3. PMCTL Register Bit Descriptions (RW)

Bit	Name	Description
5–0	PLLM	**PLL Loop Pre multiplier.** PLLM = 0 PLL multiplier = 128 0<PLLM<63 PLL multiplier = 2 × PLLM Reset value = CLK_CFG[1:0] **ADSP-2146x Settings** / **ADSP-2147x2148x Settings** 00 = 000110 = 6x / 00 = 001000 = 8x 01 = 100000 = 32x / 01 = 100000 = 32x 10 = 010000 = 16x / 10 = 010000 = 16x 11 = 000110 (Reserved) / 11 = 000110 (Reserved)

Table A-3. PMCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
7–6	PLLD	**PLL Divider (Output Post Divider).** 00 = Clock divider = 2 01 = Clock divider = 4 10 = Clock divider = 8 11 = Clock divider = 16
8	INDIV	**PLL Input Clock Pre Divider.** 0 = Divide by 1 1 = Divide by 2
9 (RW1S)	DIVEN	**Output Clock Divider Change Enable.** Enables the post divider to allow core and peripheral clock variations. 0 = No effect 1 = Register new divider (PLLD, SDCKR/DDR2CKR, LPCKR) values. When the PLL is programmed using the multipliers and the post dividers, the DIVEN and PLLBP bits should NOT be programmed in the same core clock cycle. Note that this bit is self clearing.
11–10	Reserved	
12	CLK-OUTEN	**Clockout Enable.** Mux select for CLKOUT and RESETOUT. 0 = Mux output = $\overline{\text{RESETOUT}}$ 1 = Mux output = CLKOUT The CLKOUT functionality is not characterized and only used for test purposes.
14–13	Reserved	
15	PLLBP	**PLL Bypass Mode Indication.** 0 = PLL is in normal mode 1 = Put PLL in bypass mode
17–16 (RO)	CRAT	**PLL Hardware Configuration Ratio, CLK_CFG1–0 pins.** After reset, both CLK_CFG pins define the CLKIN to core clock ratio. This ratio can be changed with the PLLM and PLLD bits. CRAT = CLK_CFG[1:0] **ADSP-2146x Settings** / **ADSP-2147x/2148x Settings** 00 = 6x / 00 = 8x 01 = 32x / 01 = 32x 10 = 16x / 10 = 16x 11 = (Reserved) / 11 = (Reserved)

Table A-3. PMCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
20–18	EPCKR	**External Port Clock Ratio.** Core clock to AMI/SDRAM or DDR2 clock. For DDR2 clock 125 MHz is min. **ADSP-2146x Settings** 000 = RATIO = 2.0 010 = RATIO = 3.0 100 = RATIO = 4.0 (AMI only) all other settings = reserved **ADSP-2147x/2148x Settings** 000 = RATIO = 2.0 001 = RATIO = 2.5 010 = RATIO = 3.0 011 = RATIO = 3.5 100 = RATIO = 4.0 all other settings = reserved
22–21	LCKR	**Link Port Clock Ratio.** Core clock to link port clock (ADSP-2146x only). 00 = RATIO = 2.0 01 = RATIO = 2.5 10 = RATIO = 3.0 (default) 11 = RATIO = 4.0
31–23	Reserved	

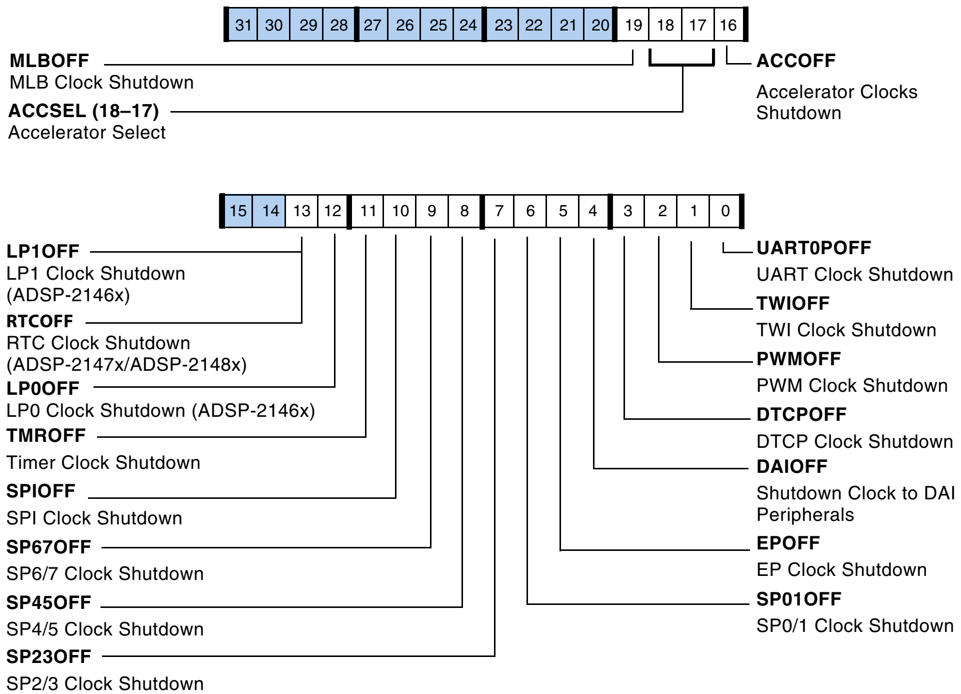

Figure A-3. PMCTL1 Register

Table A-4. PMCTL1 Register Bit Descriptions (RW)

Bit	Name	Description
0	UART0OFF	**Shutdown Clock to UART.** 0 = UART is in normal mode 1 = Shutdown clock to UART
1	TWIOFF	**Shutdown Clock to TWI.** 0 = TWI is in normal mode 1 = Shutdown clock to TWI
2	PWMOFF	**Shutdown Clock to PWM3–0.** 0 = PWM is in normal mode 1 = Shutdown clock to PWM

Table A-4. PMCTL1 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
3	DTCPOFF	**Shutdown Clock to MTM/DTCP.** 0 = MTM is in normal mode 1 = Shutdown clock to MTM
4	DAIOFF	**Shutdown Clock to DAI.** Shutdown clock to DAI related peripherals—ASRC, S/PDIF, PCGA–D, IDP, PDAP and DAI routing registers SRU. 0 = DAI is in normal mode 1 = Shutdown clock to DAI
5	EPOFF	**Shutdown Clock to External Port (AMI/SDRAM/DDR2).** 0 = External port is in normal mode 1 = Shutdown clock to external port
6	SP01OFF	**Shutdown Clock to SPORT 0, 1.** 0 = SPORTs in normal mode 1 = Shutdown clock to SPORTs
7	SP23OFF	**Shutdown Clock to SPORT 2, 3.** 0 = SPORTs in normal mode 1 = Shutdown clock to SPORTs
8	SP45OFF	**Shutdown Clock to SPORT 4, 5.** 0 = SPORTs in normal mode 1 = Shutdown clock to SPORTs
9	SP67OFF	**Shutdown Clock to SPORT 6, 7.** 0 = SPORTs in normal mode 1 = Shutdown clock to SPORTs
10	SPIOFF	**Shutdown Clock to SPI/SPIB.** 0 = SPI in normal mode 1 = Shutdown clock to SPI
11	TMROFF	**Shutdown Clock to Peripheral Timers 0/1.** 0 = Timer is in normal mode 1 = Shutdown clock to timer
12	LP0OFF	**Shutdown Clock to Link Port 0.** 0 = LP0 is in normal mode 1 = Shutdown clock to LP0 This bit is reserved for the ADSP-2147x and ADSP-2148x)

Table A-4. PMCTL1 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
13	LP1OFF/ RTCOFF	**Shutdown Clock to Link Port 1 (ADSP-2146x).** 0 = LP1 is in normal mode 1 = Shutdown clock to LP1 **Shutdown Clock to Real Time Clock (ADSP-2147x/ADSP-2148x).** 0 = RTC is in normal mode 1 = Shutdown clock to RTC
15–14	Reserved	
16	ACCOFF	**Shutdown Clock to Accelerator.** 0 = Accelerator is in normal mode 1 = Shutdown clock to accelerator
18–17	ACCSEL	**Accelerator Select.** 00 = Select FIR 01 = Select IIR 10 = Select FFT 11 = Reserved
19	MLBOFF	**Shutdown Clock to Media Local Bus.** 0 = MLB is in normal mode 1 = Shutdown clock to MLB
31–20	Reserved	

Running Reset Control Register (RUNRSTCTL)

The `RUNRSTCTL` register is used to control the running reset functionality and is described in Table A-5.

Table A-5. Running Reset Control Register Bit Descriptions (RW)

Bit	Name	Description
0	PM_RUNRST_PINEN	**Configures the RESETOUT pin for RUNRST input.** 0 = RESETOUT pin is RESETOUT 1 = RESETOUT pin is RUNRST input
1	PM_RUNRST_EN	**Enable the Running Reset Functionality.** If this bit is cleared, attempting to cause a running reset by toggling the RUNRSTIN pin has no affect. 0 = Running reset disabled 1 = Running reset enabled
31–2	Reserved	

Programmable Interrupt Priority Control Registers

The processor core supports 19 programmable prioritized interrupts. Any peripheral interrupt output may be connected to any programmable priority interrupt input. Table A-6 lists the sources.

Source Signals

Table A-6. Default Interrupt Routing

Selection Code	Source (Peripheral)	Description	Destination (Default Programmable Interrupt)
00000 (0x0)	DAIHI	DAI high priority	P0I
00001 (0x1)	SPIHI	SPI high priority	P1I
00010 (0x2)	GPTMR1I	GP Timer 0	P2I
00011 (0x3)	SP1I	SPORT1	P3I
00100 (0x4)	SP3I	SPORT3	P4I

Table A-6. Default Interrupt Routing (Cont'd)

Selection Code	Source (Peripheral)	Description	Destination (Default Programmable Interrupt)
00101 (0x5)	SP5I	SPORT5	P5I
00110 (0x6)	SP0I	SPORT0	P6I
00111 (0x7)	SP2I	SPORT2	P7I
01000 (0x8)	SP4I	SPORT4	P8I
01001 (0x9)	EPDM0I	External port DMA0	P9I
01010 (0xA)	GPTMR1I	GP Timer 1	P10I
01011 (0xB)	SP7I	SPORT7	P11I
01100 (0xC)	DAILI	DAI low priority	P12I
01101 (0xD)	EPDM1I	External Port DMA1	P13I
01110 (0xE)	DPII	DPI	P14I
01111 (0xF)	MTMI	Memory-to-Memory	P15I
10000 (0x10)	SP6I	SPORT6	P16I
10001 (0x11)	Disabled		P17I
10010 (0x12)	SPILI	SPI B low priority	P18I
10011 (0x13)	UART0RxI	UART0 receive	
10100 (0x14)	Disabled		
10101 (0x15)	UART0TxI	UART0 transmit	
10110 (0x16)	Disabled		
10111 (0x17)	TWII	Two-Wire Interface	
11000 (0x18)	PWMI	Pulse Width Modulator	
11001 (0x19)	LP0I/RTCI	Link port0/Real time clock	
11010 (0x1A)	LP1I	Link port 1	
11011 (0x1B)	ACC0I	Accelerator DMA	
11100 (0x1C)	ACC1I	Accelerator MAC	
11101 (0x1D)	MLBI	Media Local Bus	

Table A-6. Default Interrupt Routing (Cont'd)

Selection Code	Source (Peripheral)	Description	Destination (Default Programmable Interrupt)
11110 (0x1E)	Disabled		
11111 (0x1F)	Software	Select logic level high (1)	

Destination Signal Control Registers (PICRx)

This 32-bit read/write registers, shown in Figure A-4 through Figure A-7, control programmable priority interrupts and default interrupt sources. An example is shown below.

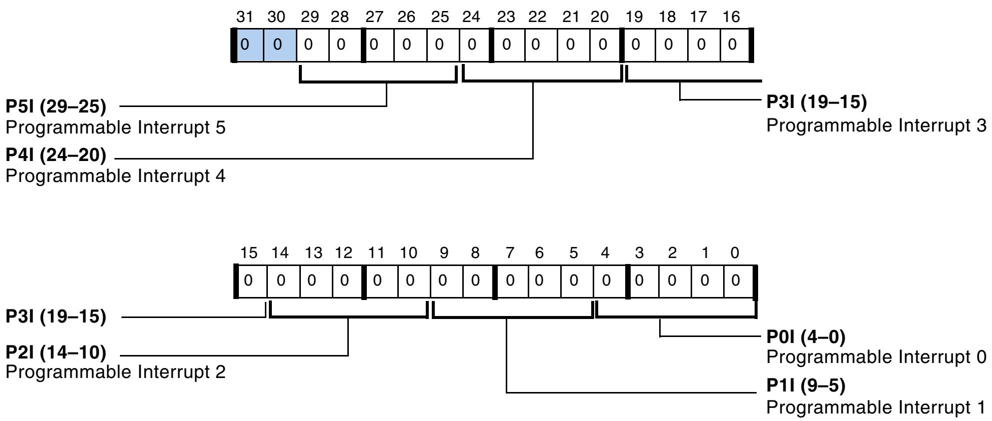

Figure A-4. PICR0 Register

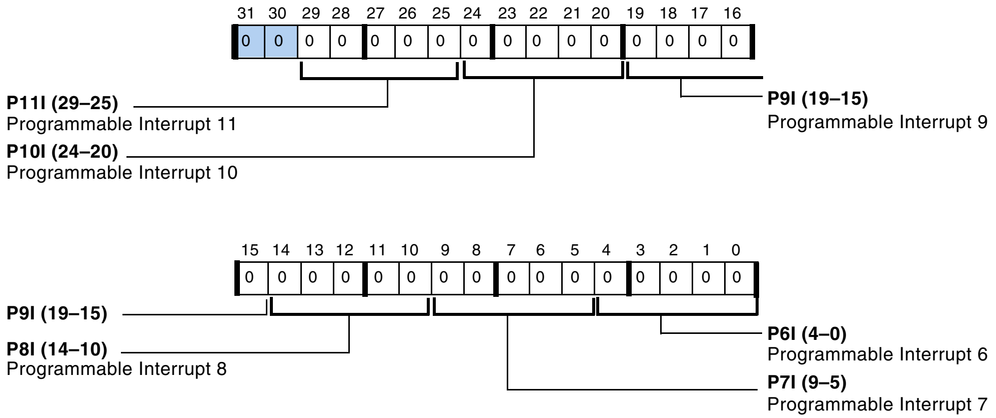

Figure A-5. PICR1 Register

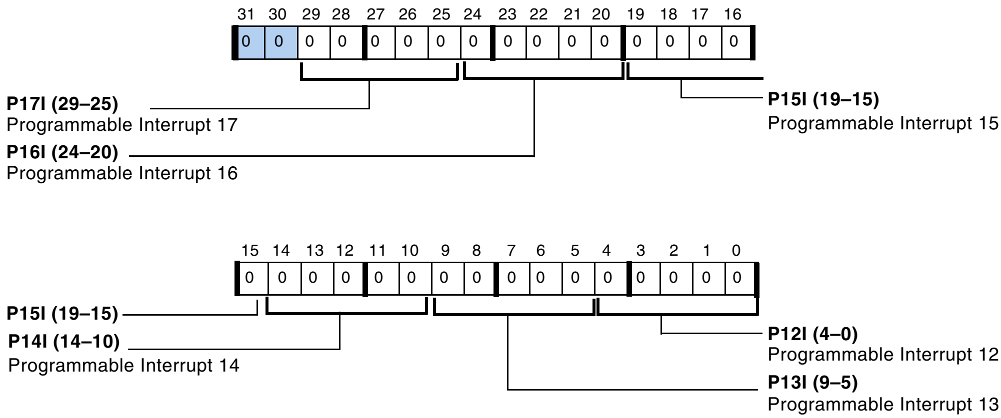

Figure A-6. PICR2 Register

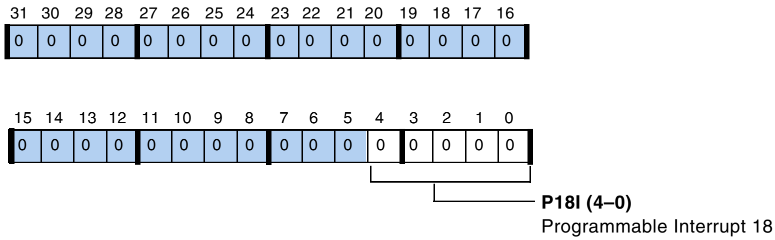

Figure A-7. PICR3 Register

DAI/DPI Interrupt Control Registers

The DAI interrupt registers are listed in Table A-7 and shown in Figure A-8. Note that for each of these registers the bit names and numbers are the same.

Table A-7. DAI Interrupt Registers

Register	Description
DAI_IRPTL_H (ROC) DAI_IRPTL_HS (RO)	High priority interrupt latch register Shadow high priority interrupt latch register
DAI_IRPTL_L (ROC) DAI_IRPTL_LS (RO)	Low priority interrupt latch register Shadow low priority interrupt latch register
DAI_IMASK_PRI (RW)	Core interrupt priority assignment register
DAI_IMASK_RE (RW)	Rising edge interrupt mask register
DAI_IMASK_FE (RW)	Falling edge interrupt mask register

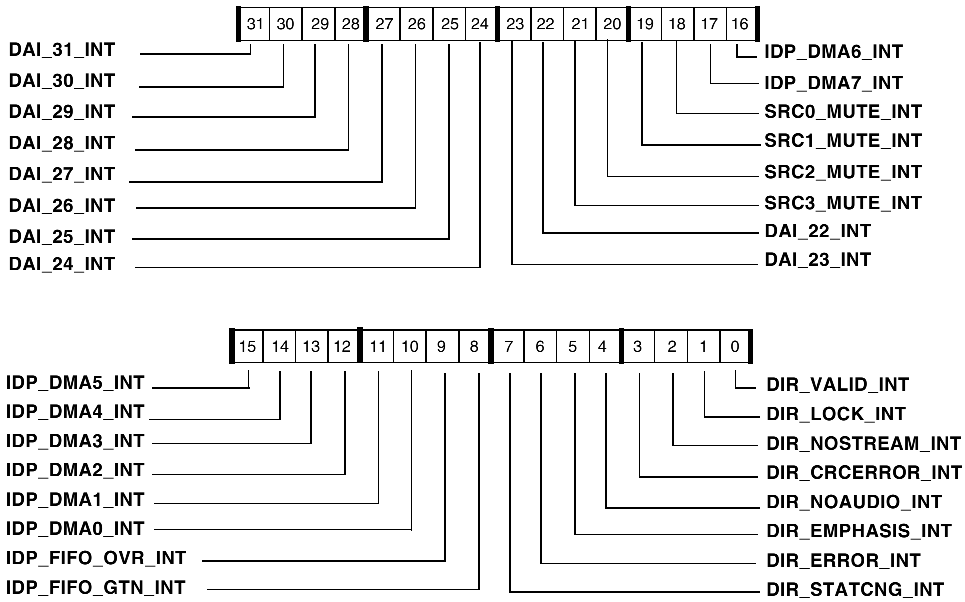

Figure A-8. DAI Interrupt Latch/Mask register

The DPI interrupt registers are shown in Figure A-9 and listed in Table A-8. Note that for each of these registers the bit names and numbers are the same.

Table A-8. DPI Interrupt Registers

Register	Description
DPI_IRPTL (ROC)	Interrupt Latch Register
DPI_IRPTL_SH (RO)	Shadow Interrupt Latch Register
DPI_IMASK_RE (RW)	Rising Edge Interrupt Mask Register
DPI_IMASK_FE (RW)	Falling Edge Interrupt Mask Register

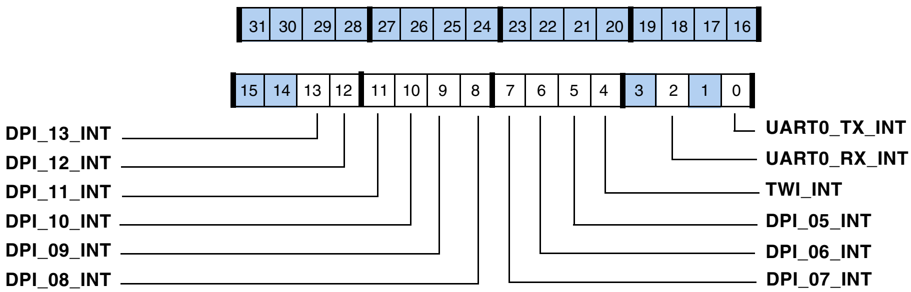

Figure A-9. DPI Interrupt Latch/Mask Register

External Port Registers

The registers in the following sections include the external port, the DDR2/SDRAM controller, and the AMI registers.

External Port Control Register (EPCTL)

The following registers are used to control the asynchronous memory interface (AMI), the DDR2 and SDRAM controller, and the shared memory interface. Bits 0–3 select a DRAM memory bank. For the ADSP-2146x processors the memory is DDR2. For the ADSP-2147x and ADSP-2148x processors the memory is SDRAM.

The external port control register can be programmed to arbitrate the accesses between the processor core and DMA, and between different DMA channels. This register is shown in Figure A-10 and described in Table A-9.

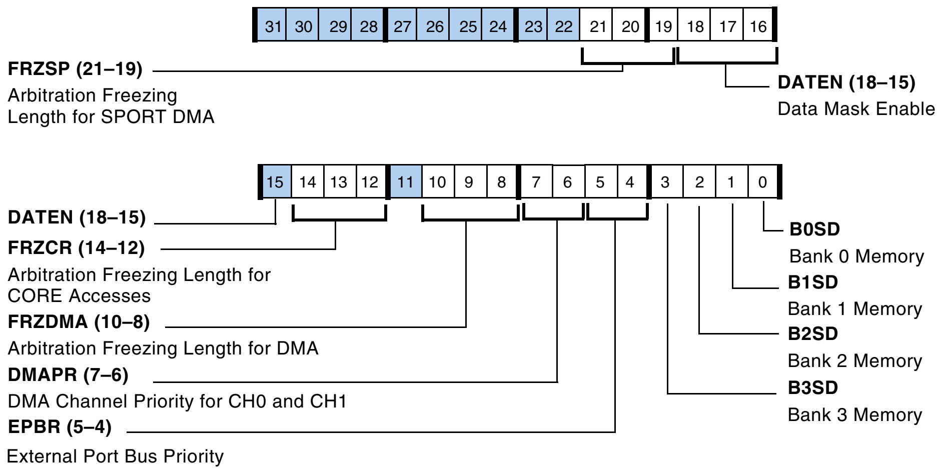

Figure A-10. EPCTL Register

Table A-9. EPCTL Register Bit Descriptions (RW)

Bit	Name	Description
0	B0SD	**Select Bank 0 Memory.** 0 = Bank 0 non-DRAM 1 = Bank 0 DDR2 for 2146x/SDRAM for 2147x/8x
1	B1SD	**Select Bank 1 Memory.** 0 = Bank 1 Non-DRAM 1 = Bank 0 DDR2 for 2146x/SDRAM for 2147x/8x
2	B2SD	**Select Bank 2 Memory.** 0 = Bank 2 Non-DRAM 1 = Bank 0 DDR2 for 2146x/SDRAM for 2147x/8x
3	B3SD	**Select Bank 3 Memory.** 0= Bank 3 Non-DRAM 1 = Bank 0 DDR2 for 2146x/SDRAM for 2147x/8x

Table A-9. EPCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
5–4	EPBR	**External Port Bus Priority.** 00 = Priority order from highest to lowest is SPORT, external port DMA, core 01 = Priority order from highest to lowest is external port DMA, SPORT, core 10 = Highest priority is core. SPORT and external port DMA are in rotating priority 11 = Rotating priority (default)
7–6	DMAPR	**External Port DMA Channel Priority.** 00 = Reserved 01 = Reserved 10 = EP DMA channel 0 high priority 11 = Rotating priority (default)
10–8	FRZDMA	**Arbitration Freezing Length for DMA.** 000 = No Freezing 001 = 4 Accesses 010 = 8 Accesses 011 = 16 Accesses 100 = 32 Accesses 101 = Page size (DDR2/SDRAM[1]) All others reserved
11	Reserved	
14–12	FRZCR	**Arbitration Freezing Length for CORE Accesses.** 000 = No Freezing 001 = 4 Accesses 010 = 8 Accesses 011 = 16 Accesses 100 = 32 Accesses 101 = Page size (DDR2/SDRAM[1]) All others reserved

Table A-9. EPCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
18–15	DATEN	**External Port Data Mask Enable (ADSP-2147x/ADSP-2148x).** In packing disable mode (PKDIS=0) the AMI memory controller masks byte wise data with zeros. The 16-bit data bus has two bytes of data. For example, if DATEN1–0 is 01, data byte 0 is masked with zeros and so on. 0000 = no effect 0001 = Data7–0 is zero 0010 = Data15–8 is zero 0011 = Data15–0 are zero All other settings reserved These bits are reserved for the ADSP-2146x models.
21–19	FRZSP	**Arbitration Freezing Length for SPORT DMA.** 000 = No Freezing 001 = 4 Accesses 010 = 8 Accesses 011 = 16 Accesses 100 = 32 Accesses 101 = Page size (DDR2/SDRAM1) All others reserved
31–22	Reserved	

1 The EPCTL register automatically reads the DRAM controller page size setting (DDR2 or SDRAM).

External Port DMA Control Registers (DMACx)

The `DMAC0-1` registers control the DMA function of their respective DMA channels as described in "Operating Modes" on page 4-128. These registers apply to all processors described in this manual and are shown in Figure A-11 and described in Table A-10.

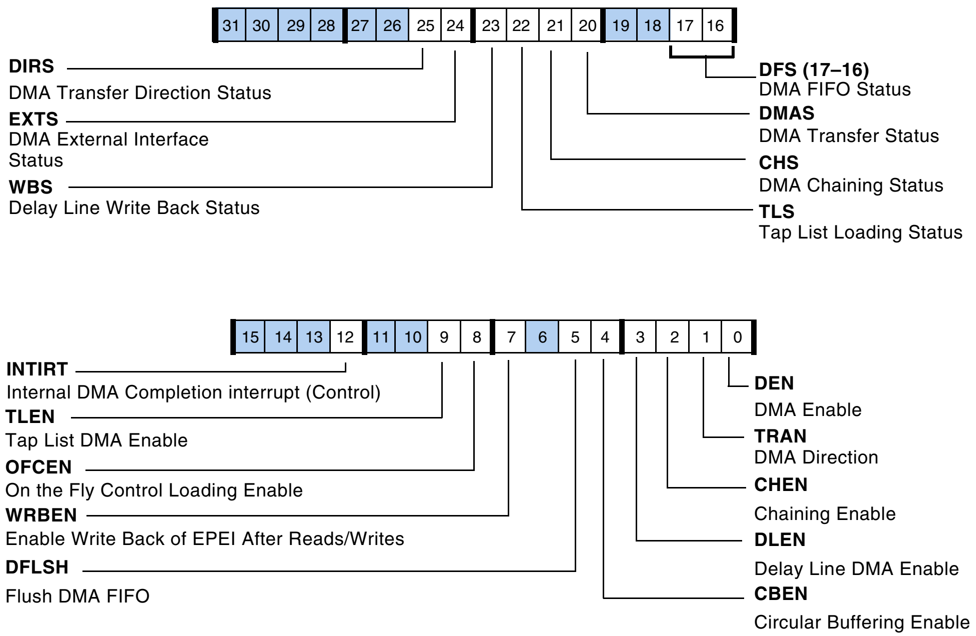

Figure A-11. DMACx Registers

Table A-10. External Port DMA Register Bit Descriptions (RW)

Bit	Name	Description
0	DEN	**DMA Enable.** 0 = External port channel x DMA is disabled 1 = Enable External port DMA for channel x
1	TRAN	**DMA Direction.** Determines the DMA data direction. For internal to internal transfers, TRAN must be set. 0 = Write to internal memory (external reads) 1 = Read from internal memory (external writes) Note: If delay line DMA is enabled then the TRAN bit doesn't have any effect. For delay line DMA, transfer direction depends on the state of delay line transfers.
2	CHEN	**Enable Chaining.** 0 = Chaining disabled 1 = Chaining enabled

Table A-10. External Port DMA Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
3	DLEN	**Enable Delay Line DMA.** DLEN is applicable only if CHEN=1. 0 = Delay-line DMA disabled 1 = Delay-line DMA enabled
4	CBEN	**Circular Buffering Enable.** 0 = Disables circular buffering with delay line DMA 1 = Enables circular buffering with delay line DMA Circular buffering can be used with normal DMA as well, if circular buffering is enabled with chaining in normal DMA then ELEP and EBEP should be part of the TCB.
5 (RW1S)	DFLSH	**Flush DMA FIFO.** The buffer is only flushed if this bit is set. It can be set with the enable bit. It takes 6 core cycles to flush the buffer. Also clears the DFS bit.
6	Reserved	
7	WRBEN	**Enable Write Back of EIEP After Reads/Writes.** Write back is automatically enabled for delay line DMA. WRBEN is applicable only if chaining is enabled (CHEN = 1)
8	OFCEN	**On the Fly Control Loading Enable.** The control bits in CPEP register are used to describe the next TCB behavior if OFCEN is set and therefore the DMA controls can be changed from TCB to TCB. 0 = Disables the control bits in CPEP register 1 = Enables the control bits in CPEP register. Note if chaining is enabled with OFCEN bit set then TRAN bit has no effect, and direction is determined by CPD bit in CPEP register.
9	TLEN	**Scatter/Gather (Tap List) DMA Enable.** 0 = Disables the tap list based scatter/gather DMA 1 = Enables the tap list based scatter/gather DMA
11–10	Reserved	
12	INTIRT	**Internal DMA Completion Interrupt (Control).** 0 = Interrupt on access completion (internal/external DMA completion depending on external read/write) 1 = Interrupt on internal DMA completion This bit is provided for backward compatibility with older SHARC processors.
15–13	Reserved	

Table A-10. External Port DMA Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
17–16 (RO)	DFS	**DMA FIFO Status.** 00 = FIFO empty 01 = FIFO partially full 11 = FIFO full 10 = Reserved
19–18	Reserved	
20 (RO)	DMAS	**DMA Transfer Status.** 0 = DMA idle 1 = DMA in progress
21 (RO)	CHS	**DMA Chaining Status.** 0 = DMA chain loading is not active 1 = DMA chain loading is active
22 (RO)	TLS	**TAP List Loading Status.** 1 = TAP list loading is active 0 = TAP list loading is not active
23 (RO)	WBS	**Delay Line Write Pointer Write Back Status.** 0 = Write pointer write back is not active 1 = Write pointer write back is active
24 (RO)	EXTS	**DMA External Interface Status.** 0 = DMA external interface does not have any access pending 1 = DMA external interface has access pending
25 (RO)	DIRS	**DMA Transfer Direction Status.** 0 = DMA direction is external reads 1 = DMA direction is external writes This is useful for delay line DMA where the transfer direction changes with the state of the DMA state machine. For standard DMA, DIRS reflects the state of the TRAN bit.
31–26	Reserved	

Asynchronous Memory Interface Registers (AMI)

The next two sections describe the control and status registers for the AMI.

AMI Control Registers (AMICTLx)

The `AMICTL0-3` registers control the mode of operations for the four banks of external memory. This register is shown in Figure A-12 and described in Table A-11. Note for all AMI timing bit settings, all defined cycles are derived from the SDRAM clock.

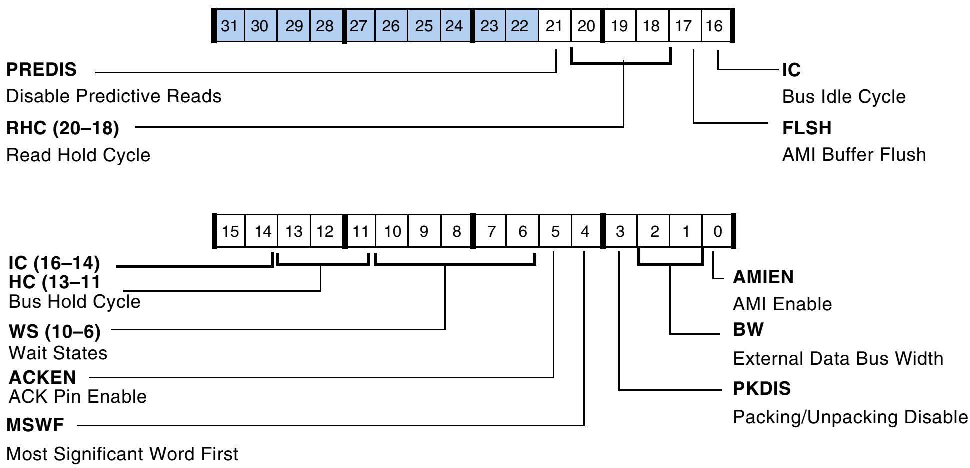

Figure A-12. AMICTLx Registers

Table A-11. AMICTLx Register Bit Descriptions (RW)

Bit	Name	Description
0	AMIEN	**AMI Enable.** Enables the AMI controller for the dedicated external bank. 0 = AMI is disabled 1 = AMI is enabled To access an external memory bank, the AMIEN bit in the corresponding AMICTLx register has to be set. If any of the AMIEN bits are set, then the AMI is enabled and can access memory. However, bank selects cannot be driven for that bank whose AMIEN is not set (but read/write strobes can occur). Any access made to a bank whose AMIEN bit is not set occurs at WS = 2 and 8-bit mode without any hold/idle cycles. In any case this access occurs without the bank select and is a void access. Moreover, the AMIEN bit should not be cleared when an access is on-going (when the AMIS bit in the AMISTAT register is set).
2–1	BW	**External Data Bus Width (ADSP-2147x/ADSP-2148x).** 00 = 8-bit 01 = 16-bit 10, 11 = Reserved These bits are reserved for the ADSP-2146x models.
3	PKDIS	**Packing Disable.** 8/16-bit data received packed to 32-bit data. Similarly, 32-bit data to be transmitted is unpacked to two 16-bit data or four 8-bit data. For disable packing 8/16-bit data received zero-filled, for transmitted data only 16-bit or the 8-bit LSB part of the 32-bit data is written to external memory. **ADSP-2146x Settings** **ADSP-2147x/2148x Settings** 0 = 8 to 32 packing 0 = 8/16 to 32 packing 1 = no packing 1 = no packing
4	MSWF	**Most Significant Word First.** Applicable only with packing disabled (PKDIS=0). 1st 8/16-bit word read/write occupies the least significant position in the 32-bit packed word or 1st 8/16-bit word read/write occupies the most significant position in the 32-bit packed word. **ADSP-2146x Settings** **ADSP-2147x/2148x Settings** 0 = 1st 8 bit is LSW 0 = 1st 8/16 bit is LSW 1 = 1st 8 bit is MSW 1 = 1st 8/16 bit is MSW

Table A-11. AMICTLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
5	ACKEN	**Enable the ACK pin.** If enabled, reads/writes to devices must be extended by the corresponding devices by pulling ACK low. When ACKEN is set, then the ACK pin is sampled after the wait state value is programmed.
10–6	WS	**Wait States.** 00000 = Reserved (wait state value of 32 if used) 00001 = wait state = 1 (min if ACK input used) 00010 = wait state = 2 ... 11111 = Wait state = 31
13–11	HC	**Bus Hold Cycle at the End of Write Access.** 000 = Disable bus hold cycle 001 = Hold address for one external port clock cycle 010 = Hold address for two external port clock cycles ... 111 = Hold address for seven cycles
16–14	IC	**Bus Idle Cycle.** Default Idle cycles are inserted whenever read to write in a bank or read to read between two external banks or a read to the SDC occurred. A bus idle cycle is an inactive bus cycle that the processor automatically generates to avoid data bus driver conflicts. Such a conflict can occur when a device with a long output disable time continues to drive after RD is deasserted, while another device begins driving on the following cycle. Idle cycles are also required to provide time for a slave in one bank to three-state its ACK driver, before the slave in the next bank enables its ACK driver. 000 = 0 idle cycles 001 = 1 idle cycle … 111 = 7 idle cycles
17 (RW1S)	FLSH	**AMI Packing Buffer Flush.** 0 = Buffer holds the data 1 = The buffer is only flushed if this bit is set. It can be set with the AMIEN bit. It takes four core cycles to flush the buffer.

Table A-11. AMICTLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
20–18	RHC	**Read Hold Cycle.** Controls the delay between two reads. 000 = Disable read hold cycle 001 = Hold address for one cycle ... 111 = Hold address for seven cycles
21	PREDIS	**Disable Predictive Reads.** 0 = Predictive reads enabled 1 = Predictive reads disabled For more information, see "Predictive Reads" on page 4-27.
31–22	Reserved	

AMI Status Register (AMISTAT)

This 32-bit, read-only register provides status information for the AMI interface and can be read at any time. This register is shown in Figure A-13 and described in Table A-12.

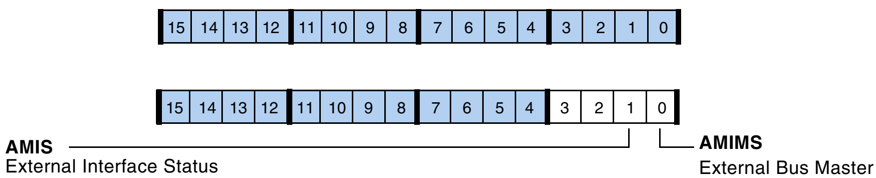

Figure A-13. AMISTAT Register

Table A-12. AMISTAT Register Bit Descriptions (RO)

Bit	Name	Description
0	AMIMS	**AMI External Bus Master.** **ADSP-2146x Settings** / **ADSP-2147x/2148x Settings** 0 = Reserved / 0 = SDRAM Controller 1 = AMI (always master*) / 1 = AMI controller (default) *Since the AMI and DDR2 pins are independent on the ADSP-2146x products, AMIMS always reads 1.
2–1	AMIS	**External Interface Status.** 0 = AMI interface idle 1 = AMI access pending
15–4	Reserved	

SDRAM Registers

This section provides complete descriptions of the SDRAM controller's memory-mapped registers for SDRAM programming. Programs may write to the SDRAM control registers as long as the controller is not accessing memory devices. Otherwise, the controller responds to any writes to its registers after it finishes any ongoing memory accesses.

Control Register (SDCTL)

The SDRAM memory control register includes all programmable parameters associated with the SDRAM access timing and configuration. This register is shown in Figure A-14 and described in Table A-13.

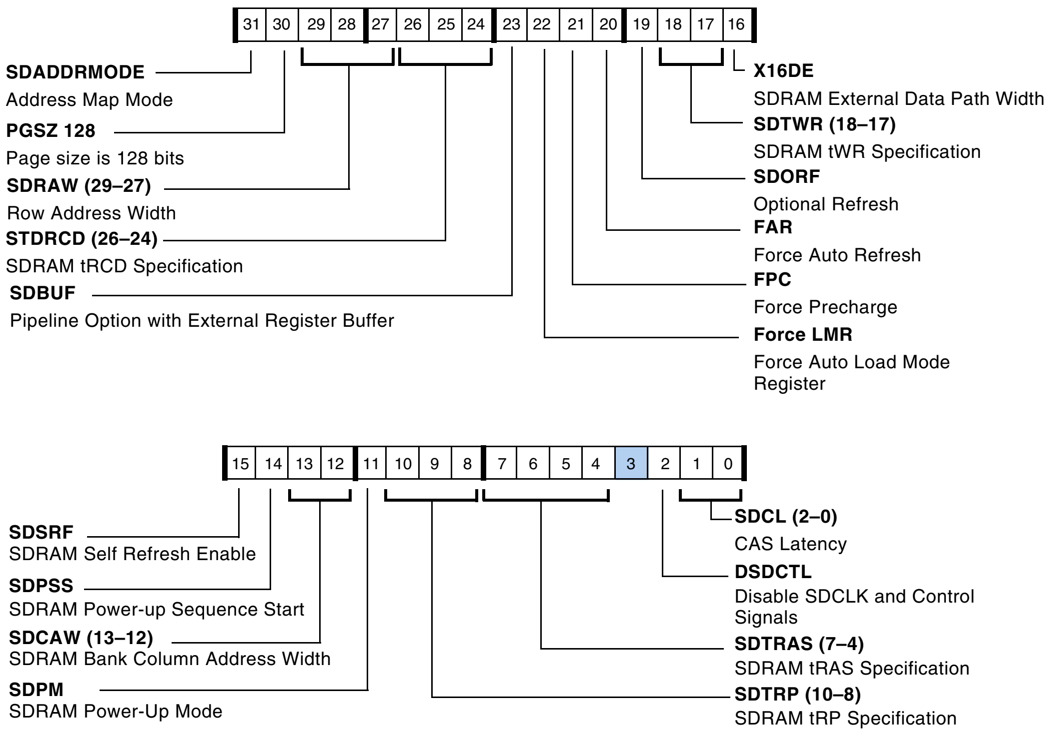

Figure A-14. SDCTL Register

Table A-13. SDCTL Register Bit Descriptions (RW)

Bit	Name	Description
1–0	SDCL	**CAS Latency.** 2–3 SDCLK cycles. The delay in clock cycles between when the SDRAM detects the read command and when it provides the data at its output pins. 00, 01 = Reserved 10 = 2 cycles (default) 11 = 3 cycles A CAS latency of 2 is supported only up to 133 MHz SDCLK.
2	DSDCTL	**Disable Controller and Clocks.** Used to enable or disable the SDC and its pins. If DSDCTL is set, any access to SDRAM address space does not occur externally and all SDC control pins are in their inactive states and the SDRAM clock is not running. 0 = Active 1 = Disabled When not using SDRAM or when using parts without an external port, systems should set this bit as early as possible after booting to reduce power consumption. If the SDSRA bit is set (self-refresh), setting the DSDCTL bit freezes the SDCLK to reduce power.
3	Reserved	
7–4	SDTRAS	**tRAS Specification.** Row Active Open Delay is 1–15 SDCLK cycles. Based on the system clock frequency and the timing specifications of the SDRAM used. Programmed parameters apply to all four banks in the external memory. Refer to the SDRAM data sheet.
10–8	SDTRP	**tRP Specification.** Row Precharge Delay is 1–8 SDCLK cycles. Based on the system clock frequency and the timing specifications of the SDRAM used. Programmed parameters apply to all four banks in the external memory. Refer to the SDRAM data sheet.
11	SDPM	**Power-Up Mode.** The SDPM and SDPSS bits work together to specify and trigger an SDRAM power-up (initialization) sequence. If the SDPM bit is set (=1), the SDC performs a precharge all command, followed by a load mode register command, followed by eight auto-refresh cycles. If the SDPM bit is cleared (=0), the SDC performs a precharge all command, followed by eight auto-refresh cycles, followed by a load mode register command. Refer to the SDRAM data sheet.

Table A-13. SDCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
13–12	SDCAW	**Bank Column Address Width.** The number of columns in an internal bank. Also referred to as page size. 00 = 8 bits 01 = 9 bits 10 = 10 bits 11 = 11 bits
14 (RW1S)	SDPSS	**Power-Up Sequence Start.** The power-up sequence is triggered by setting this bit. Note that there is a latency for this first access to SDRAM because the SDRAM power-up sequence takes many cycles to complete. 0 = No effect 1 = Enable power-up on next SDRAM access
15 (RW1S)	SDSRF	**Self-Refresh Enable.** When the SDSRF bit is set to 1, self-refresh entry command is triggered. Once the SDC completes any active transfers, the SDC executes the sequence of commands to put the SDRAM into self-refresh mode. Any access to the enabled SDRAM bank causes the SDC to trigger a self-refresh exit command.
16	X16DE	**External 16-bit Data Path Width.** Programs should always set (=1) this bit. 0 = Reserved 1 = 16-bit
18–17	SDTWR	**tWR Specification.** Write To Precharge Delay) is 1–3 SDCLK cycles. Based on the system clock frequency and the timing specifications of the SDRAM used. Programmed parameters apply to all four banks in the external memory. Refer to the SDRAM data sheet.
19	SDORF	**Optional Auto-Refresh Command.** 0 = Auto-refresh occurs when refresh counter expires 1 = Auto-refresh not performed
20 (RW1S)	FAR	**Force Auto-Refresh Command.** Performs an auto-refresh immediately. 0 = No effect 1 = Force auto-refresh

Table A-13. SDCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
21 (RW1S)	FPC	**Force Precharge.** Performs a precharge all immediately. 0 = No effect 1 = Force precharge
22 (RW1S)	FMR	**Force Load Mode Register Command.** This command performs a load mode register command immediately. 0 = No effect 1 = Force MR
23	SDBUF	**Pipeline Option with External Register Buffer.** **0 = No buffer option** 1 = External SDRAM CTL/ADDR control buffer enable
26–24	SDTRCD	**tRCD Specification.** RAS to CAS Delay is = 1–7 SDCLK cycles. Based on the system clock frequency and the timing specifications of the SDRAM used. Programmed parameters apply to all four banks in the external memory (0x1 default). See the SDRAM data sheet.
29–27	SDRAW	**Row Address Width.** 000=8, 001=9 010=10, 011=11 100=12, 101=13 110=14, 111=15
30	PGSZ 128	**Page Size of 128 Words.** This bit allows programs to configure the SDC for a page size of 128 words (7 bits) which supports most available 32 Mb SDRAMs. 0 = No effect, page size decided by SDCAW bits. 1 = Page size 128 words. Column width = 7 bits, override CAW settings.
31	SDADDRMODE	**Select Address Mapping.** This bit selects how data is stored in memory. 0 = Bank interleaving 1 = Page interleaving

Refresh Rate Control Register (SDRRC)

The SDRAM refresh rate control register provides a flexible mechanism for specifying the auto-refresh timing. This register is shown in

Figure A-15. For information on using the SMODIFY bit see “SDRAM Read Optimization” on page 4-59.

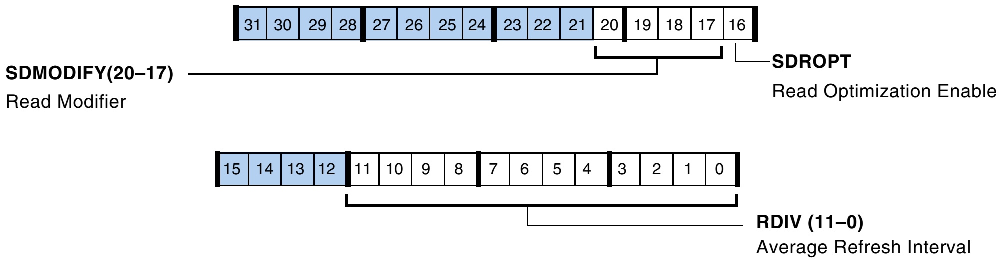

Figure A-15. SDRRC Register

Table A-14. SDRRC Register Bit Descriptions (RW)

Bit	Name	Description
11–0	RDIV	**Refresh Interval.** RDIV setting defines the average refresh interval between two subsequent refresh commands. The formula is shown in “Refresh Rate Control” on page 4-36. Note that the SDRAM manufacturer data sheets distinguish between commercial, industrial and automotive grades.
15–12	Reserved	
16	SDROPT	**Read Optimization.** If set (=1) enables read optimization to improve read throughput for core or external port DMA access. 0 = Disabled 1 = Enabled (default)
20–17	SDMODIFY	**SDRAM Read Modifier.** According to SDROPT bit this bit should be set to match the DAG or DMA modifier. 0000 = 0 0001 = 1 (default) 1111 = 15
31–21	Reserved	

Control Status Register 0 (SDSTAT0)

The SDRAM control status register provides information on the state of the SDC. This information can be used to determine when it is safe to alter SDC control parameters or as a debug aid. This register is shown in Figure A-16 and described in Table A-15.

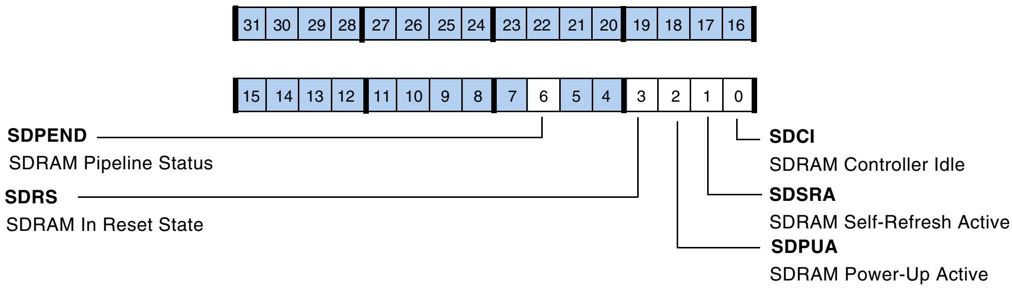

Figure A-16. SDSTAT0 Register

Table A-15. SDSTAT0 Register Bit Descriptions (RO)

Bit	Name	Description
0	SDCI	**SDC Idle.** This bit is set if the SDC is performing a command or auto-refresh. If no access, this bit is cleared. 0 = SDC idle 1 = SDC access
1	SDSRA	**SDC Self-Refresh Mode.** If set, controller is in self-refresh mode. 0 = Non self-refresh mode (SDCKE pin high) 1 = Self-refresh mode (SDCKE pin low)
2	SDPUA	**SDC Power-Up Active.** If set, controller is in power-up mode. 0 = Non power-up mode (SDPSS bit cleared in SDCTL) 1 = Power-up mode (SDPSS-bit set in SDCTL)
3	SDRS	**SDC Reset State.** If set, power-up sequence occurred. 0 = No power-up sequence 1 = Power-up sequence occurred
5–4	Reserved	

Table A-15. SDSTAT0 Register Bit Descriptions (RO) (Cont'd)

Bit	Name	Description
6	SDPEND	**SDC Controller Pipeline Status.** 0 = No access pending in controller pipeline 1 = Read/Write access pending in controller pipeline.
31–7	Reserved	

Controller Status Register 1 (SDSTAT1)

This register reports the SDRAM bank active/idle status. This register is shown in Figure A-17 and described in Table A-16.

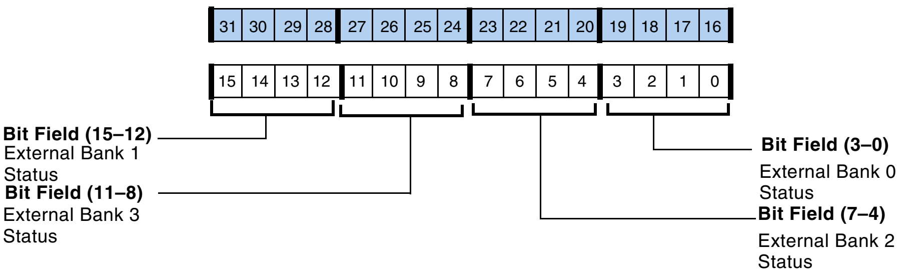

Figure A-17. SDSTAT1 Register

Table A-16. SDSTAT1 Register Bit Descriptions (RO)

Bit Field	Field Name	Description
3–0	External Bank 0 Status	**External Bank 0 Active/Precharge State.** xxx1 = Internal bank 0 in open state xxx0 = Internal bank 0 in precharge state xx1x = Internal bank 1 in open state xx0x = Internal bank 1 in precharge state … 1xxx = Internal bank 3 in open state 0xxx = Internal bank 3 in precharge state

Table A-16. SDSTAT1 Register Bit Descriptions (RO) (Cont'd)

Bit Field	Field Name	Description
7–4	External Bank 1 Status	**External Bank 0 Active/Precharge State.** xxx1 = Internal bank 0 in open state xxx0 = Internal bank 0 in precharge state xx1x = Internal bank 1 in open state xx0x = Internal bank 1 in precharge state … 1xxx = Internal bank 3 in open state 0xxx = Internal bank 3 in precharge state
11–8	External Bank 2 Status	**External Bank 0 Active/Precharge State.** xxx1 = Internal bank 0 in open state xxx0 = Internal bank 0 in precharge state xx1x = Internal bank 1 in open state xx0x = Internal bank 1 in precharge state … 1xxx = Internal bank 3 in open state 0xxx = Internal bank 3 in precharge state
15–12	External Bank 3 Status	**External Bank 0 Active/Precharge State.** xxx1 = Internal bank 0 in open state xxx0 = Internal bank 0 in precharge state xx1x = Internal bank 1 in open state xx0x = Internal bank 1 in precharge state … 1xxx = Internal bank 3 in open state 0xxx = Internal bank 3 in precharge state

DDR2 Registers

This section provides complete descriptions of the DDR2 controller's memory-mapped registers for DDR2 programming

Programs may write to the DDR2 control registers as long as the controller is not accessing memory devices. Otherwise, the controller responds to any writes to its registers after it finishes any ongoing memory accesses.

DDR2 Control Register 0 (DDR2CTL0)

The DDR DDR2CTL0 register includes the programmable parameters associated with the DDR configuration. Figure A-18 and Table A-17 show the corresponding control bit definitions.

The FEMRx, FLMR, FDLLCAL, FAR, FPC, SREF_EXIT, DDR2SRF, and DDR2PSS bits are automatically cleared on the next clock edge cycle after they are set.

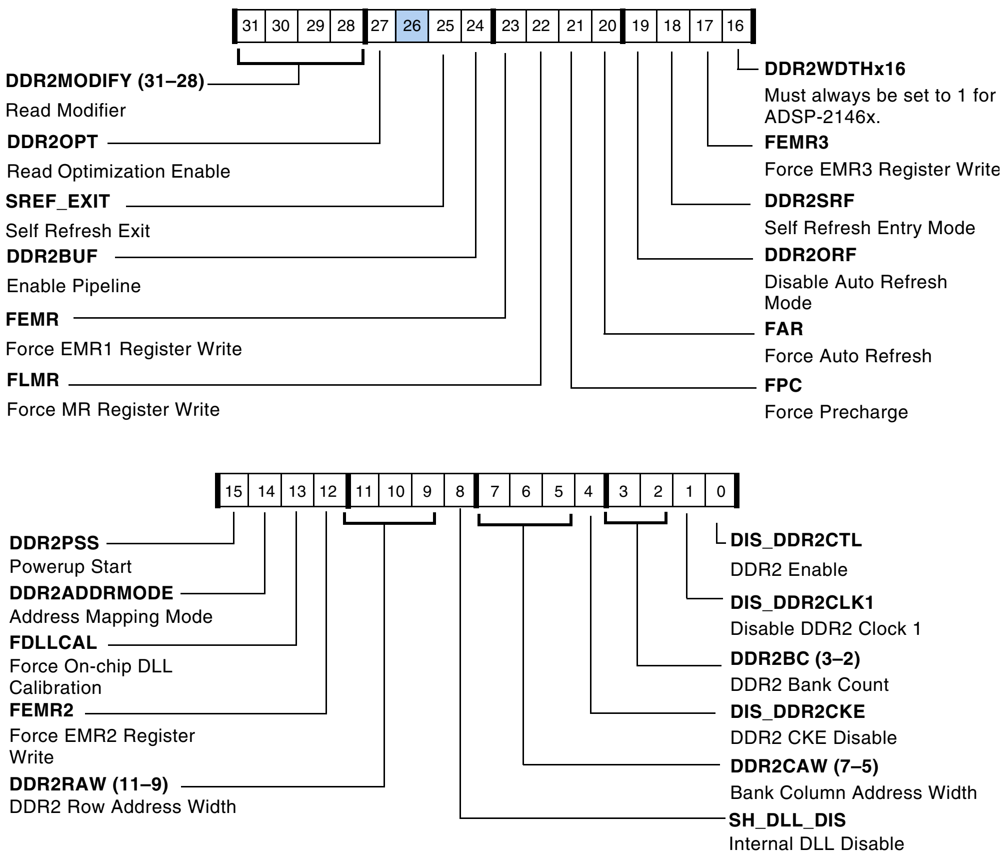

Figure A-18. DDR2CTL0 Register

Table A-17. DDR2CTL0 Register Bit Descriptions (RW)

Bit	Name	Description
0	DIS_DDR2CTL	**Disable DDR2 Control Pins.** If set, no accesses to external DDR2 DRAM address spaces occur. All associated control pins (DDR2 CLK, DDR2_RAS, DDR2_CAS, DDR2_WE, DDR2_CS, DDR2_ODT except DDR2_CKE) are in their inactive states 0 = Enable Control Pins 1 = Disable Control Pins This bit should not be set when DDR2 interface is active. It can be set in self-refresh mode to reduce pin power consumption.
1	DIS_DDR2CLK1	**Disable DDR2 Clock 1.** Used to disable the 2nd output clock of the controller. By default, both output clocks are driven. 0 = Activate 1 = Disable (default)
3–2	DDR2BC	**Bank Count—4 or 8 Bank Device.** 00 = Reserved 01 = 4 Bank device 10 = 8 Bank device (default) 11 = Reserved
4	DIS_DDR2CKE	**Precharge Power-Down Mode.** If set, the DDR2CKE signal is deasserted to bring the DDR2 into precharge power-down mode. Note that memory banks are not refreshed in this mode. 1 = Enter Precharge Power-down Mode 0 = Exit Precharge Power-down Mode
7–5	DDR2CAW	**Bank Column Address Width.** Number of columns in an internal bank. Also referred to as page size. 000 = Page width 256 001 = Page width 512 010 = Page width 1024 011 = Page width 2048 100 = Page width 4096 Other values are reserved
8	SH_DLL_DIS	**SHARC DDR2 Controller DLL Disable.** Bypass on-chip DLL for debug mode only. 0 = Enable SHARC low/high byte DLL 1 = Disable SHARC low/high byte DLL

Table A-17. DDR2CTL0 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
11–9	DDR2RAW	**Row Address Width.** 000 = 8 bits 001 = 9 bits ... 111 = 15 bits
12 (RW1S)	FEMR2	**Force EMR2 Register Write.** Forces EMR2 only if the banks are all precharged. 0 = No effect 1 = Force EMR2 register write to DDR2
13 (RW1S)	FDLLCAL	**Force DLL External Bank Calibration.** Triggers a calibration by multiple read commands for sensing the phase delay between internal DDR2 clocks and the received DQS signals. Only assigned external banks (EPCTL register) allow calibration. 0 = No effect 1 = Trigger DLL for external bank calibration
14	DDR2ADDRMODE	**Select the Address Mapping.** This bit selects how the data are stored in the DDR2 memory. 0 = Page interleaving map (consecutive pages/different banks) 1 = Bank interleaving map (consecutive banks)
15 (RW1S)	DDR2PSS	**Power-Up Sequence Start.** The power-up sequence is started by setting this bit. Note that the entire power-up sequence takes many cycles to complete. The more external banks assigned, the longer the power-up time. 0 = No effect 1 = Trigger power-up sequence Note that the power-up sequence does NOT require a memory access to be executed. If using forced commands, this bit should be cleared.
16	DDR2WDTHx16	**External 16-bit Data Path Width.** Programs should always set (=1) this bit. 0 = Reserved 1 = 16-bit
17 (RW1S)	FEMR3	**Force EMR3 Register Write.** Forces EMR3 only if the banks are all precharged. 0 = No effect 1 = Force EMR3 register write to DDR2

Table A-17. DDR2CTL0 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
18 (RW1S)	DDR2SRF	**Self-Refresh Mode.** 0 = No effect 1 = Enters self-refresh mode
19	DDR2ORF	**Auto-Refresh Command.** If this bit is set, the auto-refresh command is not issue to the DDR2 memory. This mode allows data streaming connection to FPGA were the refresh is not required. 0 = Auto-refresh command occurs when refresh counter expires. 1 = Auto-refresh not performed
20 (RW1S)	FARF	**Force Auto-Refresh.** This bit allows programs to explicitly trigger an auto-refresh command. To use this bit requires that bit 21 is also set, otherwise the DDR2 may crash. 0 = No effect 1 = Force auto-refresh
21 (RW1S)	FPC	**Force Precharge All.** This bit allows programs to explicitly trigger a PREA command. 0 = No effect 1 = Force precharge
22 (RW1S)	FLMR	**Force Load Mode Register.** Forces MR only if the banks are all precharged. 0 = No effect 1 = Force MR register write to DDR2
23 (RW1S)	FEMR	**Force EMR1 Register Write.** Forces EMR1 only if the banks are all precharged. 0 = No effect 1 = Force EMR1 register write to DDR2
24	DDR2BUF	**Enable Pipeline.** Enabled this bit if the nominal capacitive load is exceeded by connecting DDR2 chips in parallel (4 x IO4). 0 = Disable 1 = External DDR2 control/address buffer enable
25 (RW1S)	SREF_EXIT	**Self-Refresh Exit.**

Table A-17. DDR2CTL0 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
27	DDR2OPT	**Read Optimization Enable.** 0 = Disable read optimization 1 = Enable read optimization (default)
31–28	DDR2MODIFY	**Read Modifier (In Optimization Mode).** 0000 = Modifier 0 0001 = Modifier 1 (default) ... 1111 = Modifier 15 Note that these bits only are only effective in SISD mode.

DDR2 Timing Control Register 1 (DDR2CTL1)

The DDR2CTL1 register includes the programmable parameters associated with the DDR access timing. Figure A-19 and Table A-18 show the DDR timing control bit definitions. All the values are defined in terms of number of DDR2 clock cycles.

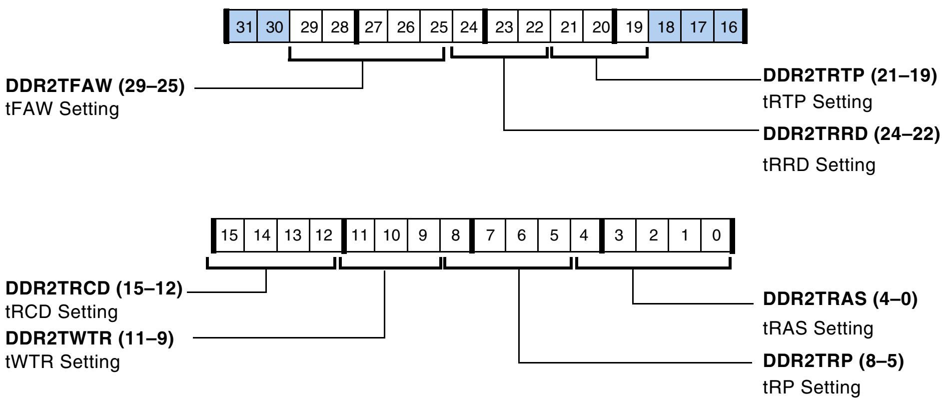

Figure A-19. DDR2CTL1 Register

Table A-18. DDR2CTL1 Register Bit Descriptions (RW)

Bit	Name	Description
4–0	DDR2TRAS	**Row Active Time.** 00000 = Reserved 00001 = 1 clock cycle 00010 = 2 clock cycles … 11111 = 31 clock cycles (0x6 default)
8–5	DDR2TRP	**Row Precharge Time.** Note that for 8 banked devices the timing spec becomes t_{RP} + $1t_{CK}$. 0000 = Reserved 0001 = 1 clock cycle 0010 = 2 clock cycles … 1111 = 15 clock cycles (0x3 default)
11–9	DDR2TWTR	**Write to Read Delay.** 000 = Reserved 001 = 1 clock cycle 010 = 2 clock cycles (default) … 111 = 7 clock cycles
15–12	DDR2TRCD	**RAS to CAS Delay.** 000 = Reserved 001 = 1 clock cycle 010 = 2 clock cycles … 111 = 7 clock cycles (0x3 default)
18–16	Reserved	
21–19	DDR2TRTP	**Read to Precharge Delay.** 000 = Reserved 001 = 1 clock cycle 010 = 2 clock cycles (default) … 111 = 7 clock cycles

Table A-18. DDR2CTL1 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
24–22	DDR2TRRD	**Row to Row Activation Delay.** 000 = Reserved. 001 = 1 clock cycle (default) 010 = 2 clock cycles … 111 = 7 clock cycles
29–25	DDR2TFAW	Force Activation Window. For 8 banked devices up to 4 banks open in activation window. For 4 banked devices the settings are ignored. 00000 = Reserved 00001 = 1 clock cycle 00010 = 2 clock cycles … 11111 = 31 clock cycles (0xA default)
31–30	Reserved	

DDR2 Control Register 2 (DDR2CTL2)

Figure A-20 and Table A-19 show the DDR2 control register 2 bit definitions. Values written into this register are loaded into the DDR2 mode register during power up (or when `Force LMR` bit in the `DDR2CTL0` register is set). This register should be initialized before starting the Initialization sequence.

This register's contents should not be changed while DDR2 interface is active. Also whenever this register contents are changed a initialization sequence must be executed to reflect this register contents in to the DDR2 mode register.

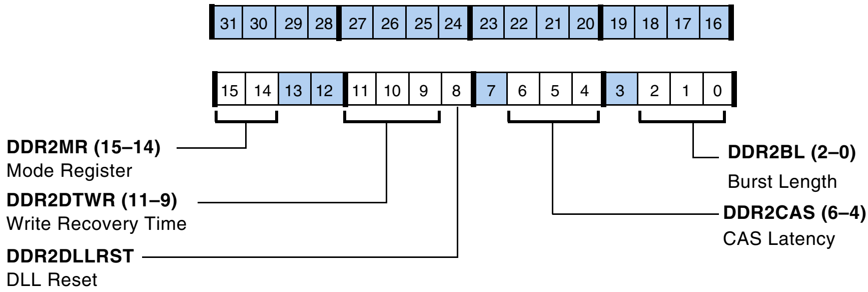

Figure A-20. DDR2CTL2 Register

Table A-19. DDR2CTL2 Register Bit Descriptions (RW)

Bit	Name	Description
2–0	DDR2BL	**Burst Length.** 010 = BL = 4 All other settings reserved.
3	DDR2BT	**Burst Type.** 0 = Sequential Other setting reserved.
6–4	DDR2CAS	**CAS Latency.** 000 = Reserved 001 = Reserved 010 = 2 clock cycles (default) … 111 = 7 clock cycles
7	Test mode	**Test Mode.** Bit cleared test mode is not supported.
8 (RW1S)	DDR2DLLRST	**DLL DDR2 Memory Reset.** Debug mode only. 0 = Normal 1 = Reset

Table A-19. DDR2CTL2 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
11–9	DDR2TWR	**Write Recovery Time.** 000 = Reserved 001 = 2 clock cycle 010 = 3 clock cycles (default) ... 110 = 7 clock cycles 111 = Reserved
13–12	Active Power down Exit	This bit cleared, slow exit power-down not supported.
15–14 (RO)	DDR2MR	**Mode Register.** Set to 00.
31–16	Reserved	

DDR2 Control Register 3 (DDR2CTL3)

The DDR2CTL3 register includes the programmable parameters associated with the DDR2 extended mode register (EMR1). Figure A-21 and Table A-20 show the bit definitions. All the values are defined in terms of number of clock cycles. Values written into this register are loaded into the DDR2 extended mode register during power up (or when Force EMR bit in DDR2CTL0 is set). This register should be initialized before starting the initialization sequence.

This register's contents should not be changed while DDR2 interface is active. Also, whenever this register's contents are changed, an initialization sequence must be executed to reflect this register contents in to the extended mode register.

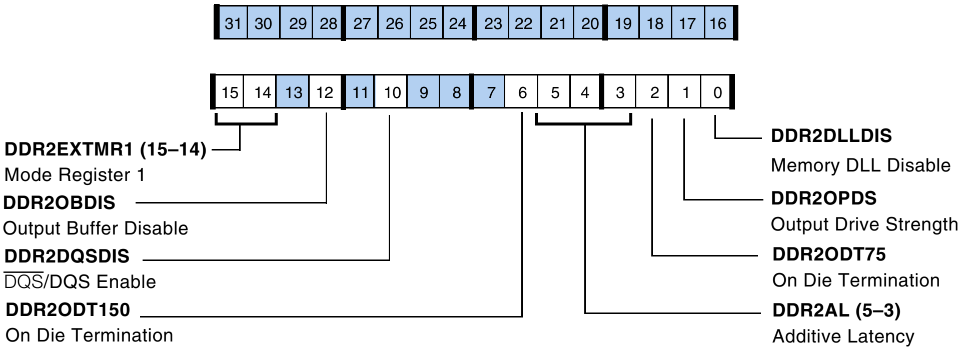

Figure A-21. DDR2CTL3 Register

Table A-20. DDR2CTL3 Register Bit Descriptions (RW)

Bit	Name	Description
0	DDR2DLLDIS	**DDR2 Memory DLL Disable.** Debug mode only. 0 = Enable DDR2 low/high byte DLLs 1 = Disable DDR2 low/high byte DLLs
1[1]	DDR2OPDS	**DDR2 Memory Output Drive Strength for Data and DQS.** 0 = Full strength 1 = Reduced strength Note the Output Drive Strength for DDR2 Controller is fixed to: – full strength for Data and DQS – half strength for CLK, ADDR and CMD
5–3	DDR2AL	**Additive Latency.** Additive latency reduces command bus conflicts to enable commands to be issued more efficiently. Note that the DDR2 controller performance is primary regardless of the AL settings. 000 = 0 clock cycles. 001 = 1 clock cycles. … 101 = 5 clock cycles. 110, 111 = Reserved. See “Additive Latency” on page 4-106.

Table A-20. DDR2CTL3 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
6, 2	DDR2ODT	**On Die Termination Value.** Bit 6 and 2 are required for truth table. 00 = ODT disabled 01 = 75 ohm 10 = 150 ohm 11 = 50 ohm
9–7	OCD Calibration Mode.	All bits cleared. OCD calibration not supported.
10	DDR2DQSDIS	**Differential $\overline{\text{DQS}}$/DQS Disable.** 0 = Enable DQS and $\overline{\text{DQS}}$ 1 = Disable $\overline{\text{DQS}}$ (default)
11	RDQS Disable	Bit cleared, x8 read DQS not supported.
12	DDR2OBDIS	**Output Buffer Disable.** 0 = Enable 1 = Disable
13	Future use	Bit cleared
15–14 (RO)	DDR2EXTMR1	**Extended Mode Register 1.** Set to 01.
31–16	Reserved	

1 When a program sets (or clears) the OPDS bit in the DDR2CTL3 register, the drive strengths of the DDR2 memory's data and strobe pins is driven by the memory device at reduced (or full) strength, respectively. By default, the bit is cleared (full drive strength). However, drive strengths of address, control, and differential clock pins which are outputs from the controller are fixed and not affected. Also note that from a controller standpoint, address, control, and clock signals are always driven at half drive strength, while data and strobe pins are always driven at full drive strength.

DDR2 Control Register 4 (DDR2CTL4)

The DDR2CTL4 register includes the programmable parameters associated with the DDR2 extended mode register 2 (EMR2). Table A-21 shows the DDR2 control register bit definition. All the values are defined in terms of number of clock cycles. Values written into this register are loaded into the DDR2 extended mode register 2 during power up (or when the force

EMR2 bit in the DDR2CTL0 register is set). This register should be initialized before starting the initialization sequence.

This register's contents should not be changed while DDR2 interface is active. Also whenever this register contents are changed an initialization sequence must be executed to reflect this register contents in to the DDR2 extended mode register 2.

Table A-21. DDR2CTL4 Register Bit Descriptions (RW)

Bit	Name	Description
13–0	Self Refresh Rate	(all bits cleared) 2x self-refresh rate high temp not supported
15–14 (RO)	DDR2EXTMR2	**Extended Mode Register 2.** Set to 10.
31–16	Reserved	

DDR2 Control Register 5 (DDR2CTL5)

The DDR2CTL5 register includes the programmable parameters associated with the DDR2 extended mode register 3 (EMR3).Table A-22 shows the DDR2 control register bit definition. All the values are defined in terms of number of clock cycles. Values written into this register are loaded into the DDR2EMR3 register during power up (or when the Force EMR3 bit in DDR2CTL0 is set). This register should be initialized before starting the initialization sequence.

This register's contents should not be changed while the DDR2 interface is active. Also, whenever this register's contents are changed an initialization sequence must be executed to reflect this register's contents in the DDR2EMR3 register.

Table A-22. DDR2CTL5 Register Bit Descriptions (RW)

Bit	Name	Description
13–0	Future use	All bits cleared
15–14 (RO)	DDR2EXTMR3	**Extended Mode Register 3.** Set to 11.
31–16	Reserved	

Refresh Rate Control Register (DDR2RRC)

The DDR2 refresh rate control register (Figure A-22 and Table A-23) provides a flexible mechanism for specifying the auto-refresh timing. For more information, see "Refresh Rate Control" on page 4-81.

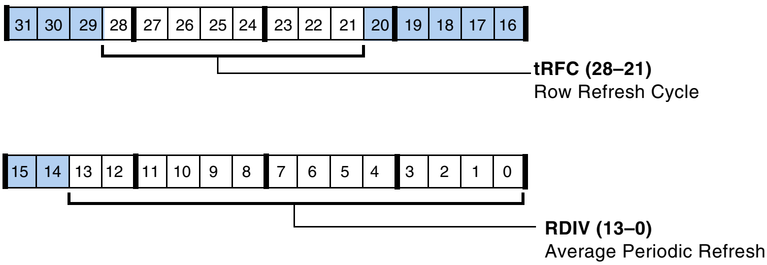

Figure A-22. DDR2RRC Register

Table A-23. DDR2RRC Register Bit Descriptions (RW)

Bit	Name	Description
13–0	RDIV	RDIV setting defines the average refresh interval between two subsequent refresh commands. The formula is shown in "Refresh Rate Control" on page 4-81. Note that the DDR2 manufacturer data sheets distinguish between commercial, industrial and automotive grades.
20–14	Reserved	

Table A-23. DDR2RRC Register Bit Descriptions (RW)

Bit	Name	Description
28–21	t_{RFC}	Row refresh cycle is the time after the refresh command to refresh a row. Programmable from 0 to 255 (0x14 default). Note that the DDR2 manufacturer data sheets distinguish between commercial, industrial and automotive grades.
31–29	Reserved	

Controller Status Register 0 (DDR2STAT0)

The register (Figure A-23 and Table A-24) provides information on the state of the controller. This information can be used to determine when it is safe to alter DDR2 controller control parameters or as a debug aid.

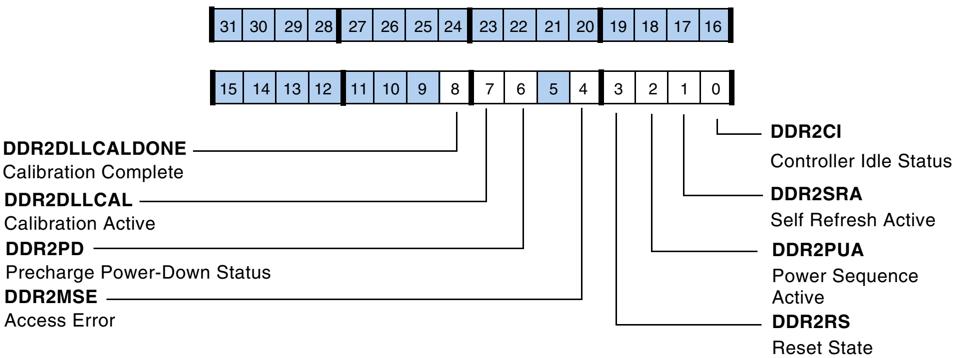

Figure A-23. DDR2STAT0 Register

Table A-24. DDR2STAT0 Register Bit Descriptions (RO)

Bit	Name	Description
0	DDR2CI	**Controller Idle Status.** 0 = Controller busy performing access or auto-refresh 1 = Controller idle (default)
1	DDR2SRA	**Self-Refresh Active.** 0 = Not in self-refresh mode 1 = Active
2	DDR2PUA	**Power-Up Sequence Active.** 0 = DDR2 not in power-up 1 = DDR2 in power-up initialization sequence
3	DDR2RS	**DLL Reset.** 0 = A power-up to DDR2 has been initialized since last DDR2 controller reset 1 = No power-up sequence occurred since last DDR2 controller reset (default)
4 (RW)	DDR2MSE	**Access Error (sticky bit).** 0 = No Error 1 = An access request to DDR2 occurred while the interface is disabled (DIS_DDR2CTL bit set in DDR2CTL0 register) Write a 0 to clear this bit (if set, sticky bit).
5	Reserved	
6	DDR2PD	**Precharge Power-Down Status.** 0 = Not in precharge power-down 1 = DDR2 in precharge power-down state (DIS_DDR2CKE bit set and DDR2CKE signal deasserted)
7	DDR2DLLCAL	**DLL External Bank Calibration Status.** 0 = Not in DLL calibration sequence 1 = DLL calibration active This bit is set during the DLL external bank calibration and cleared if finished.

Table A-24. DDR2STAT0 Register Bit Descriptions (RO) (Cont'd)

Bit	Name	Description
8	DDR2DLLCAL-DONE	**DLL External Bank Calibration After Reset Status.** 0 = A DLL calibration sequence is not happened since last DDR2 controller reset 1 = A DLL calibration sequence occurred since last DDR2 controller reset Note this bit is only set for first calibration after reset and remains set.
31–9	Reserved	

Controller Status Register 1 (DDR2STAT1)

This register reports the DDR2 bank active/idle status. This register is shown in Figure A-24 and described in Table A-25.

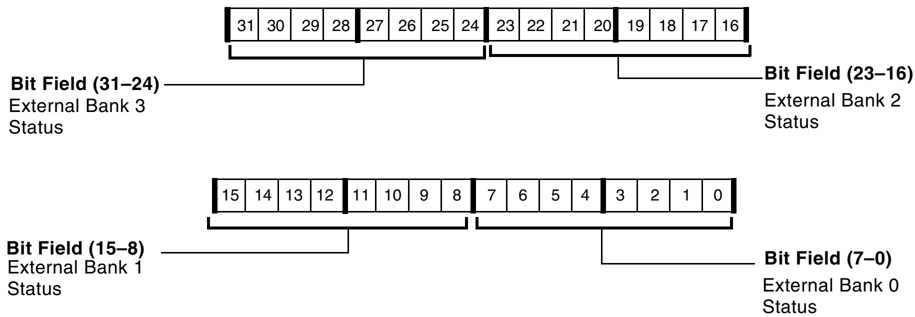

Figure A-24. DDR2STAT1 Register

Table A-25. DDR2STAT1 Register Bit Descriptions (RO)

Bit Field	Field Name	Description
7–0	External Bank 0 Status	**External Bank 0 Active/Precharge State.** xxxxxxx1 = Internal bank 0 in open state xxxxxxx0 = Internal bank 0 in precharge state xxxxxx1x = Internal bank 1 in open state xxxxxx0x = Internal bank 1 in precharge state … 1xxxxxxx = Internal bank 7 in open state 0xxxxxxx = Internal bank 7 in precharge state
15–8	External Bank 1 Status	**External Bank 0 Active/Precharge State.** xxxxxxx1 = Internal bank 0 in open state xxxxxxx0 = Internal bank 0 in precharge state xxxxxx1x = Internal bank 1 in open state xxxxxx0x = Internal bank 1 in precharge state … 1xxxxxxx = Internal bank 7 in open state 0xxxxxxx = Internal bank 7 in precharge state
23–16	External Bank 2 Status	**External Bank 0 Active/Precharge State.** xxxxxxx1 = Internal bank 0 in open state xxxxxxx0 = Internal bank 0 in precharge state xxxxxx1x = Internal bank 1 in open state xxxxxx0x = Internal bank 1 in precharge state … 1xxxxxxx = Internal bank 7 in open state 0xxxxxxx = Internal bank 7 in precharge state
31–24	External Bank 3 Status	**External Bank 0 Active/Precharge State.** xxxxxxx1 = Internal bank 0 in open state xxxxxxx0 = Internal bank 0 in precharge state xxxxxx1x = Internal bank 1 in open state xxxxxx0x = Internal bank 1 in precharge state … 1xxxxxxx = Internal bank 7 in open state 0xxxxxxx = Internal bank 7 in precharge state

DLL0 Control Register 1 (DLL0CTL1)

The DLL0CTL1 register shown in Figure A-25 and described in Table A-26 includes the programmable parameters associated with the DLL0 device. Note that it takes at least 9 core clock cycles to perform a DLL reset.

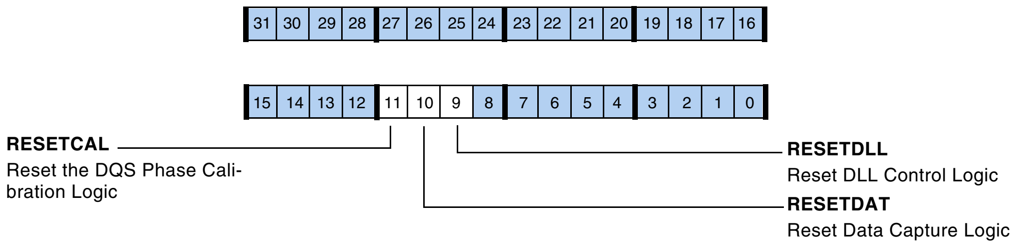

Figure A-25. DLL0CTL1 Register

Table A-26. DLL0CTL1 Register Bit Descriptions (RW)

Bit	Name	Description
8–0	Reserved	
9	RESETDLL	**Reset DLL Control Logic.** Active high, when active, it resets the DLL control logic only, including the 90 degree DQS shifter. 0 = No effect 1 = Reset DLL0 control logic
10	RESETDAT	**Reset Data Capture Logic.** Active high, when active, it resets the data capture logic only, including P and N buffers. 0 = No effect 1 = Reset DLL0 data capture logic
11	RESETCAL	Reset DQS Phase Calibration Logic. Active high, when active, it resets the DQS phase calibration logic. 0 = No effect 1 = Reset DLL0 DQS phase calibration logic
31–12	Reserved	

DLL1 Control Register 1 (DLL1CTL1)

The `DLL1CTL1` register shown in Figure A-26 and described in Table A-27 includes the programmable parameters associated with the DLL1 device. Note that it takes at least 9 core clock cycles to perform a DLL reset.

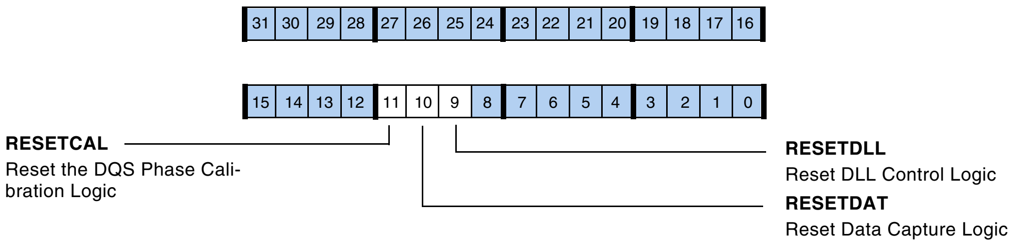

Figure A-26. DLL1CTL1 Register

Table A-27. DLL1CTL1 Register Bit Descriptions (RW)

Bit	Name	Description
8–0	Reserved	
9	RESETDLL	**Reset DLL Control Logic.** Active high, when active, it resets the DLL control logic only, including the 90 degree DQS shifter. 0 = No effect 1 = Reset DLL1 control logic
10	RESETDAT	**Reset Data Capture Logic.** Active high, when active, it resets the data capture logic only, including P and N buffers. 0 = No effect 1 = Reset DLL1 data capture logic
11	RESETCAL	Reset DQS Phase Calibration Logic. Active high, when active, it resets the DQS phase calibration logic. 0 = No effect 1 = Reset DLL1 DQS phase calibration logic
31–12	Reserved	

DLL Status Registers (DLL0STAT0, DLL1STAT0)

The DLL0STAT0 status register indicates the DLL lock status.

Table A-28. DLL0STAT0 Register Bit Descriptions (RW)

Bit	Name	Description
0–29	Reserved	
30	DLL_LOCKED	**Reset DLL Control Logic.** If this bit is set, indicates that the on-chip DLL for DDR2 controller has locked. After reset is de-asserted the DLL automatically locks to the default DDR2 CLK frequency even if the controller is not enabled.
31	Reserved	

DDR2 Pad Control Register 0 (DDR2PADCTL0)

The DDR2PADCTL0 register shown in Figure A-27 and described in Table A-29 includes the programmable parameters associated with the DDR2 DATA, DQS and DDR2CLK pads.

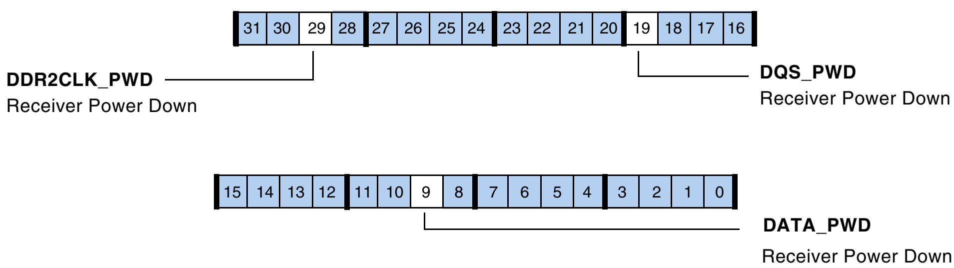

Figure A-27. DDR2PADCTL0 Register

Table A-29. DDR2PADCTL0 Register Bit Descriptions (RW)

Bit	Name	Description
8–0	Reserved	
9	DATA_PWD	**Data Pad Receiver Power Down.** 0 = Normal mode 1 = Power-down mode
18–10	Reserved	
19	DQS_PWD	**DQS Pad Receiver Power Down.** 0 = Normal mode 1 = Power-down mode
28–20	Reserved	
29	DDR2CLK_PWD	**Clock Pad Receiver Power Down.** 0 = Normal mode 1 = Power-down mode
31–30	Reserved	

DDR2 Pad Control Register 1 (DDR2PADCTL1)

The `DDR2PADCTL1` register shown in Figure A-28 and described in Table A-30 includes the programmable parameters associated with the DDR2 Command (`CS`, $\overline{\texttt{CAS}}$, $\overline{\texttt{RAS}}$, $\overline{\texttt{WE}}$, `ODT`) and Address pad control.

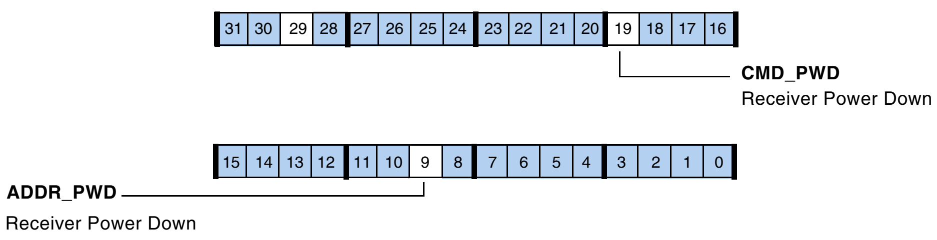

Figure A-28. DDR2PADCTL1 Register

Table A-30. DDR2PADCTL1 Register Bit Descriptions (RW)

Bit	Name	Description
8–0	Reserved	
9	ADDR_PWD	**Address Pad Receiver Power Down.** 0 = Normal mode 1 = Power-down mode
18–10	Reserved	
19	CMD_PWD	**Command Pad Receiver Power Down.** 0 = Normal mode 1 = Power-down mode
28–31	Reserved	

Peripheral Registers

The registers in the following sections are used for the peripherals that are not routed through the signal routing units (SRU, SRU2).

Link Port Registers

The following sections describe the link port status and control registers.

Control Register (LCTLx)

Figure A-29 and Table A-31 describe the bit fields within this register.

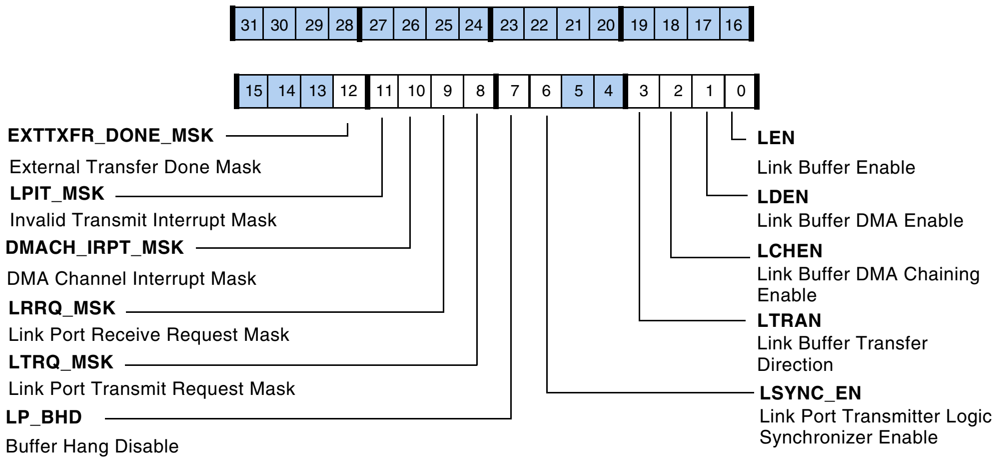

Figure A-29. LCTLx Registers

Table A-31. LCTLx Register Bit Descriptions (RW)

Bit	Name	Description
0	LEN	**Link Port Enable.** Enables if set (=1) or disables if cleared (=0) the link port. When the bit transitions from high to low the link buffer x is flushed which takes 2 core clock cycles. The corresponding LSTAT and LRERR bits are also cleared.
1	LDEN	**Link Buffer DMA Enable.** Enables (if set, =1) or disables (if cleared, = 0) DMA transfers link buffer x (LBUF).
2	LCHEN	**Link Buffer DMA Chaining Enable.** Enables (if set, =1) or disables (if cleared, =0) DMA chaining link buffer x (LBUF).
3	LTRAN	**Link Buffer Transfer Direction.** This bit selects the transfer direction (transmit if set, =1) (receive if cleared, = 0) for link buffer x (LBUF).
5–4	Reserved	
6	LSYNC_EN	**Link Port Transmitter Logic Synchronizer Enable.** Enables the synchronizer logic within the link port transmitter. This bit is undefined for ADSP-2146x rev 0.0 and is only available for silicon rev 0.1 and beyond. See the processor IC anomaly list available on the web. 0 = Link port transmitter logic is not enabled. 1 = Link port transmitter logic is enabled.

Table A-31. LCTLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
7	LP_BHD	**Buffer Hang Disable.** 0 = Core stalls when read from empty receive or write to full transmit buffer attempted 1 = Prevents a core hang.
8	LTRQ_MSK	**Transmit Request Mask.** 0 = Mask 1 = Unmask
9	LRRQ_MSK	**Receive Request Mask.** 0 = Mask 1 = Unmask
10	DMACH_ IRPT_MSK	**DMA Channel Count Interrupt Mask.** Must be set to generate interrupt if DMA count is zero and is compatible with traditional SHARC processors. 0 = Mask 1 = Unmask
11	LPIT_MASK	**Invalid Transmit Interrupt Mask.** 0 = Mask 1 = Unmask
12	EXTTXFR_- DONE_ MSK	**External Transfer Done Interrupt Mask.** Valid for core and DMA accesses. If set interrupt is generated when the FIFO is empty. Note if bit 10 is also set for DMA, two interrupts are generated, one for DMA count = 0 and one for FIFO empty. 0 = Mask 1 = Unmask
31–13	Reserved	

Status Registers (LSTATx)

Figure A-30 and Table A-32 describe the bit fields within this register.

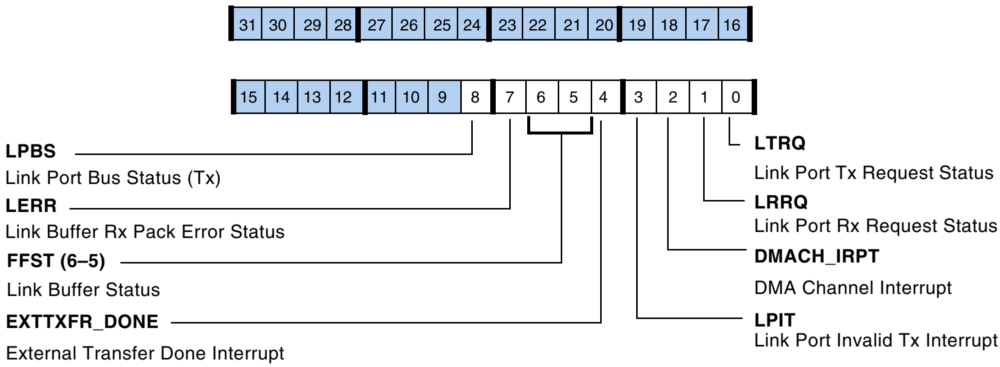

Figure A-30. LSTATx Register

Table A-32. LSTATx Register Bit Descriptions (RO)

Bit	Name	Description
0 (ROC)	LTRQ	**Transmit Request Status.** Indicates when another processor is attempting to send data through a particular link port. Two processors can communicate without prior knowledge of the transfer direction, link port number, or exactly when the transfer is to occur.
1 (ROC)	LRRQ	**Receive Request Status.** Indicates when another processor is attempting to receive data through a particular link port. Two processors can communicate without prior knowledge of the transfer direction, link port number, or exactly when the transfer is to occur.
2 (ROC)	DMACH_IRPT	**DMA Channel Count Interrupt.** An internal transfer complete interrupt is generated by the transmitter once the word count is zero by setting bit 10 (DMACH_IRPT_MSK) in the LCTL register. When DMA is not enabled, this interrupt is generated when the word count is zero. Also, when DMA is enabled, the DMA engine checks if DMA has been completed.

Table A-32. LSTATx Register Bit Descriptions (RO) (Cont'd)

Bit	Name	Description
3 (ROC)	LPIT	**Invalid Transmit Interrupt.** Generated when the transmitter is driving LCLKx high because the receiver has not asserted LACKx and LCLKx goes low due to a processor reset.
4 (ROC)	EXTTXFR_DONE	**External Transfer Done Interrupt.** Generated by the transmitter once the external transfer is completed
6–5	FFST	**Link Buffer Status.** 00 = empty, 01 = reserved, 10 = one word, 11 = full (Cleared when the Link Port is disabled)
7	LERR	**Link Buffer Receive Pack Error Status.** 0 = Packing complete 1 = Packing incomplete
8	LPBS	**Bus Status (Transmitter).** To safely disable link port transmit operation first poll the FFST bit and second the LPBS bit. 0 = Bus is idle 1 = Bus busy
31–9	Reserved	

Memory-to-Memory DMA Register

The following DMA related registers are used when performing internal-to-internal DMA through the MTM port.

DMA Control (MTMCTL Register)

The `MTMCTL` register (Figure A-31) allows programs to transfer blocks of 64-bit data from one internal memory location to another. The MTM module must set the `MTMFLUSH` bit to clear the buffer which takes 6 core clock cycles

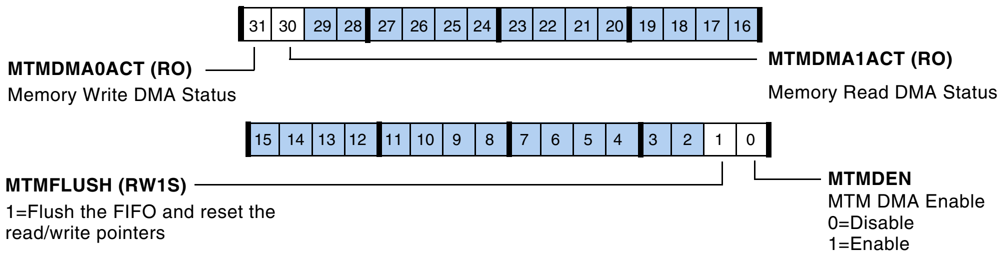

Figure A-31. MTMCTL Register (RW)

Pulse Width Modulation Registers

The following registers control the operation of pulse width modulation on the processor.

Global Control Register (PWMGCTL)

This register enables or disables the four PWM groups, in any combination and provides synchronization across the groups. Note that disable bits have higher priority over the enable bits (bit 1 higher as bit 0 and so on). This 16-bit register is shown in Figure A-32.

For the PWM global control register, the traditional read-modify-write operations to disable the PWM group have changed. The action is to directly write—this simplifies the enable/disable of the PWM groups and can be done with fewer instructions. For example, instead of the following code:

```
ustat3 = dm(PWMGCTL);         /* PWM General Control Register */
bit set ustat3 PWM_DIS0;      /* disables group 0 */
dm(PWMGCTL) = ustat3;
```

Use:

```
ustat3 = PWM_DIS0;
dm(PWMGCTL) = ustat3;
```

Writes to the enable and disable bit-pairs for a PWM group works as follows.

`PWM_DISx` = 0, `PWM_ENx` = 0 – No action
`PWM_DISx` = 0, `PWM_ENx` = 1 – Enable the PWM group
`PWM_DISx` = 1, `PWM_ENx` = x – Disable the PWM group

For reads, the interpretation is as follows.

`PWM_DISx` = 0, `PWM_ENx` = 0 – PWM group is disabled
`PWM_DISx` = 1, `PWM_ENx` = 1 – PWM group is enabled

Any other read combination is not possible. Reads of the `PWMGCTL` register returns the enable status on both the enable and disable bits.

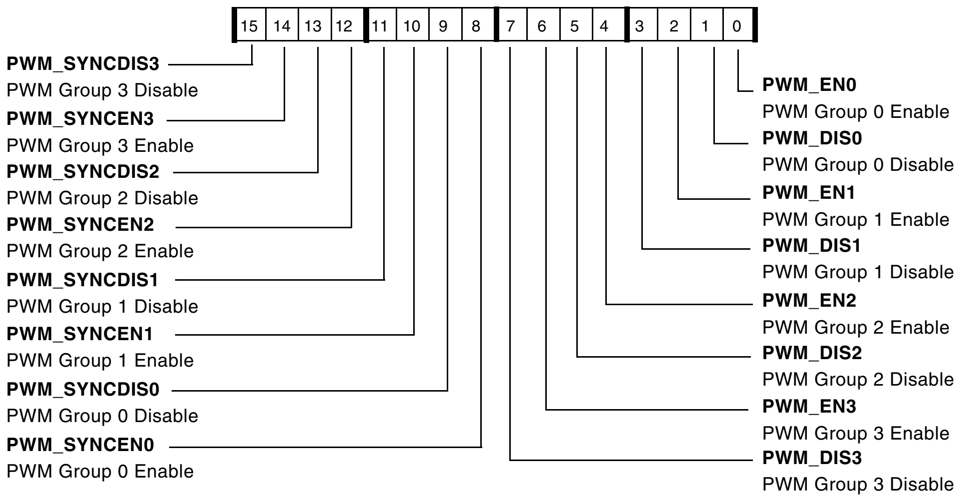

Figure A-32. PWMGCTL Register

Table A-33. PWMGCTL Register Bit Descriptions (RW)

Bit	Name	Function
0, 2. 4, 6	PWM_ENx0	PWM Group x Enable
1, 3, 5, 7	PWM_DISx	PWM Group x Disable
8, 10, 12, 14	PWM_SYNCENx	PWM Group x Enable
9, 11, 13, 15	PWM_SYNCDISx	PWM Group x Disable

Global Status Register (PWMGSTAT)

This register provides the status of each PWM group (Table A-34). The status bits are set depending on the IRQEN bit. The ISR needs to write one to clear the status bits.

Table A-34. PWMGSTAT Register Bit Descriptions (RW1C)

Bit	Name	Function
0	PWM_STAT0	PWM Group 0 Period Completion Status
1	PWM_STAT1	PWM Group 1 Period Completion Status
2	PWM_STAT2	PWM Group 2 Period Completion Status
3	PWM_STAT3	PWM Group 3 Period Completion Status
15–4	Reserved	

Control Register (PWMCTLx)

These registers, shown in Figure A-33 and described in Table A-35, are used to set the operating modes of each PWM block. They also allow programs to disable interrupts from individual groups.

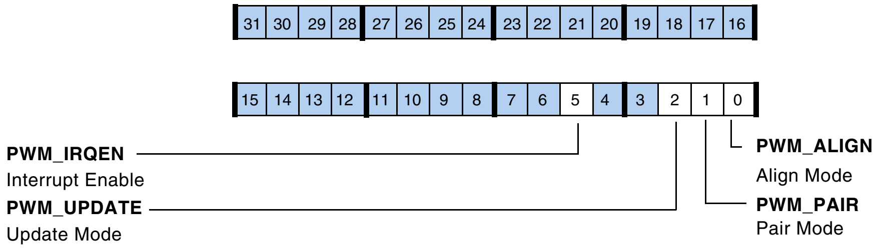

Figure A-33. PWMCTLx Register

Table A-35. PWMCTLx Register Bit Descriptions (RW)

Bit	Name	Description
0	PWM_ALIGN	**Align Mode.** 0 = Edge-aligned. The PWM waveform is left-justified in the period window. 1 = Center-aligned. The PWM waveform is symmetrical.
1	PWM_PAIR	**Pair Mode.** 0 = Non-paired mode. The PWM generates independent signals (for example xH, xL) 1 = Paired mode. The PWM generates the complementary signal from the high side output (xL=/xH).
2	PWM_UPDATE	**Update Mode.** 0 = Single update mode. The duty cycle values are programmable only once per PWM period. The resulting PWM patterns are symmetrical about the mid-point of the PWM period. 1 = Double update mode. A second update of the PWM registers is implemented at the mid-point of the PWM period. PWM_UPDATE mode has only effect for center aligned mode (PWM_ALIGN=1).
4–3	Reserved	
5	PWM_IRQEN	**Enable PWM Interrupts.** 0 = Interrupts not enabled 1 = Interrupts enabled
31–6	Reserved	

Status Registers (PWMSTATx)

These 16-bit registers, described in Table A-36, report the status of the phase and mode for each PWM group.

Table A-36. PWMSTATx Register Bit Descriptions (RO)

Bit	Name	Description
0	PWM_PHASE	**PWM Phase Status.** Set during center aligned mode in the second half of each PWM period. Allows programs to determine the particular half-cycle (first or second) during PWM interrupt service routine, if required. 0 = First half 1 = Second half (default) In edge aligned mode this bit is always set.
1	Reserved	
2	PWM_PAIRSTAT	**PWM Paired Mode Status.** 0 = Inactive paired mode 1 = Active paired mode
15–3	Reserved	

Output Disable Registers (PWMSEGx)

These 16-bit registers, shown in Figure A-34 and described in Table A-37, control the output signals of the four PWM groups. The output signals are enabled by default.

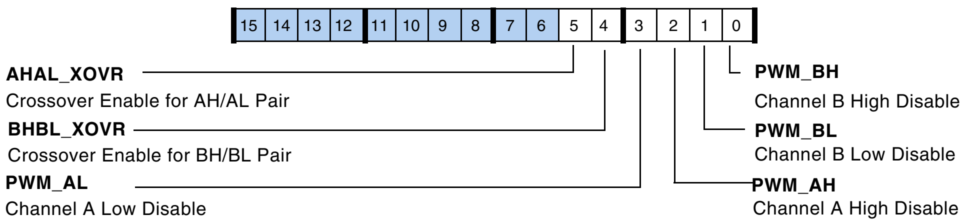

Figure A-34. PWMSEGx Register

Table A-37. PWMSEGx Register Bit Descriptions (RW)

Bit	Name	Description
0	PWM_BH	**Channel B High Disable.** Enables or disables the channel B output signal. 0 = Enable 1 = Disable
1	PWM_BL	**Channel B Low Disable.** Enables or disables the channel B output signal. 0 = Enable 1 = Disable
2	PWM_AH	**Channel A High Disable.** Enables or disables the channel A output signal. 0 = Enable 1 = Disable
3	PWM_AL	**Channel A Low Disable.** Enables or disables the channel A output signal. 0 = Enable 1 = Disable
4	BHBL_XOVR	**Crossover Enable for BH/BL Pair.** 0 = Disable 1 = Enable
5	AHBL_XOVR	**Crossover Enable for AH/AL Pair.** 0 = Disable 1 = Enable
15–6	Reserved	

Polarity Select Registers (PWMPOLx)

These 16-bit registers, described in Table A-38, control the polarity of the four PWM groups which can be set to either active high or active low. Note that bit 1 has priority over bit 0, bit 3 over bit 2 and so on. In paired mode, it is expected to maintain polarity coherency by setting the same polarity for both the high and low side of a PWM pair.

Table A-38. PWMPOLx Register Bit Descriptions (RW)

Bit	Name	Description
0	PWM_POL1AL	**Channel AL Polarity 1.** 0 = Channel AL polarity 0 1 = Channel AL polarity 1 (default)
1	PWM_POL0AL	**Channel AL Polarity 0.** 0 = Channel AL polarity 0 1 = Channel AL polarity 1 (default)
2	PWM_POL1AH	**Channel AH Polarity 1.** 0 = Channel AH polarity 0 1 = Channel AH polarity 1 (default)
3	PWM_POL0AH	**Channel AH Polarity 0.** 0 = Channel AH polarity 0 1 = Channel AH polarity 1 (default)
4	PWM_POL1BL	**Channel BL Polarity 1.** 0 = Channel AL polarity 0 1 = Channel AL polarity 1 (default)
5	PWM_POL0BL	**Channel BL Polarity 0.** 0 = Channel AL polarity 0 1 = Channel AL polarity 1 (default)
6	PWM_POL1BH	**Channel BH Polarity 1.** 0 = Channel BH polarity 0 1 = Channel BH polarity 1 (default)
7	PWM_POL0BH	**Channel BH Polarity 0.** 0 = Channel BH polarity 0 1 = Channel BH polarity 1 (default)
15–8	Reserved	

Period Registers (PWMPERIODx)

These 16-bit RW registers control the unsigned period of the four PWM groups. This register is double buffered for double update mode. A change in one half cycle of PWM switching period only takes effect in the next half period.

Duty Cycle High Side Registers (PWMAx, PWMBx)

The 16-bit duty-cycle control registers (RW) directly control the A/B (two's-complement) duty cycles of the two pairs of PWM signals.

Duty Cycle Low Side Registers (PWMALx, PWMBLx)

The 16-bit duty-cycle control registers (RW) directly control the AL/BL duty cycles (two's-complement) of the non-paired PWM signals. These can be different from the AH/BH cycles.

Dead Time Registers (PWMDTx)

These 16-bit RW registers set up a short time delay (10-bit, unsigned) between turning off one PWM signal and turning on its complementary signal.

Debug Status Registers (PWMDBGx)

These 16-bit registers aid in software debug activities.

Table A-39. PWMDBGx Register Bit Descriptions (RO)

Bit	Name	Function
0	PWM_AL	Channel A Low Output Signal for S/W Observation
1	PWM_AH	Channel A High Output Signal for S/W Observation
2	PWM_BL	Channel B Low Output Signal for S/W Observation
3	PWM_BH	Channel B High Output Signal for S/W Observation
15:4	Reserved	

FFT Accelerator Registers

The following sections describe the registers used to program and debug the FFT accelerator.

General Control Register (FFTCTL1)

The global control register (FFTCTL1) shown in Figure A-35 and described in Table A-39 is used to enable, start, and reset the FFT module. It is also used to enable DMA and debug operation.

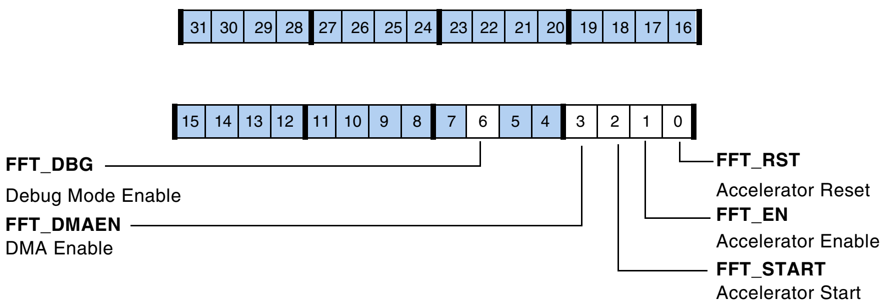

Figure A-35. FFTCTL1 Register

Table A-40. FFTCTL1 Register Bit Descriptions (RW)

Bits	Name	Description
0	FFT_RST	**Reset Accelerator.** Setting this bit puts the accelerator into reset mode. Explicit clearing of this bit is necessary to take the accelerator out of reset. 0 = Normal operation 1 = Reset–the input/output buffers are immediately flushed.
1	FFT_EN	**Accelerator Enable.** 0 = Disable 1 = Enable
2	FFT_START	**Start Accelerator.** 0 = Idle 1 = Start

Table A-40. FFTCTL1 Register Bit Descriptions (RW) (Cont'd)

Bits	Name	Description
3	FFT_DMAEN	**DMA Enable.** 0 = Disable 1 = Enable
5–4	Reserved	
6	FFT_DBG	**Debug Mode Enable.** 0 = Disable 1 = Enable
31–7	Reserved	

Control Register (FFTCTL2)

The FFT control register, shown in Figure A-36 and described in Table A-41, is used to set up individual FFT parameters (such as length) and how the module process the FFT, such as data packing.

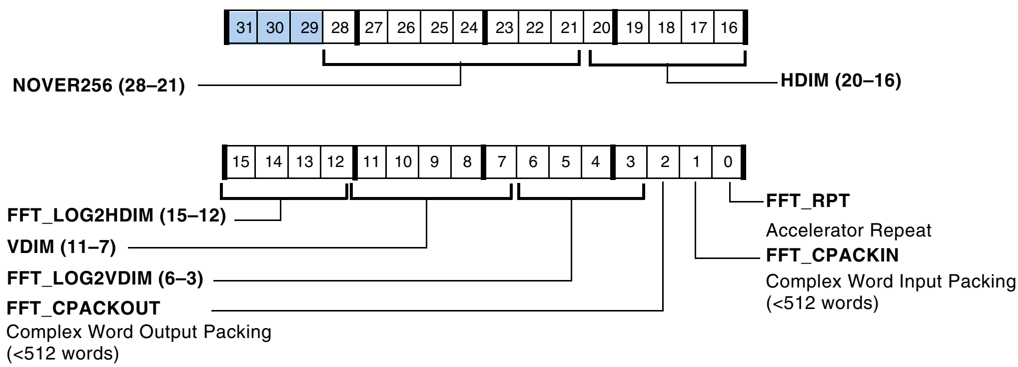

Figure A-36. FFTCTL2 Register

Table A-41. FFTCTL2 Register Bit Descriptions (RW)

Bits	Name	Description
0	FFT_RPT	**Accelerator Repeat.** See "No Repeat Mode" and "Repeat Mode" on page 7-13 0 = Go to idle when done 1 = Repeat when done
1	FFT_CPACKIN	**Complex Word Input Packing for <512 Words.** 0 = No packing, first all reals, then all imag are received by the accelerator 1 = Complex numbers are packed Real/Imag. For >256 words, this bit is always set.
2	FFT_CPACKOUT	**Complex Word Output Packing for <512 Words.** 0 = No packing, first all reals, then all imag are sent by the accelerator 1 = Complex numbers are packed Real/Imag. For >256 words, this bit is always set.
6–3	FFT_LOG2VDIM	**Log2 (VDIM).**
11–7	VDIM	**VDIM/16.** Vertical Column Dimension of the FFT computation matrix. V = 16, VDIM = 1 V = 32, VDIM = 2 V = 64, VDIM = 3 V = 128, VDIM = 4 V = 256, VDIM = 5
15–12	FFT_LOG2HDIM	**Log2 (HDIM).**
20–16	HDIM	**HDIM/16.** Horizontal Column Dimension of the FFT computation matrix. For small FFTs, (<512 points) only VDIM required HDIM = 0.
28–21	NOVER256	N/256 = HDIM × VDIM (used for large FFTs)
31–29	Reserved	

Multiplier Status Register (FFTMACSTAT)

The `FFT_MACSTAT` register, described in Table A-42, can be written only in debug mode. The status bits are sticky and are cleared when read.

Table A-42. FFT_MACSTAT Register Bit Descriptions (ROC)

Bits	Name	Description
0	FFT_NAN	Bits 3–0 follow the IEEE STD for floating point numbers.
1	FFT_DENORM	
2	FFT_OVR	
3	FFT_UDR	
31–4	Reserved	

DMA Status Register

The bits in the status register, described in Table A-43 report DMA status information.

Table A-43. FFTDMASTAT Register Bit Descriptions (RO)

Bits	Name	Description
0	ICPLD	Input Chain Pointer Loading
1	IDMASTAT	Input DMA in Progress
2 (ROC)	IDMACHIRPT	Input DMA Channel Interrupt
3	OCPLD	Output Chain Pointer Loading
4	ODMASTAT	Output DMA in Progress
5 (ROC)	ODMACHIRPT	Output DMA Channel Interrupt
31–6	Reserved	

Debug Registers (FFTDADDR, FFTDDATA)

Bits 31–0 is the FFT_DDATA register correspond to the data to be read or written. When a data write is performed first this register is loaded with data which needs to be written, then the FFT_DADDRESS register is loaded with the write address of the location. Note that these registers should be written/read only in debug mode. In Table A-44, A is a meaningful address bit.

Table A-44. DADDRESS Register Bit Descriptions (RW)

Bits	Name	Description
12–0	ADDRESS	**Address Bit.** Access to local memory requires debug mode. The MSB bits of the address decode the memory location. 000AAAAAAAAAA = read data memory (2^10) 100AAAAAAAAAA = write data memory (2^10) 010xAAAAAAAAA = read coefficient memory (2^9) 110xAAAAAAAAA = write coefficient memory (2^9) A = valid address bits
31–13	Reserved	

FIR Accelerator Registers

The following sections describe the registers used to program and debug the FIR accelerator.

Global Control Register (FIRCTL1)

The FIRCTL1 register, shown in Figure A-37 and described in Table A-45, is used to configure the global parameters for the accelerator. These include the number of channels, channel auto iterate, DMA enable, and accelerator enable.

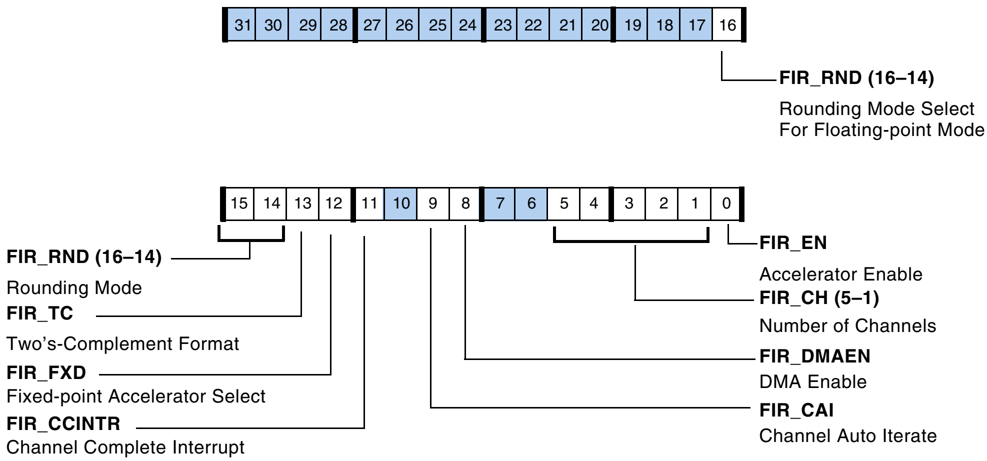

Figure A-37. FIRCTL1 Register

Table A-45. FIRCTL1 Register Bit Descriptions (RW)

Bits	Name	Description
0	FIR_EN	**FIR Enable.** 0 = FIR disabled 1 = FIR enabled
5–1	FIR_CH32–1	**Number of Channels.** Programmable between 0–31 0 = FIR_CH1 31 = FIR_CH32
7–6	Reserved	
8	FIR_DMAEN	**DMA Enable.** 0 = DMA disabled 1 = DMA enabled
9	FIR_CAI	**Channel Auto Iterate.** 0 = TDM Processing stops (idle) once all channels are over 1 = Moves to first channel and continues TDM processing in a loop when all channels are over
10	Reserved	

Table A-45. FIRCTL1 Register Bit Descriptions (RW) (Cont'd)

Bits	Name	Description
11	FIR_CCINTR	**Channel Complete Interrupt.** 0 = Interrupt is generated only when all channels are done 1= Interrupt is generated after each channel is done
12	FIR_FXD	**Fixed-Point Accelerator Select.** 0 = 32-bit IEEE floating-point 1 = 32-bit fixed point
13	FIR_TC	**Two's-Complement Format Input Select For Fixed-Point Mode.** 0 = Unsigned integer 1 = Signed integer
16–14	FIR_RND	**Rounding Mode Select For Floating-Point Mode.** 000 = IEEE round to nearest (even) 001 = IEEE round to zero 010 = IEEE round to +ve infinity 011 = IEEE round to -ve infinity 100 = Round to nearest Up 101 = Round away from zero 110 = Reserved 111 = Reserved
31–17	Reserved	

Channel Control Register (FIRCTL2)

The `FIRCTL2` register, shown in Figure A-38 and described in Table A-46, is used to configure the channel specific parameters such as filter TAP length, window size, sample rate conversion, up/down sampling and ratio.

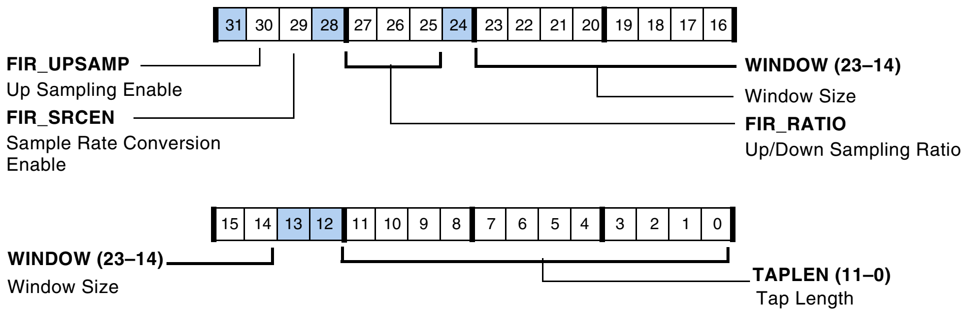

Figure A-38. FIRCTL2 Register

Table A-46. FIRCTL2 Register Bit Descriptions (RW)

Bits	Name	Description
11–0	TAPLEN	**Tap Length.** Programmable between 0–4095 Tap Length = TAPLEN + 1
13–12	Reserved	
23–14	WINDOW	**Window Size.** Window size specifies the number of sample/block to process (sample based processing = window size of 1)
24	Reserved	
27–25	FIR_RATIO	**UP/DOWN Sampling Ratio.** Sampling Ratio = RATIO + 1
28	Reserved	
29	FIR_SRCEN	**Sample Rate Conversion Enable.** 0 = Disabled 1 = Enabled
30	FIR_UPSAMP	**Up Sampling Enable.** 0 = Down Sampling 1 = Up sampling
31	Reserved	

FIR MAC Status Register (FIRMACSTAT)

This register, shown in Figure A-39 and described in Table A-47, provides the status of MAC operations. The status of all four multipliers/adders are available separately for programs to poll. In fixed-point mode only the ARIx bits are used (all other bits are reserved).

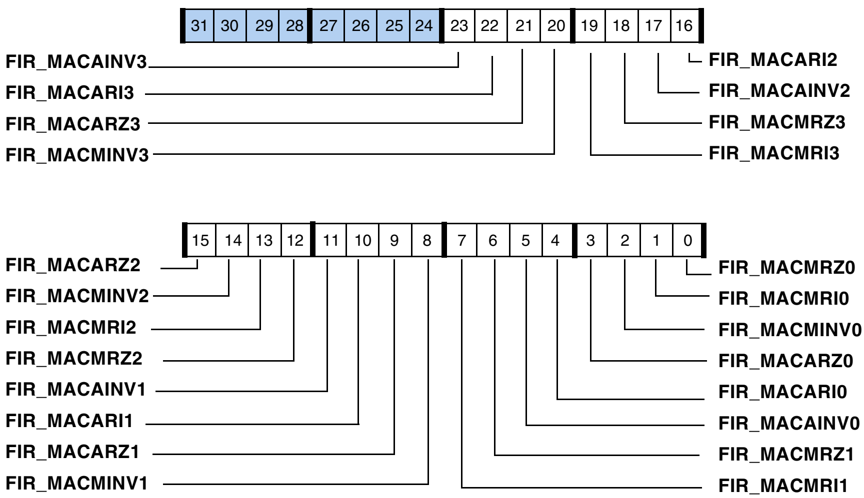

Figure A-39. FIRMACSTAT Register

Table A-47. FIRMACSTAT Register Bit Descriptions (RO)

Bits	Name	Description
0	FIR_MACMRZ0	**Multiplier Result Zero.** Set if multiplier 0 results is zero.
1	FIR_MACMRI0	**Multiplier Result Infinity.** Set if multiplier 0 results is infinity.
2	FIR_MACMINV0	**Multiply Invalid.** Set if multiplier 0 multiply operation is invalid.
3	FIR_MACARZ0	**Adder Result Zero.** Set if a adder 0 results is zero.
4	FIR_MACARI0	**Adder Result Infinity.** Set if adder 0 results is infinity. Indicates overflow in fixed-point mode.

Table A-47. FIRMACSTAT Register Bit Descriptions (RO) (Cont'd)

Bits	Name	Description
5	FIR_MACAINV0	**Addition Invalid.** Set if a adder 0 addition is invalid.
6	FIR_MACMRZ1	**Multiplier Result Zero.** Set if multiplier 1 results is zero.
7	FIR_MACMRI1	**Multiplier Result Infinity.** Set if multiplier 1 results is infinity.
8	FIR_MACMINV1	**Multiply Invalid.** Set if multiplier 1 multiply operation is invalid.
9	FIR_MACARZ1	**Adder Result Zero**. Set if a adder 1 results is zero.
10	FIR_MACARI1	**Adder Result Infinity.** Set if adder 1 results is infinity. Indicates overflow in fixed-point mode.
11	FIR_MACAINV1	**Addition Invalid.** Set if a adder 1 addition is invalid.
12	FIR_MACMRZ2	**Multiplier Result Zero.** Set if multiplier 2 results is zero.
13	FIR_MACMRI2	**Multiplier Result Infinity.** Set if multiplier 2 results is infinity.
14	FIR_MACMINV2	**Multiply Invalid.** Set if multiplier 2 multiply operation is invalid.
15	FIR_MACARZ2	**Adder Result Zero**. Set if a adder 2 results is zero.
16	FIR_MACARI2	**Adder Result Infinity.** Set if adder 2 results is infinity. Indicates overflow in fixed-point mode.
17	FIR_MACAINV2	**Addition Invalid.** Set if a adder 2 addition is invalid.
18	FIR_MACMRZ3	**Multiplier Result Zero.** Set if multiplier 3 results is zero.
19	FIR_MACMRI3	**Multiplier Result Infinity.** Set if multiplier 3 results is infinity.
20	FIR_MACMINV3	**Multiply Invalid.** Set if multiplier 3 multiply operation is invalid.
21	FIR_MACARZ3	**Adder Result Zero**. Set if a adder 3 results is zero.
22	FIR_MACARI3	**Adder Result Infinity.** Set if adder 3 results is infinity. Indicates overflow in fixed-point mode.
23	FIR_MACAINV3	**Addition Invalid.** Set if a adder 3 addition is invalid.
31–24	Reserved	

FIR DMA Status Register (FIRDMASTAT)

The information provided by this register, shown in Figure A-40 and described in Table A-48, are, chain pointer loading, coefficient DMA, data preload DMA, processing in progress, window complete, all channels complete.

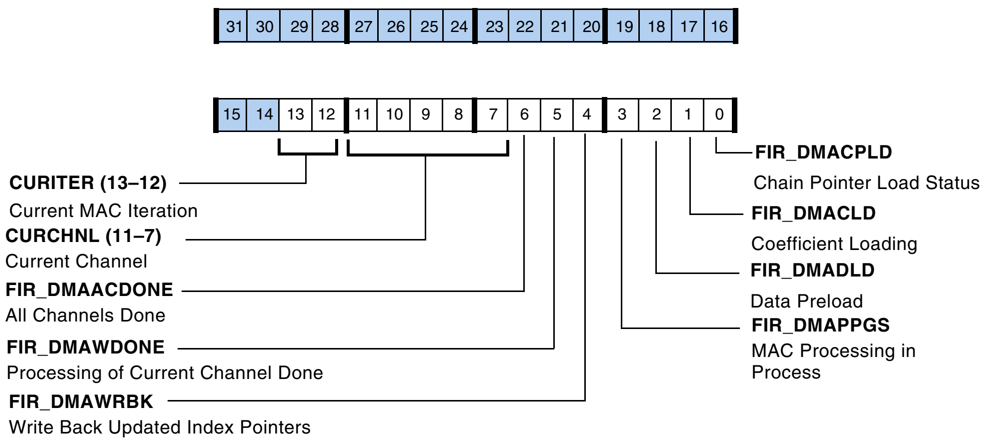

Figure A-40. FIRDMASTAT Register

Table A-48. FIRDMASTAT Register Bit Descriptions (RO)

Bits	Name	Description
0	FIR_DMACPLD	**Chain Pointer Loading Status.** High indicates state machine in chain pointer load state.
1	FIR_DMACLD	**Coefficient Loading.**
2	FIR_DMADLD	**Data Preload.**
3	FIR_DMAPPGS	**MAC Processing in Progress.**
4	FIR_DMAWRBK	**Writing Back the Updated Index Registers.**
5 (ROC)	FIR_DMAWDONE	**Processing of Current Channel Done.** (Sticky – Cleared on register read). The FIR_CCINTR bit will not affect the FIR_DMAWDONE Status Bit

Table A-48. FIRDMASTAT Register Bit Descriptions (RO)

Bits	Name	Description
6 (ROC)	FIR_DMAACDONE	**All Channels Done.** (Sticky – Cleared on register read). The FIR_CCINTR bit will not affect the FIR_DMAAC-DONE Status Bit.
11–7	CURCHNL	**Current Channel.** Channel that is being processed in the TDM slot. Zero indicates the last slot.
13–12	CURITER	**Current MAC Iteration.** Current MAC iteration in multi iteration mode. Zero indicates the final iteration.
31–14	Reserved	

FIR Debug Registers (FIRDEBUGCTL, FIRDBGADDR)

This register, shown in Figure A-41 and described in Table A-49, control the debug operation of the FIR accelerator and should only be used in debug mode.

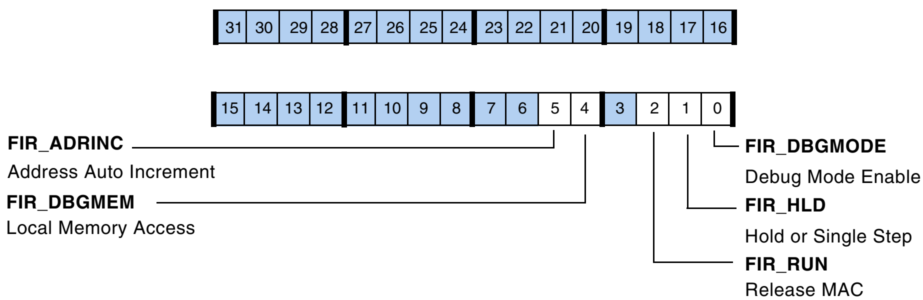

Figure A-41. FIRDEBUGCTL Register

Table A-49. FIRDEBUGCTL Register Bit Descriptions (RW)

Bits	Name	
0	FIR_DBG-MODE	**Debug Mode Enable.** 0 = Disable 1 = Enable For local memory access, the FIRCTL1 register can be cleared.
1	FIR_HLD	**Hold Or Single Step.** The function of this bit is based on the FIR_DBGMEM bit setting. For FIR_DBGMEM = 0: 1 = Single step for FIR_DBGMEM = 1: 1 = Hold data
2 (RW1S)	FIR_RUN	**Release MAC.** This bit is self clearing after one FIR clock cycle.
3	Reserved	
4	FIR_DBG-MEM	**Local Memory Access.** If set, the data and coefficients memory can be indirectly accessed.
5	FIR_ADRINC	**Address Auto Increment.** If this bit is set, the address register auto increments on DBGMEMWRDAT write and DBGMEMRDDAT reads.
31–6	Reserved	

IIR Accelerator Registers

The following sections describe the registers used to program and debug the FIR accelerator.

IIR Global Control Register (IIRCTL1)

The `IIRCTL1` register, shown in Figure A-42 and described in Table A-50, is used to configure the global parameters for the accelerator. These include number of channels, channel auto iterate, DMA enable, and accelerator enable.

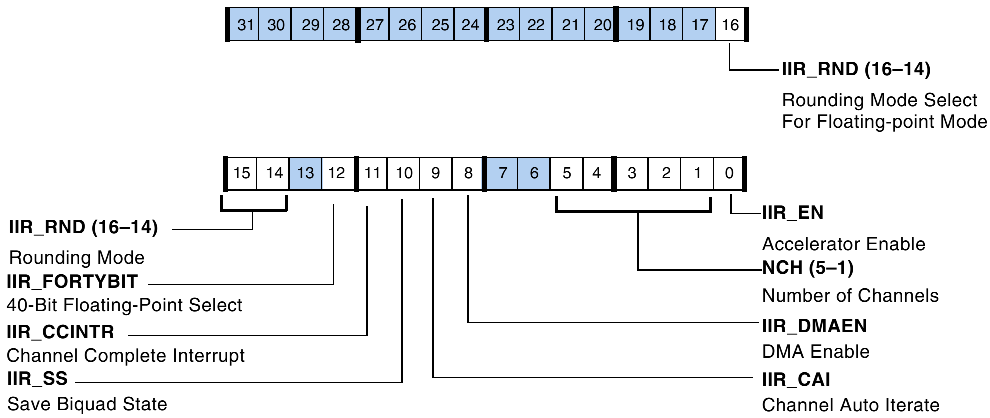

Figure A-42. IIRCTL1 Register

Table A-50. IIRCTL1 Register Bit Descriptions (RW)

Bits	Name	Description
0	IIR_EN	**IIR Enable.** 0 = IIR disabled 1 = IIR enabled
5–1	IIR_NCH	**Number of Channels.** Programmable between 0–23 Channels = NCH + 1
7–6	Reserved	
8	IIR_DMAEN	**DMA Enable.** 0 = Disable 1 = Enable
9	IIR_CAI	**Channel Auto Iterate.** 0 = TDM processing stops (idle) once all channels complete processing 1 = Moves to first channel and continues TDM processing in a loop when all channels complete processing
10	IIR_SS	**Save Biquad State.** Stores the Dk registers settings into the internal memory. This can be used to save the biquad states before switching to another high priority accelerator task.

Table A-50. IIRCTL1 Register Bit Descriptions (RW) (Cont'd)

Bits	Name	Description
11	IIR_CCINTR	**Channel Complete Interrupt.** 0 = Interrupt is generated only when all channels are done (default) 1 = Interrupt is generated after each channels is done (default)
12	IIR_FORTYBIT	**40-Bit Floating-Point Format Select.** 0 = 32-bit IEEE floating-point 1 = 40-bit IEEE floating-point
13	Reserved	
16–14	IIR_RND	**Rounding Mode Select For Floating-point Mode.** 000 = IEEE round to nearest (even) 001 = IEEE round to zero 010 = IEEE round to +ve infinity 011 = IEEE round to -ve infinity 100 = Round to nearest Up 101 = Round away from zero 110 = Reserved 111 = Reserved
31–17	Reserved	

IIR Channel Control Register (IIRCTL2)

The IIRCTL2 register, shown in Figure A-43 and described in Table A-51, is used to configure the channel specific parameters. These include number of biquads and window size.

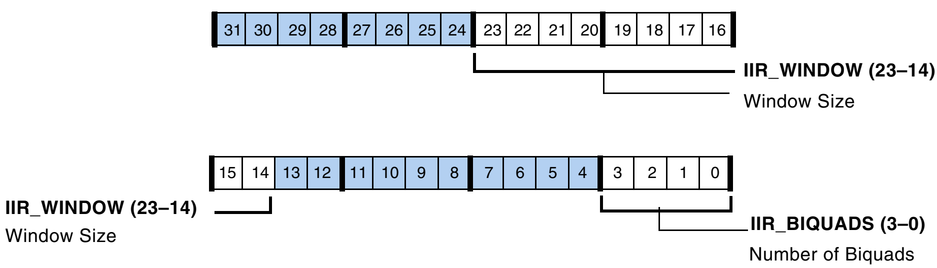

Figure A-43. IIRCTL2 Register

Table A-51. IIRCTL2 Register Bit Descriptions (RW)

Bits	Name	Description
3–0	IIR_BIQUADS	**Number of Biquads.** Programmable between 0–11. Number of Biquads = BIQUADS + 1
13–4	Reserved	
23–14	IIR_WINDOW	**Window Size Parameter.** Window size specifies the number of sample/block to process (sample based processing = window size of 1)
31–24	Reserved	

IIR MAC Status Register (IIRMACSTAT)

The IIRMACSTAT register, shown in Figure A-44 and described in Table A-52, provides the status of MAC operations.

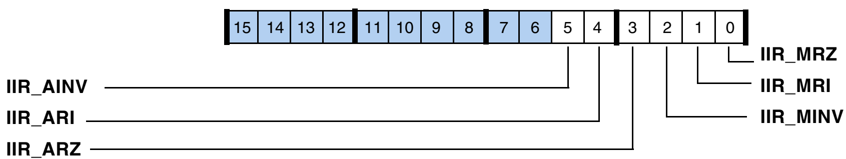

Figure A-44. IIRMACSTAT Register

Table A-52. IIRMACSTAT Register Bit Descriptions (RO)

Bits	Name	Description
0	IIR_MRZ	**Multiplier Result Zero.** Set if multiplier results is zero.
1	IIR_MRI	**Multiplier Result Infinity.** Set if multiplier results is infinity.
2	IIR_MINV	**Multiply Invalid.** Set if multiply operation is invalid.
3	IIR_ARZ	**Adder Result Zero.** Set if adder results is zero.
4	IIR_ARI	**Adder Result Infinity.** Set if adder results is infinity.
5	IIR_AINV	**Addition Invalid.** Set if addition is invalid.
31–6	Reserved	

IIR DMA Status Register (IIRDMASTAT)

The IIR DMA registers are described in "Data Transfer" on page 7-14. The IIRDMASTAT register, shown in Figure A-45 and described in Table A-53, provides the status of DMA operations. All the bits in this register are read only.

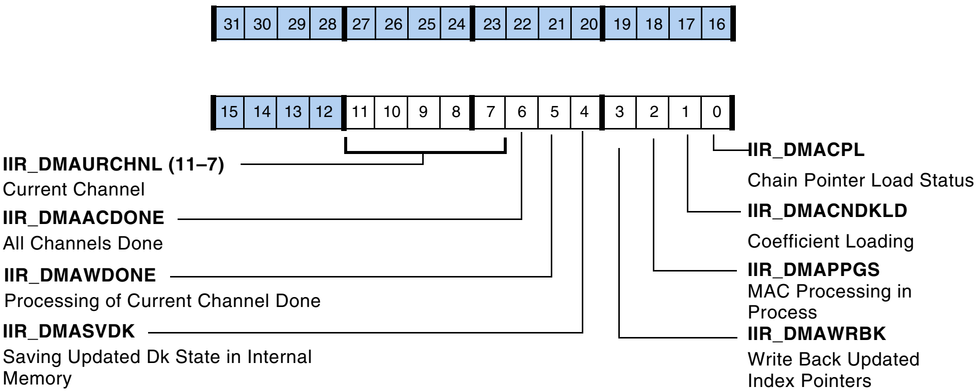

Figure A-45. IIRDMASTAT Register

Table A-53. IIRDMASTAT Register Bit Descriptions (RO)

Bits	Name	Description
0	IIR_DMACPL	**Chain Pointer Loading Status.** 1 = state machine in chain pointer load state
1	IIR_DMACnDkLD	**Coefficient and Dk Loading.**
2	IIR_DMAPPGS	**MAC Processing In Progress.**
3	IIR_DMAWRBK	**Writing Back Updated Index Registers.**

Table A-53. IIRDMASTAT Register Bit Descriptions (RO) (Cont'd)

Bits	Name	Description
4	IIR_DMASVDk	**Saving Updated Dk State in Internal Memory.** If there is more than one channel (IIR_NCH>0), IIR_DMASVDk toggles between 0 and 1 as it starts and completes the save state operation on one channel at a time. Therefore, this bit is not a reliable indicator of completion of the save state operation for all channels. To ensure graceful completion of the save state operation, you must poll both IIR_DMACPL and IIR_DMASVDk and ensure (IIR_DMACPL OR IIR_DMASVDk) = 0 after IIR_DMAACDONE is set. The recommended method for minimizing core intervention is to configure the accelerator to generate an interrupt when the processing of all the channels is complete (IIR_CCINTR bit of IIRCTL1 register is set), then poll to ensure (IIR_DMACPL OR IIR_DMASVDk) = 0 inside the interrupt service routine. To minimize the interrupt service time, the core can perform unrelated tasks before it starts polling for save state operation completion.
5 (ROC)	IIR_DMAWDONE	**Processing of Current Channel Done.** Sticky, cleared on register read. The IIR_CCINTR bit will not affect the IIR_DMAWDONE Status Bit
6 (ROC)	IIR_DMAACDONE	**All Channels Done.** Sticky, cleared on register read. The IIR_CCINTR bit will not affect the IIR_DMAACDONE Status Bit
11–7	IIR_DMACURCHNL	**Current Channel.** Channel that is being processed in the TDM slot. Zero indicates the last slot.
31–12	Reserved	

IIR Debug Registers (IIRDEBUGCTL, IIRDEBUGADDR)

The IIRDEBUGCTL register, shown in Figure A-46 and described in Table A-54, controls the debug mode operation of the IIR accelerator. Note that these registers should only be used in debug mode.

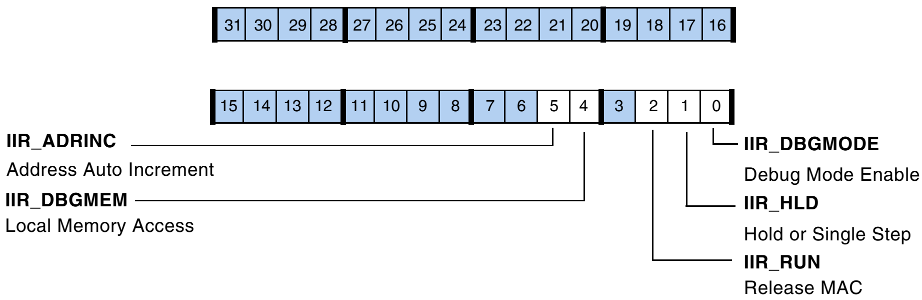

Figure A-46. IIRDEBUGCTL Register

Table A-54. IIRDEBUGCTL Register Bit Descriptions (RW)

Bits	Name	Description
0	IIR_DBGMODE	**Debug Mode Enable.** 0 = Disable 1 = Enable For local memory access, the IIRCTL1 register can be cleared.
1	IIR_HLD	**Hold or Single Step.** The function of this bit is based on the IIR_DBGMEM bit setting. For IIR_DBGMEM = 0: 1 = Single step For IIR_DBGMEM = 1: 1 = Hold data
2 (RW1S)	IIR_RUN	**Release the MAC.** This bit is self clearing after one IIR clock cycle.
3	Reserved	
4	IIR_DBGMEM	**Local Memory Access.** If set, the data and coefficients memory can be indirectly accessed.
5	IIR_ADRINC	**Address Auto Increment.** If this bit is set, the address register auto increments on IIRDBGWRDATA_H/ IIRDBGWRDATA_L writes and IIRDBGRDDATA_H/ IIRDBGRDDATA_L reads.
31–6	Reserved	

Media Local Bus Registers

The registers in this section are used to program and get status information for the MLB interface and the specific channels used.

MLB System Registers

This section lists all different control and status registers related to the MLB controller.

Device Control Configuration Register (MLB_DCCR)

This register, shown in Figure A-47 and described in Table A-55, controls the device enable/disable, clock rate, lock status and addressing.

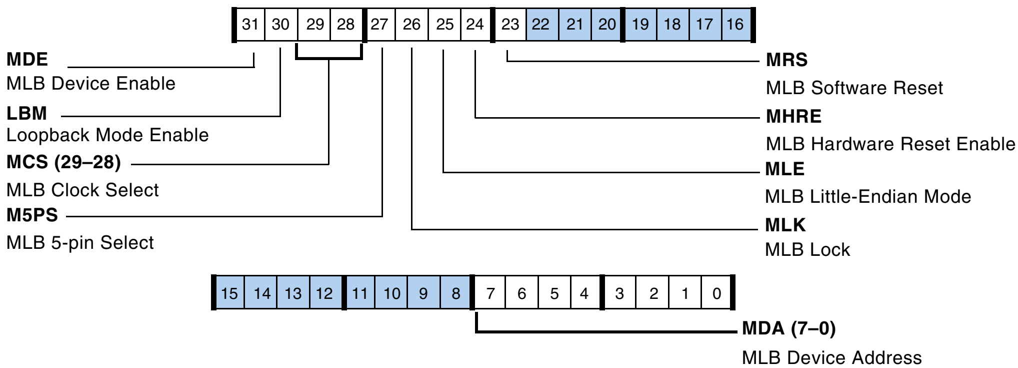

Figure A-47. MLB_DCCR Register

Table A-55. MLB_DCCR Register Bit Descriptions (RW)

Bit	Name	Description
7–0	MDA	**MLB Device Address.** Determines the unique device address (DA) for ADSP-214xx MediaLB device. MLB device address is 16 bits. Bits 15–9 and LSB are always zero. Only bits 8–1 vary and they are defined by MLB_DCCR bits 7–0. Device addresses are used by the system channel MLBSCAN command. MDA Device Address 00000001 0000 000'0 0000 001'0 = 0x0002 00000010 0000 000'0 0000 010'0 = 0x0004 00000011 0000 000'0 0000 100'0 = 0x0006 ---- 11111110 0000 000'1 1111 110'0 = 0x01FC For further information on assigning the device address, refer to the MLB specification.
22–8	Reserved	
23 (RW1S)	MRS	**MLB Software Reset.** When set, resets the MLB physical and link layer logic. Hardware clears this bit automatically.
24	MHRE	**MLB Hardware Reset Enable.** Enables hardware to automatically reset the MLB physical and link layer logic upon the reception MLB reset system command. 0 = Hardware reset option disabled 1 = MLB reset causes hardware reset
25	MLE	**MLB Little-Endian Mode.** 0 = Big-Endian mode 1 = Little-Endian mode
26 (RO)	MLK	**MLB Lock.** When set, indicates that MLB port is synchronized to the incoming MLB frame. If MLK is clear (unlocked), it is set after FRAMESYNC is detected at the same position for three consecutive frames. If MLK is set (locked), it is cleared after not receiving FRAMESYNC at the expected time for two consecutive frames. While MLK is set, FRAMESYNC patterns occurring at locations other than the expected one are ignored.
27	M5PS	**MLB 5-Pin Select.** 0 = 3-pin MLB mode 1 = 5-pin MLB mode (legacy protocol to emulate MediaLB in software)

Table A-55. MLB_DCCR Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
29–28	MCS	**MLB Clock Select.** 00 = 256Fs – supports 8 quadlets per frame 01 = 512Fs – supports 16 quadlets per frame 10 = 1024Fs – supports 32 quadlets per frame 11 = reserved
30	LBM	**Loopback Mode Enable.** 0 = Loopback disabled 1 = Loopback enabled
31	MDE	**MLB Device Enable.** 0 = MLB disabled 1 = MLB enabled–The MLB local channel buffer is immediately flushed by a HW or SW reset only

System Status Register (MLB_SSCR)

This register, shown in Figure A-48 and described in Table A-56, allows system software to monitor and control the status of the MLB network. The register is updated once per frame by hardware during the MLB system channel. The bits of this register are not valid until the ADSP-214xx is locked to the MLB interface (except for the bits associated with MLB lock and unlock, `SDMU` and `SDML`). System software must service events before the start of the next MLB frame to prevent the current frame status from being lost.

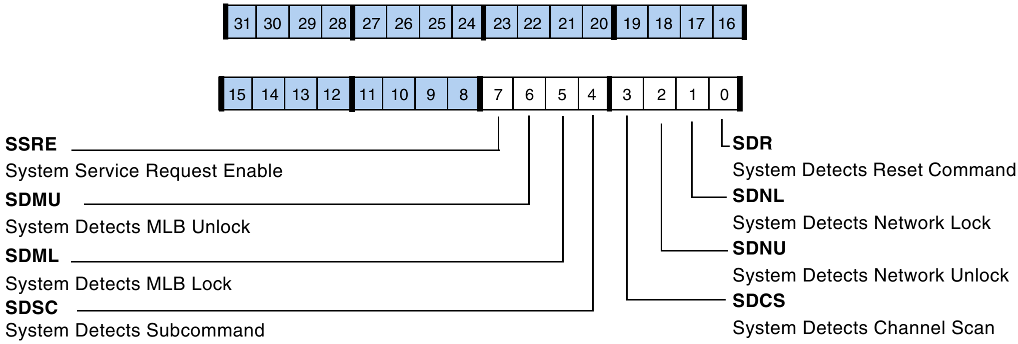

Figure A-48. MLB_SSCR Register

Table A-56. MLB_SSCR Register Bit Descriptions (RW1C)

Bit	Name	Description
0	SDR	**System Detects Reset Command.** When set, indicates MLB device has received MLBReset system command.
1	SDNL	**System Detects Network Lock.** When set, indicates the MLB device has received Most lock system command.
2	SDNU	**System Detects Network Unlock.** When set, indicates the MLB device has received MOST unlock system command.
3	SDCS	**System Detects Channel Scan.** When set, indicates the MLB device has received the MLBScan system command. The device address is stored in the SDCR register.
4	SDSC	**System Detects Subcommand.** When set, indicates the MLB device has received the MLBSubCmd system command. The device command is stored in the SDCR register and decoding is performed by software.
5	SDML	**System Detects MLB Lock.** When set, indicates the MLB device has locked onto the MLB frame.
6	SDMU	**System Detects MLB Unlock.**When set, indicates the MLB device has unlocked from the MLB frame.
7 (RW1S)	SSRE	**System Service Request Enable.** When set, indicates that the MLB device needs service. Write 1 to set.
31–8	Reserved	

System Data Configuration Register (MLB_SDCR)

This register, described in Table A-57, allows system software to receive control information from the MLB controller. The MLB_SDCR register is updated once per frame by the hardware during the MLB system channel. This register is loaded with the data from the MLBDAT_IN signal during the system channel quadlet. System software must read the MLB_SDCR register before the start of the next MLB frame to prevent the current data from being lost.

Table A-57. MLB_SDCR Register Description (RO)

Bit	Name	Description
31–0	SDATA	**System Channel Data.**

System Mask Configuration Register (MLB_SMCR)

This register, described in Table A-58, allows system software to mask system status interrupts. When a mask bit is set, the corresponding system channel interrupt is masked.

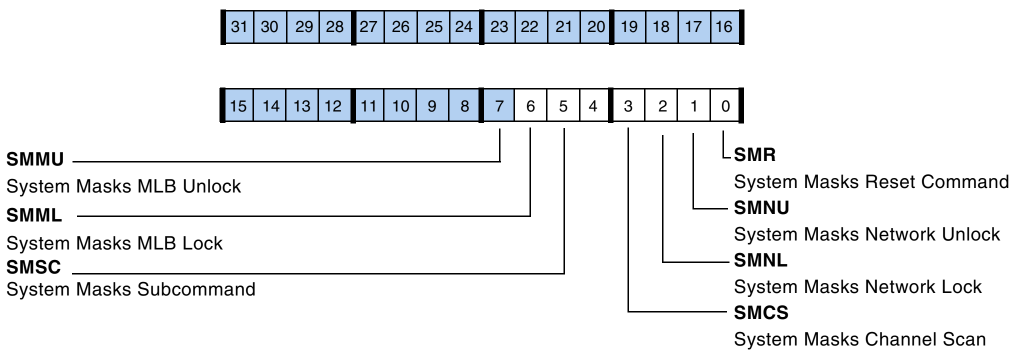

Figure A-49. MLB_SMCR Register

Table A-58. MLB_SMCR Register Bit Descriptions (RW)

Bit	Name	Description
0	SMR	**System Masks Reset Command.** When set, this bit masks system interrupts for Mlb Reset system command.
2	SMNU	**System Masks Network Unlock.** When set, this bit masks system interrupts for the MOST_unlock system command.
1	SMNL	**System Masks Network Lock.** When set, this bit masks system interrupts for the MOST_Lock system command.
3	SMCS	**System Masks Channel Scan.** When set, this bit masks system interrupts for MlbScan system command.
4	SMSC	**System Masks Subcommand.** When set, this bit masks system interrupts for MlbSubCmd (0xE6) system command.
5	SMML	**System Masks MLB Lock.** When set, this bit masks system interrupts generated when MLB lock is detected. At reset, MLB lock events are masked, (SMML = 1).
6	SMMU	**System Masks MLB Unlock.** When set, this bit masks system interrupts generated when a MediaLB unlock is detected. At reset, MediaLB unlock events are masked (SMMU = 1).
31–7	Reserved	

Channel Interrupt Status Register (MLB_CICR)

The channel interrupt status register reflects the channel interrupt status of the individual logical channels. The channel status update (`CSU`) bits are set by hardware when a channel interrupt is generated. The `CSU` bits are sticky and can only be reset by software. To clear a particular bit in this register, software must clear all of the unmasked status bits in the corresponding `MLB_CSCRx` registers.

Table A-59. MLB_CICR Register Description (RO)

Bit	Name	Description
30–0	CSU	**Channel Status Update.**
31	Reserved	

DMA Base Address Registers

The DMA address is constituted by a 5-bit base in the MLB base registers (for the corresponding channel mode select) and a 14-bit offset configured using the BCA bits in the MLB_CCBCRx register.

Synchronous Base Address Register (MLB_SBCR)

The MLB_SBCR, described in Table A-60, holds the base address of the system memory buffers of all synchronous channels in the device.

Table A-60. MLB_SBCR Register Bit Descriptions (RW)

Bit	Name	Description
4–0	STBA	Synchronous transmit base address for DMA mode
15–5	Reserved	
20–16	SRBA	Synchronous receive base address for DMA mode
31–21	Reserved	

Asynchronous Base Address Register (MLB_ABCR)

The MLB_ABCR register, described in Table A-61, holds the base address of the system memory buffers of all asynchronous channels in the device.

Table A-61. MLB_ABCR Register Bit Descriptions (RW)

Bit	Name	Description
4–0	ATBA	Asynchronous transmit base address for DMA mode
15–5	Reserved	
20–16	ARBA	Asynchronous receive base address for DMA mode
31–21	Reserved	

Control Base Address Register (MLB_CBCR)

The MLB_CBCR register, described in Table A-62, hold the base address of the system memory buffers of all control channels in the device.

Table A-62. MLB_CBCR Register Bit Descriptions (RW)

Bit	Name	Description
4–0	CTBA	Control transmit base address for DMA mode
15–5	Reserved	
20–16	CRBA	Control receive base address for DMA mode
31–21	Reserved	

Logical Channel Registers

The MLB controller supports up to 31 logical channels. Therefore the variable in the register names is valid for x = 0–30. This section lists all different control and status registers related to the logical channels.

Channel Control Registers (MLB_CECRx)

These registers define the basic attributes of a given logical channel, such as the channel enable, channel direction, and channel address. Note the definition of the bit fields is dependent on the CTYPE bit field. If the selection is synchronous channels (default) than the Figure A-51 and Table A-64 are valid. If the CTYPE bits are asynchronous/control then the Figure A-50 and Table A-63 may apply.

Figure A-50 and Table A-63 provide information for asynchronous and control channels and Figure A-51 and Table A-64 provide information for synchronous channels.

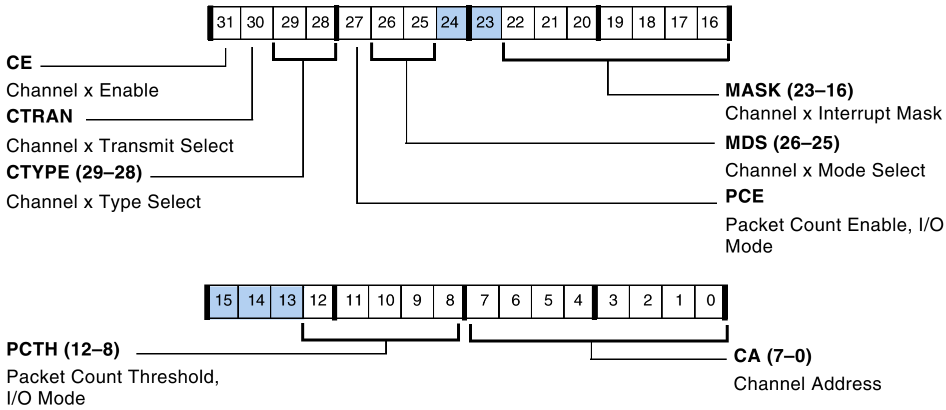

Figure A-50. MLB_CECRx Register (Asynchronous and Control Channels)

Table A-63. MLB_CECRx Register Bit Descriptions for Asynchronous and Control Channels (RW)

Bit	Name	Description
7–0	CA	**Channel Address.** These bits determine the channel address associated with this logical channel. MLB channel address is 16 bits; bits 15–9 and LSB are always zero. Only bits 8–1 vary and they are defined by MLB_CECRx bits 7–0. **Channel** **Address** 00000001 0x0002 00000010 0x0004 00000011 0x0006 00000100 0x0008 .... 11111111 0x01FE For further information on assigning the device address, refer to the MLB specification.
12–8	PCTH	**Packet Count Threshold, I/O Mode.** Software can program this field with the number of packets to receive before generating an Rx packet-count service request. This service request is generated independent of, and in addition to, other service requests generated via the standard buffer threshold mechanism. In DMA mode these bits are reserved.

Table A-63. MLB_CECRx Register Bit Descriptions for Asynchronous and Control Channels (RW) (Cont'd)

Bit	Name	Description
15–13	Reserved	
16	MASK0	**Mask Protocol Error.** When set, masks protocol error channel interrupts for this logical channel. This bit valid for all Rx channel types. This is valid for asynchronous and control Tx channels only.
17	MASK1	**Mask Detect Break.** When set, masks detect break channel interrupt for this logical channel. This bit is valid for asynchronous and control channels only.
18	MASK2 (I/O) MASK2 (DMA)	**Masks Receive Service Request.** When set, masks Rx channel service request interrupts for this logical channel. **Mask Buffer Done.** When set, masks buffer done channel interrupts for this logical channel.
19	MASK3 (I/O) MASK3 (DMA)	**Masks Transmit Service Request.** When set, masks Tx channel service request interrupts for this logical channel. **Mask Buffer Start.** When set, masks buffer start channel interrupts for this logical channel.
20	MASK4	**Mask Buffer Error.** When set, masks buffer error channel interrupts for this logical channel.
21	Reserved	
22	MASK6	**Mask Lost Frame Synchronization.** When set, masks lost frame synchronization channel interrupts for this logical channel.
24–23	Reserved	
26–25	MDS	**Channel x Mode Select.** 00 = Ping-pong DMA mode (default) 01 = reserved (only valid for synchronous channels) 10 = I/O mode enable 11 = Reserved
27	PCE	**Packet Count Enable.** Enable the Rx packet counter. This bit is valid for asynchronous and control Rx channels in I/O mode. 0 = Disable 1 = Enable

Table A-63. MLB_CECRx Register Bit Descriptions for Asynchronous and Control Channels (RW) (Cont'd)

Bit	Name	Description
29–28	CTYPE	**Channel x Type Select.** 00 = Synchronous (default) 01 = Reserved 10 = Asynchronous 11 = Control
30	CTRAN	**Channel x Transmit Select.** 0 = Receive (default) 1 = Transmit
31	CE	**Channel x Enable.** 0 = Channel n disabled (default) 1 = Enabled

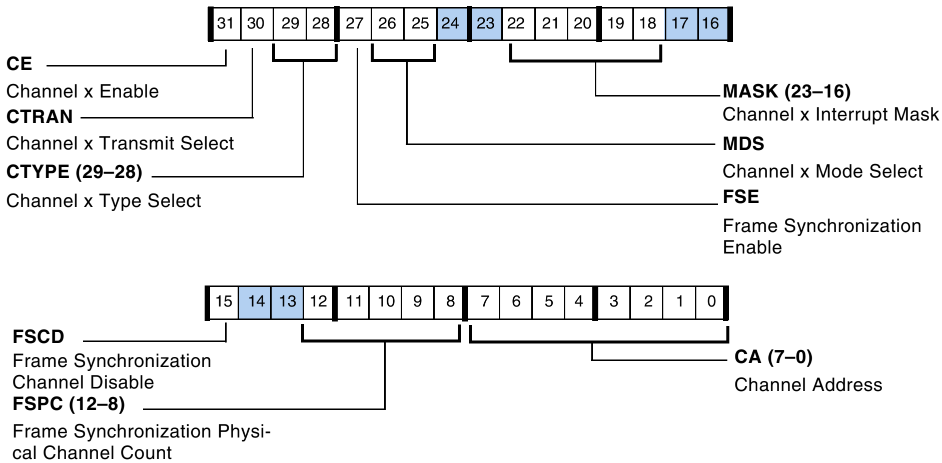

Figure A-51. MLB_CECRx Register (Synchronous Channels)

Table A-64. MLB_CECRx Register Bit Descriptions for Synchronous Channels (RW)

Bit	Name	Description
7–0	CA	**Channel Address.** These bits determine the channel address associated with this logical channel. MLB channel address is 16 bits; bits 15–9 and LSB are always zero. Only bits 8–1 vary and they are defined by MLB_CECRx bits 7–0. 00000001 0x0002 00000010 0x0004 00000011 0x0006 00000100 0x0008 .... 11111111 0x01FE
12–8	FSPC	**Frame Synchronization Physical Channel Count.** Defines the number of physical channels (quadlets) expected to match this logical channel's channel address each MLB frame.
14–13	Reserved	
15	FSCD	**Frame Synchronization Channel Disable.** When set, disables this logical channel when frame synchronization is lost.
16–17	Reserved	
18	MASK2 (I/O) MASK2 (DMA)	**Masks Receive Service Request.** When set, masks Rx channel service request interrupts for this logical channel. **Mask Buffer Done.** When set, masks buffer done channel interrupts for this logical channel.
19	MASK3 (I/O) MASK3 (DMA)	**Masks Transmit Service Request.** When set, masks Tx channel service request interrupts for this logical channel. **Mask Buffer Start.** When set, masks buffer start channel interrupts for this logical channel.
20	MASK4	**Mask Buffer Error.** When set, masks buffer error channel interrupts for this logical channel.
21	MASK5	Reserved
22	MASK6	**Mask Lost Frame Synchronization.** When set, masks lost frame synchronization channel interrupts for this logical channel.
23	Reserved	
24	Reserved	

Table A-64. MLB_CECRx Register Bit Descriptions for Synchronous Channels (RW)

Bit	Name	Description
26–25	MDS	**Channel x Mode Select.** 00 = Ping-pong DMA mode (default) 01 = Circular buffering DMA 10 = I/O mode enable 11 = Reserved
27	FSE	**Frame Synchronization Enable.** When set, enables streaming channel frame synchronization for this logical synchronous channel.
29–28	CTYPE	**Channel x Type Select.** 00 = Synchronous (default) 01 = Reserved 10 = Asynchronous 11 = Control
30	CTRAN	**Channel x Transmit Select.** 0 = Receive (default) 1 = Transmit
31	CE	**Channel x Enable.** 0 = Channel x disabled (default) 1 = Enabled

Channel Status Configuration Registers (MLB_CSCRx)

This register, shown in Figure A-52 and described in Table A-65, shows the status of the current and previous buffer for the logical channel. For all bits a 1 means the condition exists. Setting any of the `STS11-0` bits clears the interrupt acknowledge for the corresponding interrupt channel.

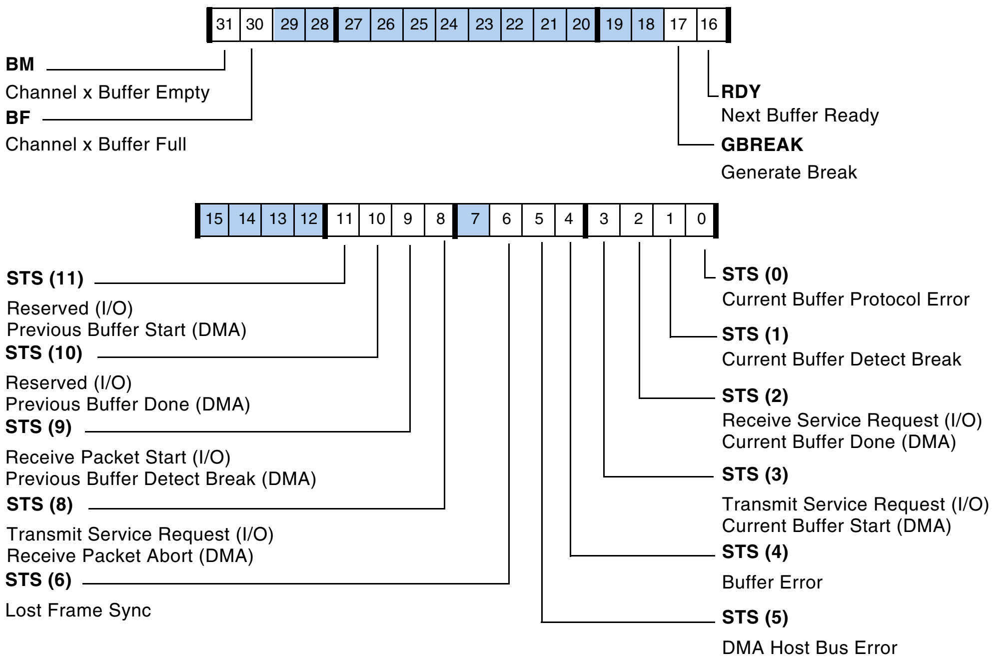

Figure A-52. MLB_CECRx Register

Table A-65. MLB_CSCRx Register Description (RW1C)

Bit	Name	Description
0	STS0	**Current Buffer Protocol Error.** Indicates that either a TX channel has detected an RxStatus of ReceiverProtocolError (72h), an RX channel has detected an invalid command for a given channel type, or an additional ControlStart (30h) or AsyncStart (20h) command has been received while in the middle of a packet. This bit is valid for all RX channel types and valid for only asynchronous and control TX channels.
1	STS1	**Current Buffer Detect Break.** Indicates that either a TX channel has detected a receiver break response, ReceiverBreak (70h), or an RX channel has detected a transmitter break command, ControlBreak (36h) or Async-Break (26h), while processing the Current Buffer. This bit is valid for asynchronous and control channels only.

Table A-65. MLB_CSCRx Register Description (RW1C) (Cont'd)

Bit	Name	Description
2	STS2 (I/O) STS2 (DMA)	**Receive Service Request (I/O).** Indicates that an RX channel is requesting service from system software. Receive service requests are issued if the number of free quadlets in the local channel buffer is less than or equal to LCBCRn.TH[9:0]. This bit is valid for all channel types. **Current Buffer Done (DMA).** Indicates that the last quadlet from the last packet (in the Current Buffer) has been successfully transmitted or received. This bit is valid for all channel types.
3	STS3 (I/O) STS3 (DMA)	**Transmit Service Request.** Indicates that a TX channel is requesting service from system software. Transmit service requests are issued if the number of valid quadlets in the local channel buffer is less than or equal to LCBCRn.TH[9:0]. This bit is valid for all channel types. **Current Buffer Start.** Indicates that the DMA controller has started processing the Current Buffer. This bit is set after the contents of CNBCRn have been loaded into CCBCRn, the CSCRn.RDY bit has been cleared (for ping-pong buffering), and hardware is available to accept the next buffer. This bit is valid for all channel types.
4	STS4	**Buffer Error.** Indicates that either a TX channel has detected a buffer underflow (attempt to pop data from an empty buffer), or an RX channel has detected a buffer overflow (attempt to push data onto a full buffer). This bit is valid for synchronous RX/TX (CECRn.FCE = 0) channels only.
5	STS5 (DMA)	**DMA Host Bus Error.** Indicates that a DMA bus error has been detected.
6	STS6	**Lost Frame Sync.** Indicates that the logical channel has lost synchronization with the MediaLB frame. This bit is valid for synchronous channels only.
7	Reserved	

Table A-65. MLB_CSCRx Register Description (RW1C) (Cont'd)

Bit	Name	Description
8	STS8 (I/O) STS8 (DMA)	**Receive Packet Abort.** Indicates that an RX channel has detected an aborted packet. Received packets are aborted if the receiver generates a break response, ReceiverBreak (70h), or detects a transmitter packet break command; ControlBreak (36h) or AsyncBreak (26h). This bit can also indicate the RX channel has detected a transmit command protocol error. This interrupt can be used by system software to detect when it has encountered the beginning of an aborted packet. This bit is valid for asynchronous and control RX channels only. **Previous Buffer Protocol Error.** Indicates that either a TX channel has detected an RxStatus of ReceiverProtocolError (72h), a RX channel has detected an invalid command for this channel type, or an additional AsyncStart (20h) or ControlStart (30h) command has been received while in the middle of a packet. This bit is valid for all RX channels and valid for only asynchronous and control TX channels.
9	STS9 (I/O) STS9 (DMA)	**Receive Packet Start.** Indicates that an RX channel has detected a transmitter packet start command; ControlStart (30h) or AsyncStart (20h). This status bit can be used by system software to detect when it has reached the end of an aborted packet. This bit is valid for asynchronous and control RX channels only. **Previous Buffer Detect Break.** Indicates that either a TX channel has detected a receiver break response, ReceiverBreak (70h), or an RX channel has detected a transmitter break command, ControlBreak (36h) or AsyncBreak (26h), while processing the Previous Buffer. This bit is valid for all channel types.
10	STS10 (DMA)	**Previous Buffer Done.** Indicates that the last quadlet of the Previous Buffer has been successfully transmitted or received. This bit is valid for all channel types. Reserved in I/O mode.
11	STS11 (DMA)	**Previous Buffer Start.** Indicates the first quadlet of the Previous Buffer has been successfully transmitted or received. This bit is valid for all channel types. Reserved in I/O mode.
15–12	Reserved	
16	RDY	**Next Buffer Ready (DMA Mode).** This bit is reserved for I/O mode. 0 = Next buffer ready for ping-pong DMA. Hardware clears this bit after the buffer begins to be processed. 1 = Next buffer ready for circular buffer DMA. Software should clear this RW1C bit only when buffer processing needs to be stopped.

Table A-65. MLB_CSCRx Register Description (RW1C) (Cont'd)

Bit	Name	Description
17 (RW)	GBREAK	**Generate Break.** 0 = No break 1 = Generate break
29–18	Reserved	
30 (RO)	BF	**Channel x Buffer Full.** When set, buffer is full.
31 (RO)	BM	**Channel x Buffer Empty.** When set, buffer is empty.

Channel x Current Buffer Configuration Registers (MLB_CCBCRx)

The registers, described in Table A-66, allow software to monitor the address pointer and buffer length of the current DMA buffer in internal memory when the logical channel is configured in DMA mode. When configured in I/O mode, this register implements the Rx data buffer. The definition of the bit fields in this register vary depending on the selected channel type.

The 5-bit offset of the DMA address comes from the base address registers.

Table A-66. MLB_CCBCRx Register Bit Descriptions (RO)

Bit	Name	Description
1–0		Reserved for other channel types
15–2	BFA	**Buffer Final Address.**
17–16		Reserved for other channel types
31–18	BCA	**Buffer Current Address.**

Channel x Next Buffer Configuration Registers (MLB_CNBCRx)

These registers, described in Table A-67, allows system software to set the start and end address of the next buffer in internal memory for the logical channel in DMA mode. When configured in I/O mode, these registers implement the Tx data buffer. The definition of bit fields in this register vary dependent on the selected channel type.

Table A-67. MLB_CNBCRx Register Description (RW)

Bit	Name	Description
1–0		Reserved for other channel types
15–2	BEA	**Next Buffer End Address.**
17–16	Reserved	
31–18	BSA	**Next Buffer Start Address.**

Local Buffer Configuration Registers (MLB_LCBCRx)

These registers, described in Table A-68, allow software to optimize the use of the local channel buffer memory. These registers should only be written by software while the logical channel is disabled. The size of the local channel buffer RAM is 124 words. At reset, this RAM is shared equally by all 31 channels with four words for each channel. The buffer depth can be up to 124 words (quadlets), when only one channel is used.

Table A-68. MLB_LCBCRx Register Description (RW)

Bit	Name	Description
12–0	SA	**Buffer Start Address.** Determines the starting address (in quadlets/4) of the channel buffer for the logical channel x. Reset value = {120, 116, 112 …4, 0} for x = 30:0 0x0000 = start address offset of 0 words 0x0001 = start address offset of 4 words 0x0002 = start address offset of 8 words … 0x001E = start address offset of 120 words 0x001F to 0x1FFF = Reserved
21–13	BD	**Buffer Depth.** Defines the depth (in quadlets/4–1) of the local channel buffer for the logical channel x. 0x000 = depth = 4 words 0x001 = depth = 8 words 0x002 = depth = 12 words … 0x01E = depth = 124 words 0x01F to 0x1FF = Reserved Reset value = 4 for all x=0:30
31–22	TH	**Buffer Threshold.** Defines the threshold (in quadlets/2) of the local channel buffer for the logical channel x in I/O mode. Hardware uses this threshold value to determine when to issue an I/O service request to system software. Reset value = 2 for all x = 0:30 0x000 = threshold = 0 word 0x001 = threshold = 2 words 0x002 = threshold = 4 words …. 0x03D = threshold = 122 words 0x03E to 0x3FF = Reserved Reserved in DMA mode

Watchdog Timer Registers

The following sections provide bit descriptions for the registers associated with the watchdog timer.

Control (WDTCTL)

The watchdog control register (WDTCTL), is a 32-bit system memory-mapped register used to configure the watchdog timer. A write to the WDT_EN bit enables the watchdog timer. This register is protected against accidental writes from the processor core by the watchdog unlock register (WDTUNLOCK). Attempts by the core to write to WDTCTL without an unlock command causes the WDT to expire, and reset the system. This condition is captured in the watchdog exception field (WDT_ERR bit in the WDTSTATUS register).

Writes made by software to this register keep it enabled. Only an External hardware reset can clear WDTCTL. Reads from this register when WDT is disabled return 0x0, and when WDT is enabled, always return 0x1.

Status (WDTSTATUS)

The WDTSTATUS register, shown in Figure A-53 and described in Table A-69, contains the watchdog timer status information. This register is not cleared by the WDT generated reset.

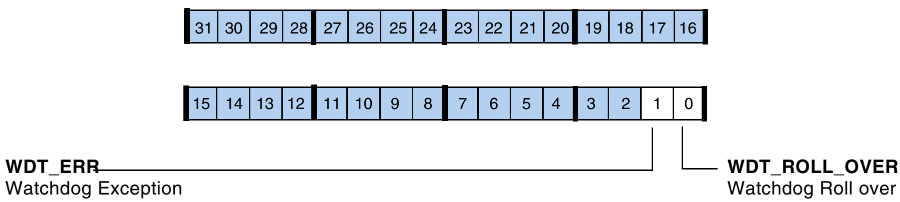

Figure A-53. WDTSTATUS Register

Table A-69. WDTSTATUS Register Bit Descriptions (RW1C)

Bit	Name	Description
0	WDT_ROLL_OVER	**Watchdog Roll Over.** Indicates that DSP core attempted to write to WDT configuration space without an unlock "command". Bit is set when the above exception occurs. Software can determine whether the watchdog has expired by interrogating this bit. This is a sticky bit that is set whenever the watchdog timer count rolls over.
1	WDT_ERR	**Watchdog Error.** Indicates that watchdog timer has expired. Bit is set when counter expires. Attempts by the core to write to the WDT configuration space without an unlock command, causes the WDT to expire and this condition is captured in the watchdog exception field. This is a sticky bit that is set whenever the above exception occurs.

Current Count (WDTCURCNT)

The `WDTCURCNT` register contains the current count value of the watchdog timer. Reads to `WDTCURCNT` return the current count value. For added safety, this register can only be updated when WDT configuration space is unlocked by programming the command in the `WDTUNLOCK` register. Values cannot be stored directly in `WDTCURCNT`, but are instead copied from `WDTCNT`.

Enabling the watchdog timer does not automatically reload `WDTCURCNT` from `WDTCNT`. The `WDTCURCNT` register is a 32-bit unsigned system memory-mapped register that must be accessed with 32-bit reads and writes.

Trip Counter (WDTTRIP)

The WDT contains a software programmable register `WDTTRIP` that sets the number of times that the WDT can expire before the $\overline{\texttt{WDTRSTO}}$ pin is continually asserted until the next time hardware reset is applied. This register is unaffected by WDT generated reset. This register can only be updated when the WDT is disabled and WDT configuration space is unlocked by programming the command in the WDTUNLOCK register.

Writes to the WDTCURCNT register have a maximum latency of 2.5 WDTCLK cycles.

Table A-70. WDTTRIP Register Bit Descriptions (RW)

Bit	Name	Description
3–0	TRIPVAL	**Current Value of Trip Counter.** This is the trip counter value, programmable from 0 to 15. The number of times WDT can expire is programmable by the TRIPVAL field Reading this register also gives the current value of the trip counter.
7–4 (RO)	CURTRIPVAL	**Current Number of WDT Resets.** Reports all current WDT generated resets ($\overline{\text{WDTRSTO}}$ asserted).

Clock Select (WDTCLKSEL)

This register, described in Table A-71, can only be updated when the WDT is disabled and WDT configuration space is unlocked by programming the command in the WDTUNLOCK register. Writes to the WDTCLKSEL register are ignored after the WDT is enabled.

Note that his register is reset on external hardware reset only. This ensures that the selected clock source remains the same even after a WDT generated reset is asserted.

Table A-71. WDTCLKSEL Register Bit Descriptions (RW)

Bit	Name	Description
0	WDT_CLK_INT_OSC	**Clock Select.** When this bit = 0, the WDTCLK source can be an external clock applied to the WDT_CLKIN pin or an external ceramic oscillator connected to the WDT_CLKIN and WDT_CLKO pins. 0 = Selects ceramic oscillator output or external clock 1 = Selects internal RC oscillator output
1	WDT_OSCPWRDWN	**Internal RC Oscillator Power Down.** 0 = Oscillator is powered up 1 = Oscillator is powered down
2	WDT_OSCNONRST	**Internal RC Oscillator Reset.** 0 = Oscillator is reset 1 = Oscillator out of reset

Period (WDTCNT)

The `WDTCNT` register holds the 32-bit unsigned count value. The `WDTCNT` register must always be accessed with 32-bit read/writes.

The watchdog count register holds the programmable count value. A valid write to the watchdog count register also pre loads the watchdog current counter. For added safety, the watchdog count register can only be updated when the WDT is disabled and WDT configuration space is unlocked by programming the command in the `WDTUNLOCK` register.

Unlock (WDTUNLOCK)

The `WDTUNLOCK` register protects the WDT configuration space against accidental writes from the processor core. Before attempting to write to the WDT configuration space, the core must unlock the WDT by writing the command value (0xAD21AD21) to this register. Attempts by the core to write to WDT configuration space without this command causes the WDT to expire. This exception is captured in the WDTSTATUS register. After configuring the WDT configuration space, the core needs to lock it again by writing any value other than the command value to the register.

Real-Time Clock Registers

The following sections describe the registers associated with the real-time clock (RTC).

Control Register (RTC_CTL)

This register, shown in Figure A-54 and described in Table A-72 control interrupt generation for the RTC.

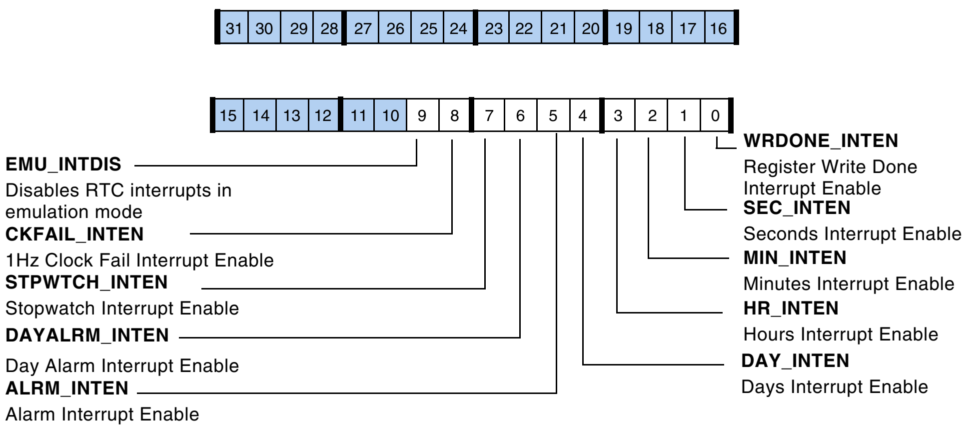

Figure A-54. RTC_CTL Register

Table A-72. RTC_CTL Register Bit Descriptions (RW)

Bit	Name	Description
0	WRDONE_INTEN	**Register Write Done Interrupt Enable.** 0 = Register Write Done interrupt disabled 1 = Register Write Done interrupt enabled
1	SEC_INTEN	**Seconds Interrupt Enable.** 0 = Seconds interrupt disabled 1 = Seconds interrupt enabled
2	MIN_INTEN	**Minutes Interrupt Enable.** 0 = Minutes interrupt disabled 1 = Minutes interrupt enabled
3	HR_INTEN	**Hours Interrupt Enable.** 0 = Hours interrupt disabled 1 = Hours interrupt enabled
4	DAY_INTEN	**Days Interrupt Enable.** 0 = Days interrupt disabled 1 = Days interrupt enabled
5	ALRM_INTEN	**Alarm Interrupt Enable.** 0 = Alarm interrupt disabled 1 = Alarm interrupt enabled
6	DAYALRM_INTEN	**Day Alarm Interrupt Enable.** 0 = Day alarm interrupt disabled 1 = Day alarm interrupt enabled
7	STPWTCH_INTEN	**Stopwatch Interrupt Enable.** 0 = Stopwatch interrupt disabled 1 = Stopwatch interrupt enabled
8	CKFAIL_INTEN	**RTC 1Hz Clock Fail Interrupt Enable.** Indicates that the oscillator failed to start. 0 = RTC 1Hz clock fail interrupt disabled 1 = RTC 1Hz clock fail interrupt enabled
9	EMU_INTDIS	**Disables/Enables RTC Interrupts in Emulation Mode.** 0 = RTC interrupts enabled (if the individual interrupt enable bit is set) in emulation mode 1 = RTC interrupts disabled in emulation mode
31–10	Reserved	

Status Register (RTC_STAT)

This register, shown in Figure A-55, The RTC Status register contains the RTC event flags and RTC interrupt status. These bits are sticky. Once set by the event, each bit remains set until cleared by a software read of this register. These sticky bits are independent of the interrupt enable bits in RTC_CTL register. Values are cleared by reading RTC_STAT register, except for the WR_PEND, ALRM_PEND and DAYALRM_PEND bits. Writes to any bit of this register has no effect. This register is cleared at reset.

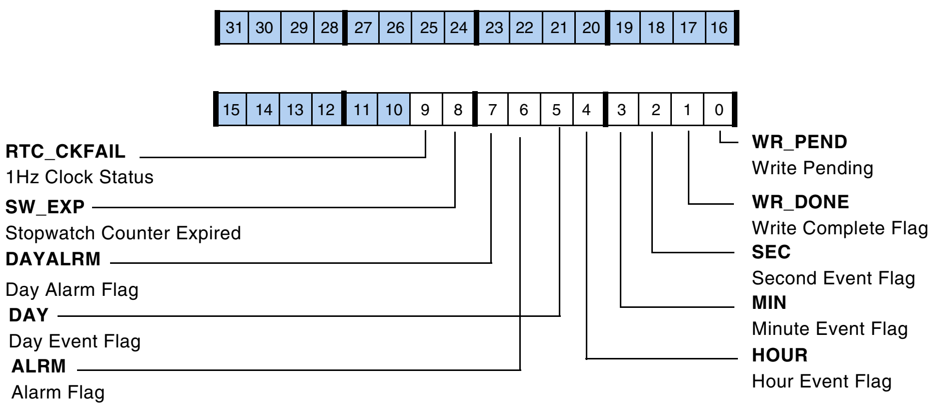

Figure A-55. RTC_STAT Register

Table A-73. RTC_STAT Register Bit Descriptions (ROC)

Bit	Name	Description
0 (RO)	WR_PEND	**1 Hz Register Write Pending.** Shows that write to 1 Hz registers (RTC_CLOCK, RTC_ALARM, RTC_SWTCH and RTC_INIT register) is pending. This bit is automatically cleared/set by hardware.
1	WR_DONE	**1 Hz Register Write Done.** Returns status of write access to 1 Hz registers. 0 = Write pending. 1 = Write done

Table A-73. RTC_STAT Register Bit Descriptions (ROC) (Cont'd)

Bit	Name	Description
2	SEC	**Second Event Flag.** 0 = Second event has not occurred 1 = Second event has occurred
3	MIN	**Minute Event Flag.** 0 = Minute event has not occurred 1 = Minute event has occurred (clock counter value x:y:z:59)
4	HOUR	**Hour Event Flag.** 0 = Hour event has not occurred 1 = Hour event has occurred (clock counter value x:y:59:59)
5	DAY	**Day Event Flag.** 0 = Day event has not occurred 1 = Day event has occurred (clock counter value x:23:59:59)
6 (RO)	ALRM	**Alarm Flag.** 0 = Daily alarm has not occurred 1 = Daily alarm has occurred
7 (RO)	DAYALRM	**Time of Day Flag.** 0 = Alarm has not occurred 1 = Alarm has occurred
8	SW_EXP	**Stop Watch Counter Expiration Flag.** 0 = Stop watch counter has not expired 1 = Stop watch counter expired
9	RTC_CKFAIL	**1 Hz Clock Status.** 0 = RTC 1 Hz clock is functional 1 = RTC 1 Hz clock failed

Stopwatch Count Register (RTC_STPWTCH)

This register, shown in Figure A-56, contains the countdown value for the stop watch. The stopwatch counts down seconds from the programmed value and generates an interrupt (if `STPWTCH_INTEN` = 1) when the count reaches 0. The register can be programmed to any value between 0 and

$(2^{16} - 1)$ seconds (that is, a range of 18 hours, 12 minutes and 15 seconds).

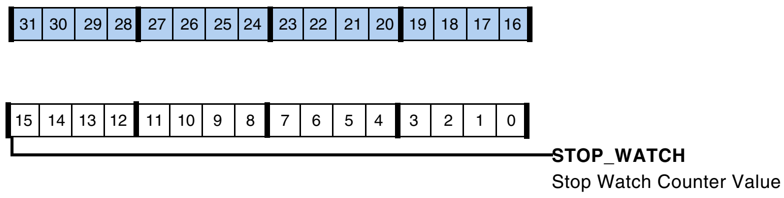

Figure A-56. RTC_STPWTCH Register (RW)

Clock Register (RTC_CLOCK)

This register, shown in Figure A-57, is used to read or write the current time. It has no reset and an undefined value when the module is first powered up. This register is updated every second. If RTC is already running when the core starts up, the values read from RTC_CLOCK are zero until the first second event occurs. In this case, programs must wait for the second event and then read the register. Writes of invalid time values are forbidden.

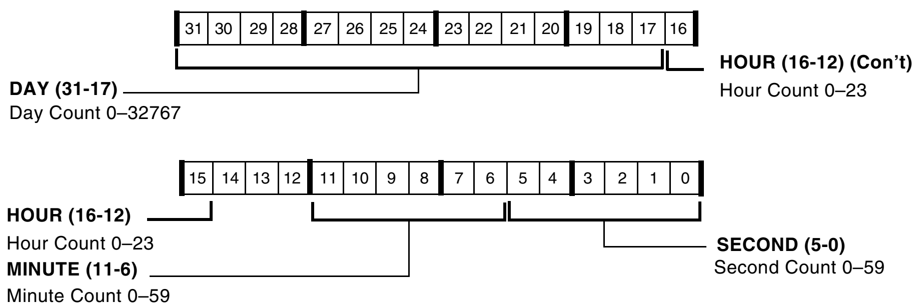

Figure A-57. RTC_CLOCK Register (RW)

Alarm Register (RTC_ALARM)

This register, shown in Figure A-58, is programmed by software for the time (in hours, minutes, and seconds) the alarm interrupt occurs. Reads and writes can occur at any time. The alarm interrupt occurs whenever the hour, minute, and second fields first match those of the RTC status register. The day interrupt occurs whenever the day, hour, minute, and second fields first match those of the RTC status register.

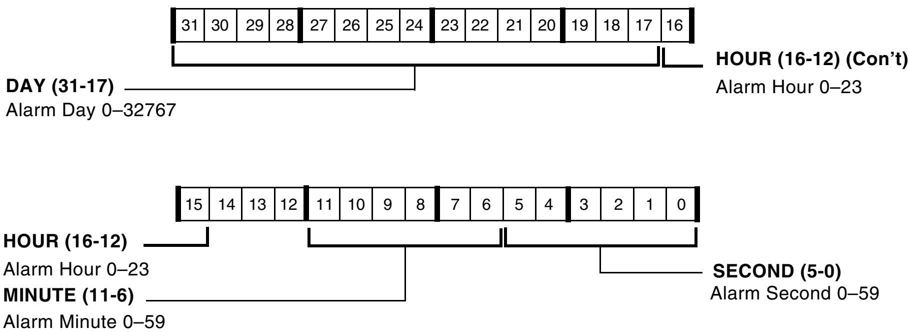

Figure A-58. RTC Alarm Register (RW)

Initialization Register (RTC_INIT)

This register, shown in Figure A-59 and described in Table A-74, provides the calibration function, powers down the unit, and grounds these buses.

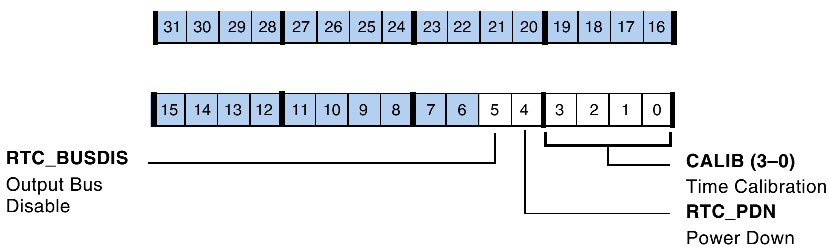

Figure A-59. RTC_INIT Register

Table A-74. RTC_INIT Register Bit Descriptions (RW)

Bit	Name	Description
3–0	CALIB	**Time Calibration.** Max ± 7 seconds. Calibration value is added/subtracted from the time every day at midnight. 1001 -> –1 second 1010 -> –2 second --------------------- 1111 -> –7 second 0001 -> +1 second 0010 -> +2 second ---------------------- 0111 -> +7 second
4	RTC_PDN	**RTC Power Down.** Active high. Write 1 to power down RTC oscillator, 0 to power up RTC oscillator. This bit must be compulsorily written once during first RTC oscillator power-up. 0 = RTC oscillator running 1 = Powers down RTC oscillator
5	RTC_BUSDIS	**Bus Disable.** Disables the bus and level shifter between core and IO domain. Setting this bit grounds this logic and prevents floating node consumption when the RTC is not used. 0 = Bus logic is enabled (RTC in use) 1 = Bus logic is disabled (RTC not in use)

Initialization Status Register (RTC_INITSTAT)

Figure A-60 and Table A-75 describe the bits in the RTC_INITSTAT register.

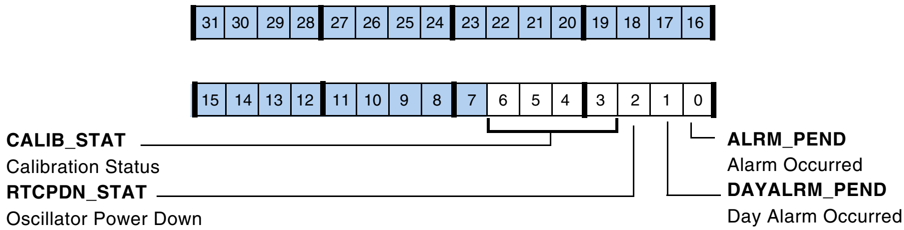

Figure A-60. RTC_INITSTAT Register

Table A-75. RTC_INITSTAT Register Bit Descriptions (RO)

Bit	Name	Description
0 (ROC)	ALRM_PEND	**Time Calibration.** Indicates that an alarm has occurred. (Useful if core has powered down or reset in the middle). 0 = No daily alarm has occurred 1 = A daily alarm has occurred This bit is cleared on reading RTC_INITSTAT.
1 (ROC)	DAYALRM_PEND	**RTC Power Down.** Indicates that an alarm has occurred. (Useful if core has powered down or reset in the middle.) 0 = No day alarm has occurred 1 = A day alarm had occurred This bit is cleared on reading RTC_INITSTAT.
2	RTCPDN_STAT	**Power Down Status.** Status of RTC oscillator power-down bit. 0 = The RTC oscillator is running 1 = The RTC oscillator is powered down
6–3	CALIB_STAT	**Calibration Status.** Indicates whether CALIB value in the RTC_INIT register has been successfully programmed in the RTC. It should be equal to the value of CALIB.

DAI Signal Routing Unit Registers

The digital applications interface is comprised of a group of peripherals and the signal routing unit (SRU). These register groups are described in the sections that follow.

For each group, the source signals (outputs) and all associated destination signals (inputs) are listed.

Group A – Clock Routing

The group A clock sources are listed in Table A-76.

Source Signals

Table A-76. Group A Sources – Serial Clock

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI_PB01_O	Pin Buffer 1
00001 (0x1)	DAI_PB02_O	Pin Buffer 2
00010 (0x2)	DAI_PB03_O	Pin Buffer 3
00011 (0x3)	DAI_PB04_O	Pin Buffer 4
00100 (0x4)	DAI_PB05_O	Pin Buffer 5
00101 (0x5)	DAI_PB06_O	Pin Buffer 6
00110 (0x6)	DAI_PB07_O	Pin Buffer 7
00111 (0x7)	DAI_PB08_O	Pin Buffer 8
01000 (0x8)	DAI_PB09_O	Pin Buffer 9
01001 (0x9)	DAI_PB10_O	Pin Buffer 10
01010 (0xA)	DAI_PB11_O	Pin Buffer 11
01011 (0xB)	DAI_PB12_O	Pin Buffer 12
01100 (0xC)	DAI_PB13_O	Pin Buffer 13

Table A-76. Group A Sources – Serial Clock (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
01101 (0xD)	DAI_PB14_O	Pin Buffer 14
01110 (0xE)	DAI_PB15_O	Pin Buffer 15
01111 (0xF)	DAI_PB16_O	Pin Buffer 16
10000 (0x10)	DAI_PB17_O	Pin Buffer 17
10001 (0x11)	DAI_PB18_O	Pin Buffer 18
10010 (0x12)	DAI_PB19_O	Pin Buffer 19
10011 (0x13)	DAI_PB20_O	Pin Buffer 20
10100 (0x14)	SPORT0_CLK_O	SPORT 0 Clock
10101 (0x15)	SPORT1_CLK_O	SPORT 1 Clock
10110 (0x16)	SPORT2_CLK_O	SPORT 2 Clock
10111 (0x17)	SPORT3_CLK_O	SPORT 3 Clock
11000 (0x18)	SPORT4_CLK_O	SPORT 4 Clock
11001 (0x19)	SPORT5_CLK_O	SPORT 5 Clock
11010 (0x1A)	DIR_CLK_O	SPDIF Receive Clock Output
11011 (0x1B)	DIR_TDMCLK_O	SPDIF Receive TDM Clock Output
11100 (0x1C)	PCG_CLKA_O	Precision Clock A Output
11101 (0x1D)	PCG_CLKB_O	Precision Clock B Output
11110 (0x1E)	LOW	Logic Level Low (0)
11111 (0x1F)	HIGH	Logic Level High (1)

Destination Signal Control Registers (SRU_CLKx)

The SRU_CLKx registers are shown in Figure A-61 through Figure A-66.

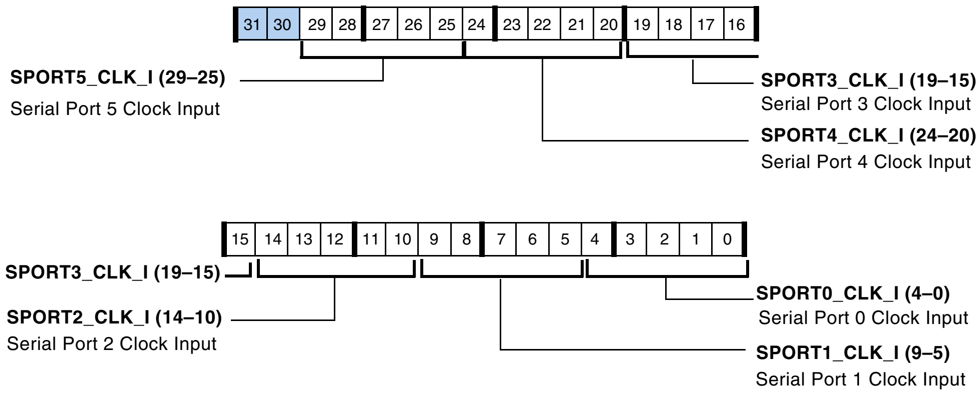

Figure A-61. SRU_CLK0 Register (RW)

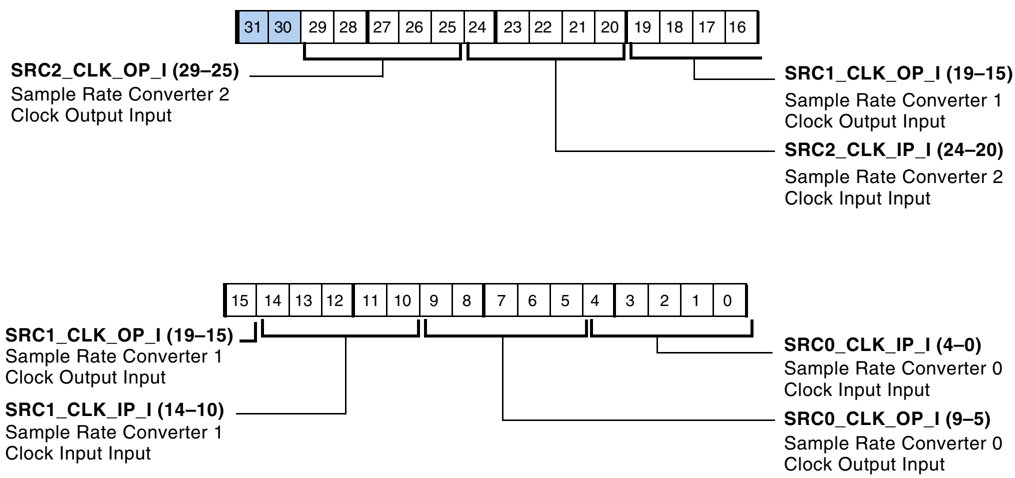

Figure A-62. SRU_CLK1 Register (RW)

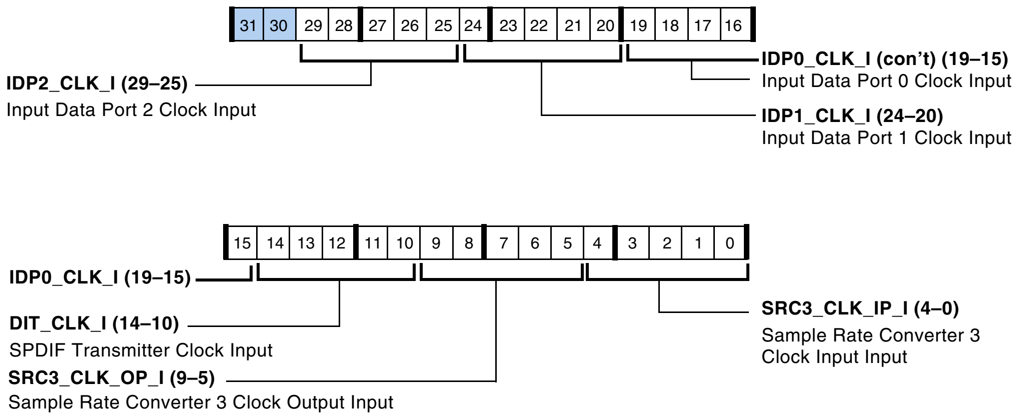

Figure A-63. SRU_CLK2 Register (RW)

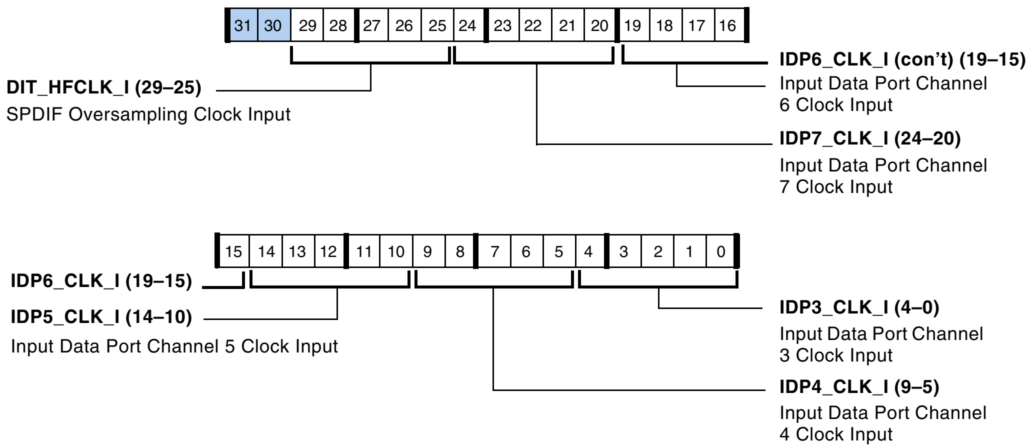

Figure A-64. SRU_CLK3 Register (RW)

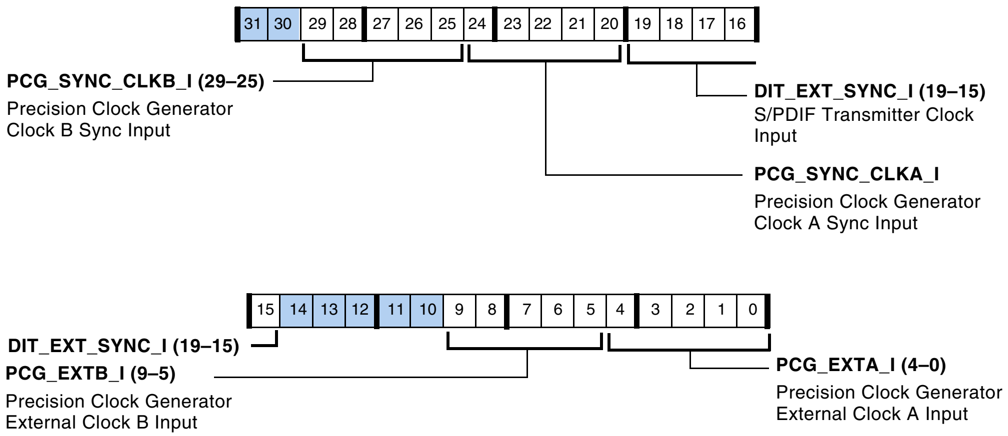

Figure A-65. SRU_CLK4 Register (RW)

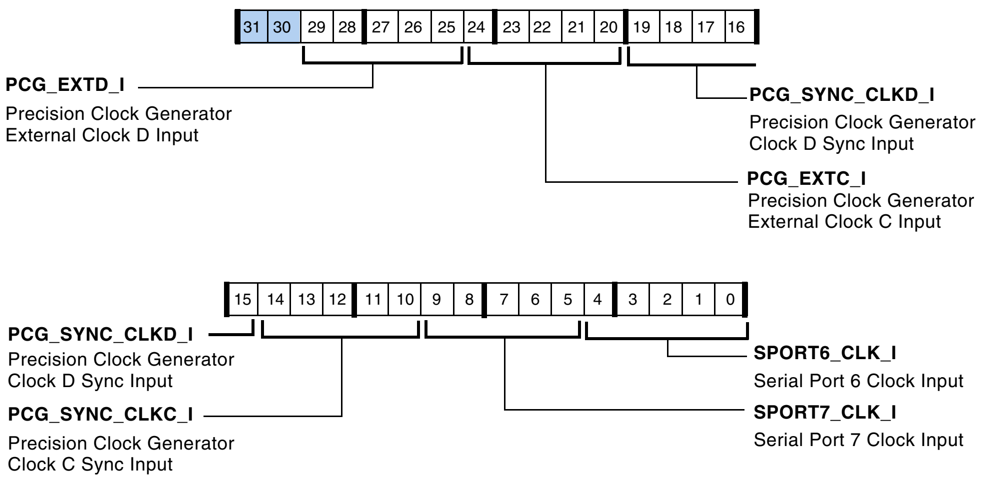

Figure A-66. SRU_CLK5 Register (RW)

Group B – Serial Data Routing

The data sources are based on the 6-bit values shown in Table A-77.

Source Signals

Table A-77. Group B Sources – Serial Data

Selection Code	Source Signal	Description (Source Selection)
000000 (0x0)	DAI_PB01_O	Pin Buffer 1
000001 (0x1)	DAI_PB02_O	Pin Buffer 2
000010 (0x2)	DAI_PB03_O	Pin Buffer 3
000011 (0x3)	DAI_PB04_O	Pin Buffer 4
000100 (0x4)	DAI_PB05_O	Pin Buffer 5
000101 (0x5)	DAI_PB06_O	Pin Buffer 6
000110 (0x6)	DAI_PB07_O	Pin Buffer 7
000111 (0x7)	DAI_PB08_O	Pin Buffer 8
001000 (0x8)	DAI_PB09_O	Pin Buffer 9
001001 (0x9)	DAI_PB10_O	Pin Buffer 10
001010 (0xA)	DAI_PB11_O	Pin Buffer 11
001011 (0xB)	DAI_PB12_O	Pin Buffer 12
001100 (0xC)	DAI_PB13_O	Pin Buffer 13
001101 (0xD)	DAI_PB14_O	Pin Buffer 14
001110 (0xE)	DAI_PB15_O	Pin Buffer 15
001111 (0xF)	DAI_PB16_O	Pin Buffer 16
010000 (0x10)	DAI_PB17_O	Pin Buffer 17
010001 (0x11)	DAI_PB18_O	Pin Buffer 18
010010 (0x12)	DAI_PB19_O	Pin Buffer 19
010011 (0x13)	DAI_PB20_O	Pin Buffer 20
010100 (0x14)	SPORT0_DA_O	SPORT 0A Data

Table A-77. Group B Sources – Serial Data (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
010101 (0x15)	SPORT0_DB_O	SPORT 0B Data
010110 (0x16)	SPORT1_DA_O	SPORT 1A Data
010111 (0x17)	SPORT1_DB_O	SPORT 1B Data
011000 (0x18)	SPORT2_DA_O	SPORT 2A Data
011001 (0x19)	SPORT2_DB_O	SPORT 2B Data
011010 (0x1A)	SPORT3_DA_O	SPORT 3A Data
011011 (0x1B)	SPORT3_DB_O	SPORT 3B Data
011100 (0x1C)	SPORT4_DA_O	SPORT 4A Data
011101 (0x1D)	SPORT4_DB_O	SPORT 4B Data
011110 (0x1E)	SPORT5_DA_O	SPORT 5A Data
011111 (0x1F)	SPORT5_DB_O	SPORT 5B Data
100000 (0x20)	SRC0_DAT_OP_O	SRC0 Data Out
100001 (0x21)	SRC1_DAT_OP_O	SRC1 Data Out
100010 (0x22)	SRC2_DAT_OP_O	SRC2 Data Out
100011 (0x23)	SRC3_DAT_OP_O	SRC3 Data Out
100100 (0x24)	SRC0_TDM_IP_O	SRC0 Data Out
100101 (0x25)	SRC1_TDM_IP_O	SRC1 Data Out
100110 (0x26)	SRC2_TDM_IP_O	SRC2 Data Out
100111 (0x27)	SRC3_TDM_IP_O	SRC3 Data Out
101000 (0x28)	DIR_DAT_O	SPDIF RX Serial Data Out
101100(0x2C)	SPORT6_DA_O	SPORT 6A Data
101101(0x2D)	SPORT6_DB_O	SPORT 6B Data
101110(0x2E)	SPORT7_DA_O	SPORT 7A Data
101111(0x2F)	SPORT7_DB_O	SPORT 7B Data
110000(0x30)	DIT_O	SPDIF TX Biphase Stream
110001(0x31)–111101(0x3D)	Reserved	

Table A-77. Group B Sources – Serial Data (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
111110 (0x3E)	LOW	Logic Level Low (0)
111111 (0x3F)	HIGH	Logic Level High (1)

Destination Signal Control Registers (SRU_DATx)

The serial data routing control registers, shown in Figure A-67 through Figure A-73 route serial data to the serial ports (A and B data channels) and the input data port.

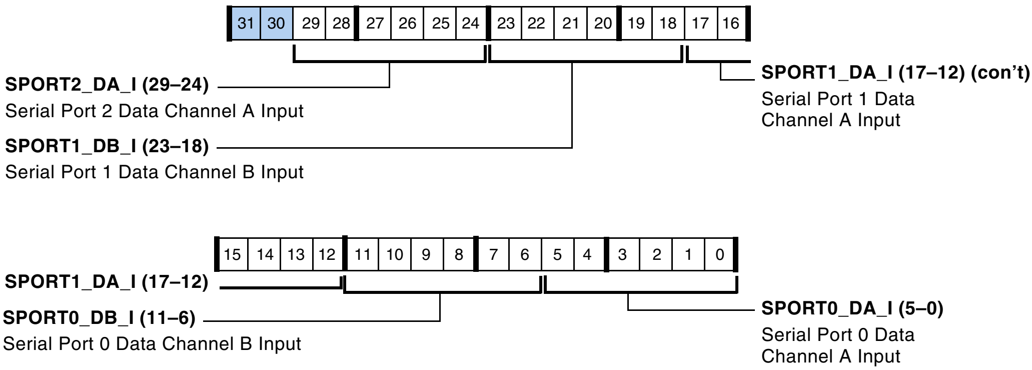

Figure A-67. SRU_DAT0 Register (RW)

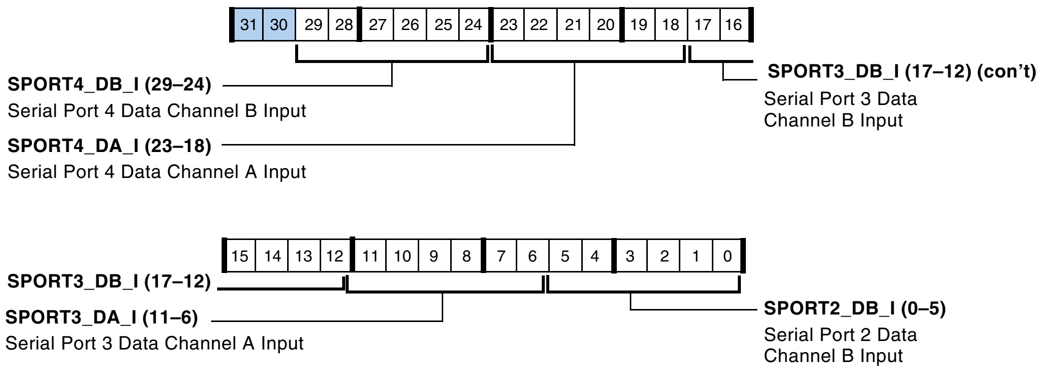

Figure A-68. SRU_DAT1 Register (RW)

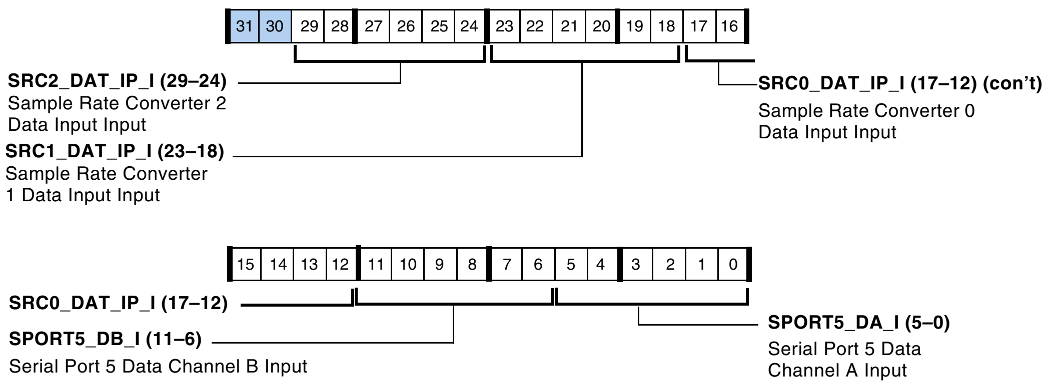

Figure A-69. SRU_DAT2 Register (RW)

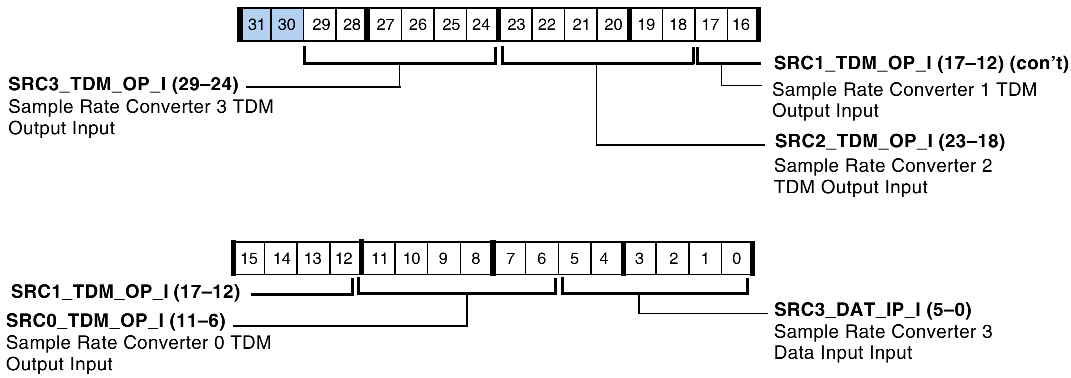

Figure A-70. SRU_DAT3 Register (RW)

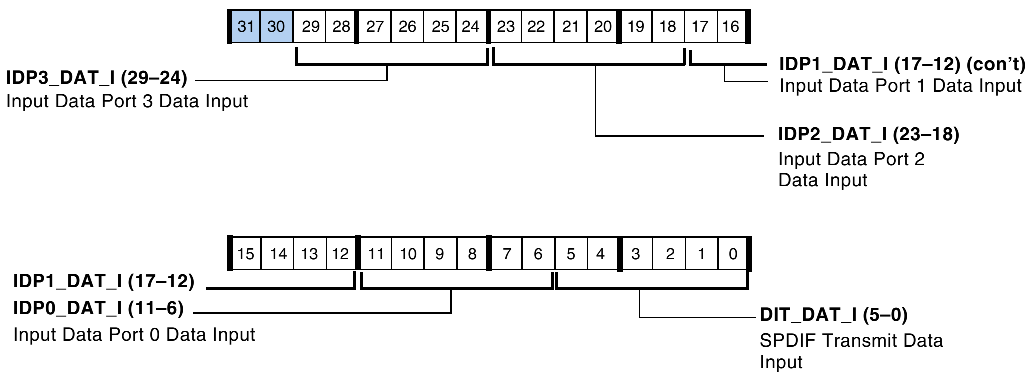

Figure A-71. SRU_DAT4 Register (RW)

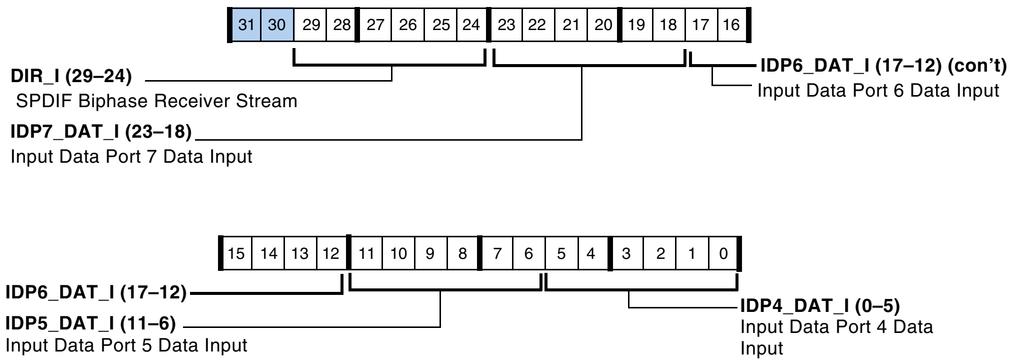

Figure A-72. SRU_DAT5 Register (RW)

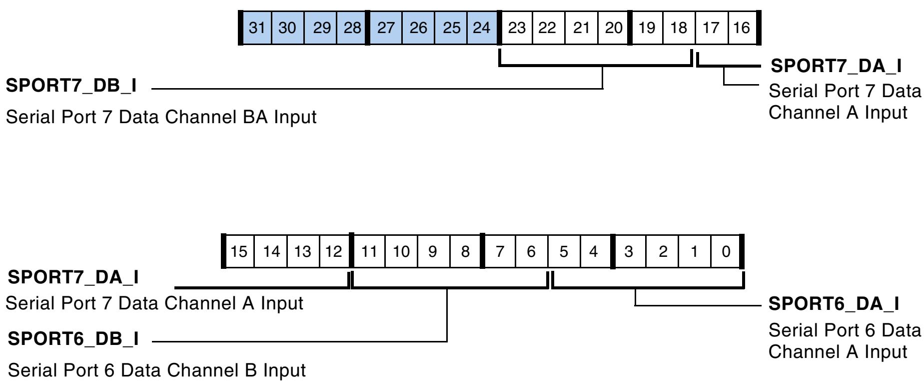

Figure A-73. SRU_DAT6 Register (RW)

Group C – Frame Sync Routing

The frame sync sources are based on the 5-bit values described in Table A-78.

Source Signals

Table A-78. Group C Sources – Frame Sync

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI_PB01_O	Pin Buffer 1
00001 (0x1)	DAI_PB02_O	Pin Buffer 2
00010 (0x2)	DAI_PB03_O	Pin Buffer 3
00011 (0x3)	DAI_PB04_O	Pin Buffer 4
00100 (0x4)	DAI_PB05_O	Pin Buffer 5
00101 (0x5)	DAI_PB06_O	Pin Buffer 6
00110 (0x6)	DAI_PB07_O	Pin Buffer 7
00111 (0x7)	DAI_PB08_O	Pin Buffer 8
01000 (0x8)	DAI_PB09_O	Pin Buffer 9
01001 (0x9)	DAI_PB10_O	Pin Buffer 10
01010 (0xA)	DAI_PB11_O	Pin Buffer 11
01011 (0xB)	DAI_PB12_O	Pin Buffer 12
01100 (0xC)	DAI_PB13_O	Pin Buffer 13
01101 (0xD)	DAI_PB14_O	Pin Buffer 14
01110 (0xE)	DAI_PB15_O	Pin Buffer 15
01111 (0xF)	DAI_PB16_O	Pin Buffer 16
10000 (0x10)	DAI_PB17_O	Pin Buffer 17
10001 (0x11)	DAI_PB18_O	Pin Buffer 18
10010 (0x12)	DAI_PB19_O	Pin Buffer 19
10011 (0x13)	DAI_PB20_O	Pin Buffer 20
10100 (0x14)	SPORT0_FS_O	SPORT 0 Frame Sync
10101 (0x15)	SPORT1_FS_O	SPORT 1 Frame Sync
10110 (0x16)	SPORT2_FS_O	SPORT 2 Frame Sync
10111 (0x17)	SPORT3_FS_O	SPORT 3 Frame Sync

Table A-78. Group C Sources – Frame Sync (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
11000 (0x18)	SPORT4_FS_O	SPORT 4 Frame Sync
11001 (0x19)	SPORT5_FS_O	SPORT 5 Frame Sync
11010 (0x1A)	DIR_FS_O	SPDIF RX Frame Sync Output
11011 (0x1B)	Reserved	
11100 (0x1C)	PCG_FSA_O	Precision Frame Sync A Output
11101 (0x1D)	PCG_FSB_O	Precision Frame Sync B Output
11110 (0x1E)	LOW	Logic Level Low (0)
11111 (0x1F)	HIGH	Logic Level High (1)

Destination Signal Control Registers (SRU_FSx)

The frame sync routing control registers are shown in Figure A-74 through Figure A-78.

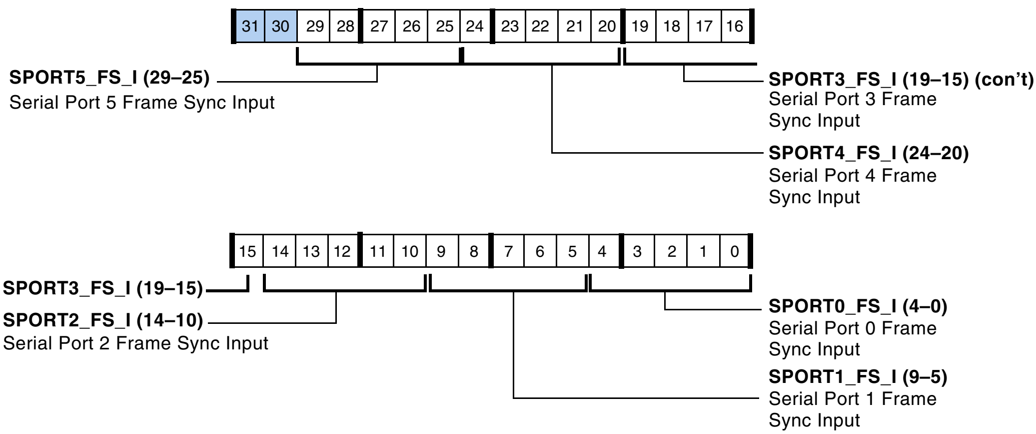

Figure A-74. SRU_FS0 Register (RW)

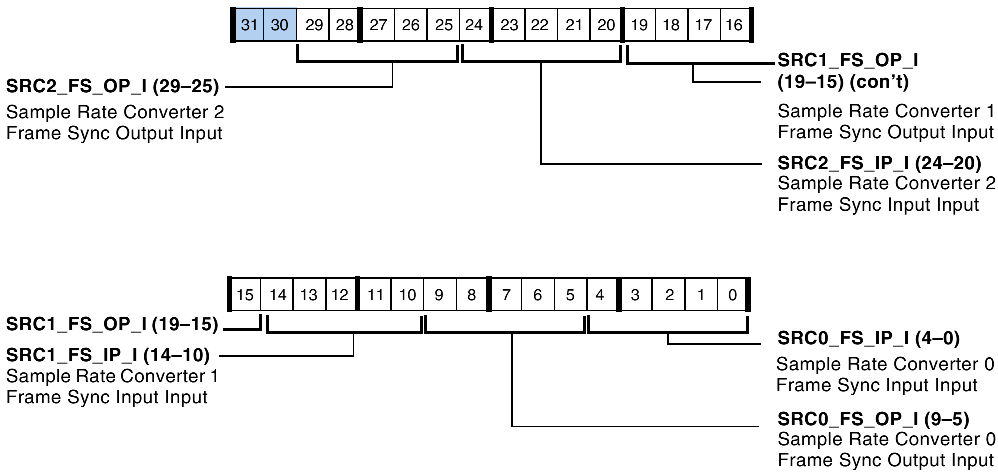

Figure A-75. SRU_FS1 Register (RW)

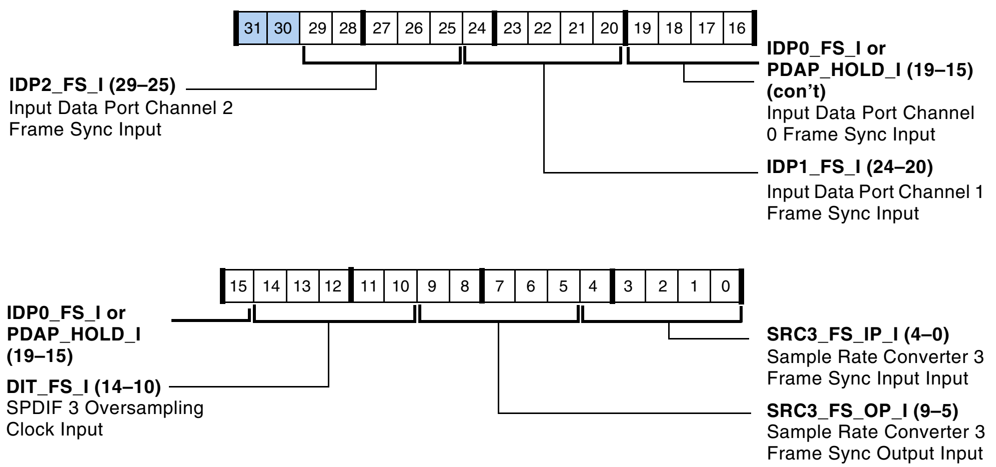

Figure A-76. SRU_FS2 Register (RW)

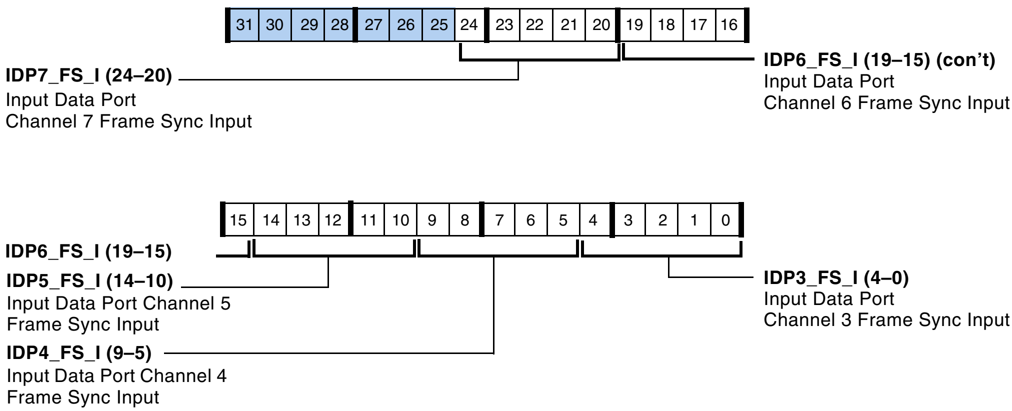

Figure A-77. SRU_FS3 Register (RW)

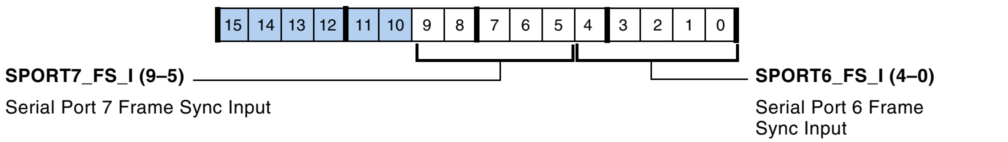

Figure A-78. SRU_FS4 Register (RW)

Group D – Pin Buffer Signal Assignments

Table A-79 lists the 7-bit source signals which may be assigned to the pin buffers.

Source Signals

Table A-79. Group D Sources – Pin Signal Assignments

Selection Code	Source Signal	Description (Source Selection)
0000000 (0x0)	DAI_PB01_O	Pin Buffer 1
0000001 (0x1)	DAI_PB02_O	Pin Buffer 2
0000010 (0x2)	DAI_PB03_O	Pin Buffer 3
0000011 (0x3)	DAI_PB04_O	Pin Buffer 4

Table A-79. Group D Sources – Pin Signal Assignments (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
0000100 (0x4)	DAI_PB05_O	Pin Buffer 5
0000101 (0x5)	DAI_PB06_O	Pin Buffer 6
0000110 (0x6)	DAI_PB07_O	Pin Buffer 7
0000111 (0x7)	DAI_PB08_O	Pin Buffer 8
0001000 (0x8)	DAI_PB09_O	Pin Buffer 9
0001001 (0x9)	DAI_PB10_O	Pin Buffer 10
0001010 (0xA)	DAI_PB11_O	Pin Buffer 11
0001011 (0xB)	DAI_PB12_O	Pin Buffer 12
0001100 (0xC)	DAI_PB13_O	Pin Buffer 13
0001101 (0xD)	DAI_PB14_O	Pin Buffer 14
0001110 (0xE)	DAI_PB15_O	Pin Buffer 15
0001111 (0xF)	DAI_PB16_O	Pin Buffer 16
0010000 (0x10)	DAI_PB17_O	Pin Buffer 17
0010001 (0x11)	DAI_PB18_O	Pin Buffer 18
0010010 (0x12)	DAI_PB19_O	Pin Buffer 19
0010011 (0x13)	DAI_PB20_O	Pin Buffer 20
0010100 (0x14)	SPORT0_DA_O	SPORT 0A Data
0010101 (0x15)	SPORT0_DB_O	SPORT 0B Data
0010110 (0x16)	SPORT1_DA_O	SPORT 1A Data
0010111 (0x17)	SPORT1_DB_O	SPORT 1B Data
0011000 (0x18)	SPORT2_DA_O	SPORT 2A Data
0011001 (0x19)	SPORT2_DB_O	SPORT 2B Data
0011010 (0x1A)	SPORT3_DA_O	SPORT 3A Data
0011011 (0x1B)	SPORT3_DB_O	SPORT 3B Data
0011100 (0x1C)	SPORT4_DA_O	SPORT 4A Data
0011101 (0x1D)	SPORT4_DB_O	SPORT 4B Data

Table A-79. Group D Sources – Pin Signal Assignments (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
0011110 (0x1E)	SPORT5_DA_O	SPORT 5A Data
0011111 (0x1F)	SPORT5_DB_O	SPORT 5B Data
0100000 (0x20)	SPORT0_CLK_O	SPORT 0 Clock
0100001 (0x21)	SPORT1_CLK_O	SPORT 1 Clock
0100010 (0x22)	SPORT2_CLK_O	SPORT 2 Clock
0100011 (0x23)	SPORT3_CLK_O	SPORT 3 Clock
0100100 (0x24)	SPORT4_CLK_O	SPORT 4 Clock
0100101 (0x25)	SPORT5_CLK_O	SPORT 5 Clock
0100110 (0x26)	SPORT0_FS_O	SPORT 0 Frame Sync
0100111 (0x27)	SPORT1_FS_O	SPORT 1 Frame Sync
0101000 (0x28)	SPORT2_FS_O	SPORT 2 Frame Sync
0101001 (0x29)	SPORT3_FS_O	SPORT 3 Frame Sync
0101010 (0x2A)	SPORT4_FS_O	SPORT 4 Frame Sync
0101011 (0x2B)	SPORT5_FS_O	SPORT 5 Frame Sync
0101100 (0x2C)	SPORT6_DA_O	SPORT 6A Data
0101101 (0x2D)	SPORT6_DB_O	SPORT 6B Data
0101110 (0x2E)	SPORT7_DA_O	SPORT 7A Data
0101111 (0x2F)	SPORT7_DB_O	SPORT 7B Data
0110000 (0x30)	PDAP_STRB_O	PDAP Data Transfer Request Strobe
0110001 (0x31)	DIT_BLKSTART_O	S/PDIF TX Block Start Output
0110100 (0x34)	SPORT6_CLK_O	SPORT 6 Clock
0110101 (0x35)	SPORT7_CLK_O	SPORT 7 Clock
0110110 (0x36)	SPORT6_FS_O	SPORT 6 Frame Sync
0110111 (0x37)	SPORT7_FS_O	SPORT 7 Frame Sync
0111000 (0x38)	PCG_CLKA_O	Precision Clock A
0111001 (0x39)	PCG_CLKB_O	Precision Clock B

Table A-79. Group D Sources – Pin Signal Assignments (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
0111010 (0x3A)	PCG_FSA_O	Precision Frame Sync A
0111011 (0x3B)	PCG_FSB_O	Precision Frame Sync B
0111100 (0x3C)	Reserved	
0111101 (0x3D)	SRC0_DAT_OP_O	SRC0 Data Output
0111110 (0x3E)	SRC1_DAT_OP_O	SRC1 Data Output
0111111 (0x3F)	SRC2_DAT_OP_O	SRC2 Data Output
1000000 (0x40)	SRC3_DAT_OP_O	SRC3 Data Output
1000001 (0x41)	DIR_DAT_O	SPDIF_RX Data Output
1000010 (0x42)	DIR_FS_O	SPDIF_RX Frame Sync Output
1000011 (0x43)	DIR_CLK_O	SPDIF_RX Clock Output
1000100 (0x44)	DIR_TDMCLK_O	SPDIF_RX TDM Clock Output
1000101 (0x45)	DIT_O	SPDIF TX Biphase Encoded Data Output
1000110 (0x46)	SPORT0_TDV_O	SPORT0 Transmit Data Valid Output
1000111 (0x47)	SPORT1_TDV_O	SPORT1 Transmit Data Valid Output
1001000 (0x48)	SPORT2_TDV_O	SPORT2 Transmit Data Valid Output
1001001 (0x49)	SPORT3_TDV_O	SPORT3 Transmit Data Valid Output
1001010 (0x4A)	SPORT4_TDV_O	SPORT4 Transmit Data Valid Output
1001011 (0x4B)	SPORT5_TDV_O	SPORT5 Transmit Data Valid Output
1001100 (0x4C)	SPORT6_TDV_O	SPORT6 Transmit Data Valid Output
1001101 (0x4D)	SPORT7_TDV_O	SPORT7 Transmit Data Valid Output
1001110 (0x4E)	Reserved	
1001111 (0x4F)	Reserved	
1010000 (0x50)	PCG_CLKC_O	Precision Clock C
1011001 (0x51)	PCG_CLKD_O	Precision Clock D
1011010 (0x52)	PCG_FSC_O	Precision Frame Sync C
1010011 (0x53)	PCG_FSD_O	Precision Frame Sync D

Table A-79. Group D Sources – Pin Signal Assignments (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
1010100 – 1111101	Reserved	
1111110 (0x7E)	LOW	Logic Level Low (0)
1111111 (0x7F)	HIGH	Logic Level High (1)

Destination Signal Control Registers (SRU_PINx)

Each physical pin (connected to a bonded pad) may be routed using the pin signal assignment registers shown in Figure A-79 through Figure A-83.

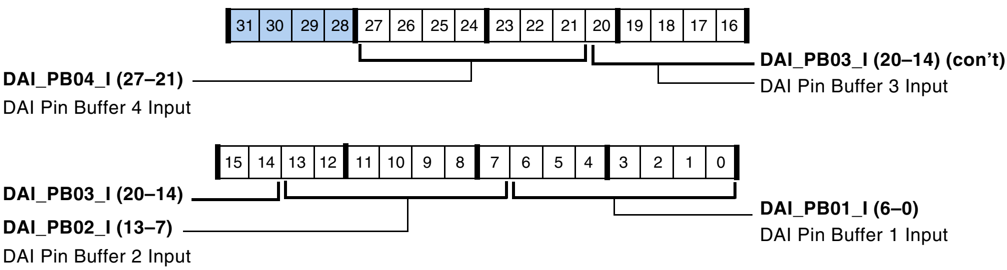

Figure A-79. SRU_PIN0 Register (RW)

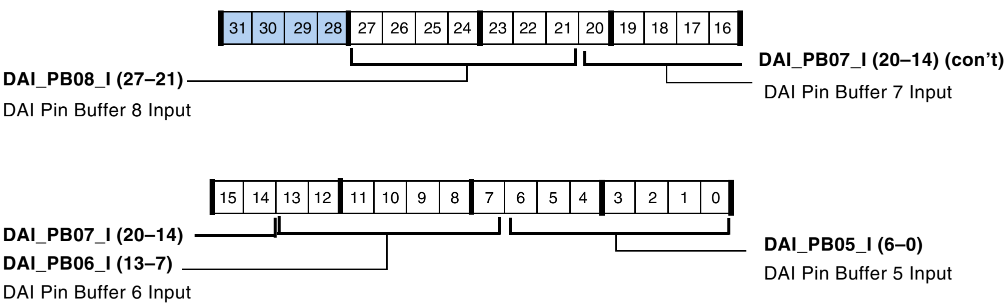

Figure A-80. SRU_PIN1 Register (RW)

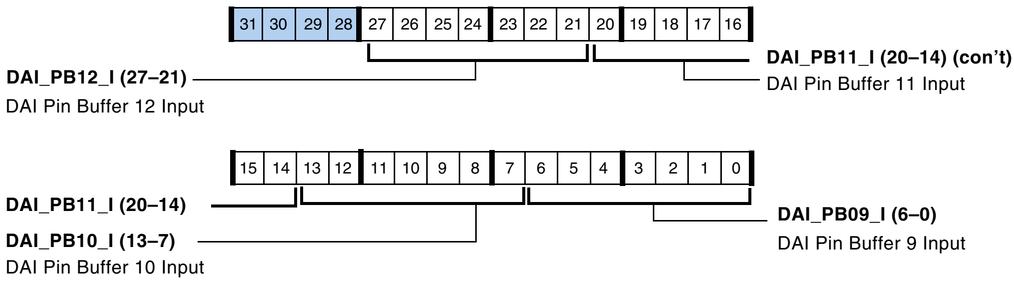

Figure A-81. SRU_PIN2 Register (RW)

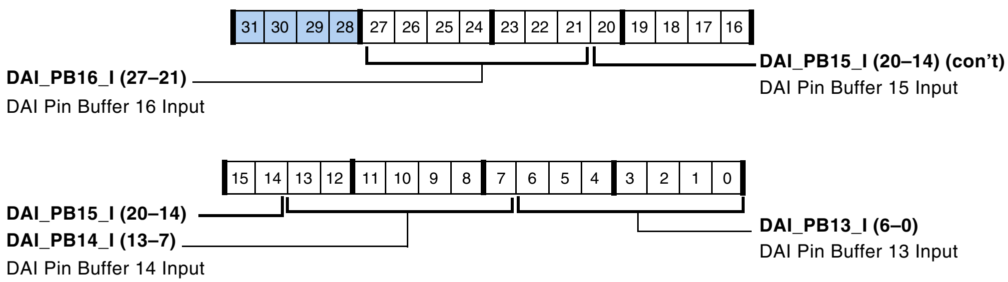

Figure A-82. SRU_PIN3 Register (RW)

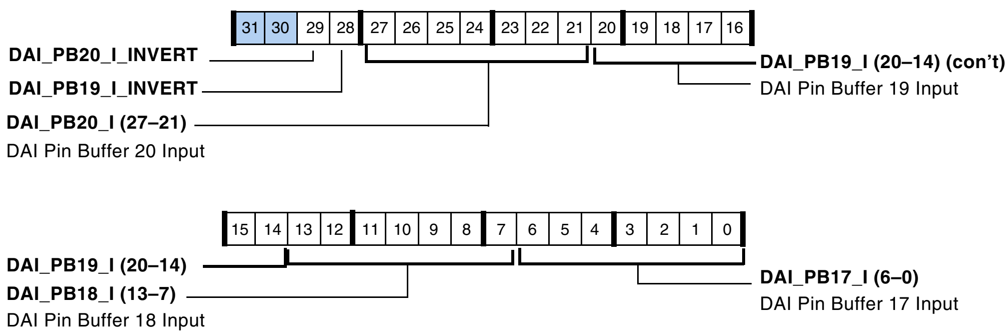

Figure A-83. SRU_PIN4 Register (RW)

Group E – Miscellaneous Signals

The miscellaneous signal routing registers correspond to the group E miscellaneous signals, listed in Table A-80.

Source Signals

Table A-80. Group E Sources – Miscellaneous Signals

Selection Code	Source Signal	Description (Output Source Selection)
00000 (0x0)	DAI_PB01_O	Pin Buffer 1 Output
00001 (0x1)	DAI_PB02_O	Pin Buffer 2 Output
00010 (0x2)	DAI_PB03_O	Pin Buffer 3 Output
00011 (0x3)	DAI_PB04_O	Pin Buffer 4 Output
00100 (0x4)	DAI_PB05_O	Pin Buffer 5 Output
00101 (0x5)	DAI_PB06_O	Pin Buffer 6 Output
00110 (0x6)	DAI_PB07_O	Pin Buffer 7 Output
00111 (0x7)	DAI_PB08_O	Pin Buffer 8 Output
01000 (0x8)	DAI_PB09_O	Pin Buffer 9 Output
01001 (0x9)	DAI_PB10_O	Pin Buffer 10 Output
01010 (0xA)	DAI_PB11_O	Pin Buffer 11 Output
01011 (0xB)	DAI_PB12_O	Pin Buffer 12 Output
01100 (0xC)	DAI_PB13_O	Pin Buffer 13 Output
01101 (0xD)	DAI_PB14_O	Pin Buffer 14 Output
01110 (0xE)	DAI_PB15_O	Pin Buffer 15 Output
01111 (0xF)	DAI_PB16_O	Pin Buffer 16 Output
10000 (0x10)	DAI_PB17_O	Pin Buffer 17 Output
10001 (0x11)	DAI_PB18_O	Pin Buffer 18 Output
10010 (0x12)	DAI_PB19_O	Pin Buffer 19 Output
10011 (0x13)	DAI_PB20_O	Pin Buffer 20 Output

Table A-80. Group E Sources – Miscellaneous Signals (Cont'd)

Selection Code	Source Signal	Description (Output Source Selection)
10100 (0x14)	SPORT0_FS_O	SPORT0 Frame Sync
10101 (0x15)	SPORT1_FS_O	SPORT1 Frame Sync
10110 (0x16)	SPORT2_FS_O	SPORT2 Frame Sync
10111 (0x17)	SPORT3_FS_O	SPORT3 Frame Sync
11000 (0x18)	SPORT4_FS_O	SPORT4 Frame Sync
11001 (0x19)	SPORT5_FS_O	SPORT5 Frame Sync
11010 (0x1A)	DIT_BLKSTART_O	S/PDIF TX Block Start Output
11011 (0x1B)	PCG_FSA_O	Precision Frame Sync A
11100 (0x1C)	PCG_CLKB_O	Precision Clock B
11101 (0x1D)	PCG_FSB_O	Precision Frame Sync B
11110 (0x1E)	LOW	Logic Level Low (0) as a Source
11111 (0x1F)	HIGH	Logic Level High (1) as a Source

Destination Signal Control Registers (SRU_MISCx)

Miscellaneous registers are shown in Figure A-84 and Figure A-85.

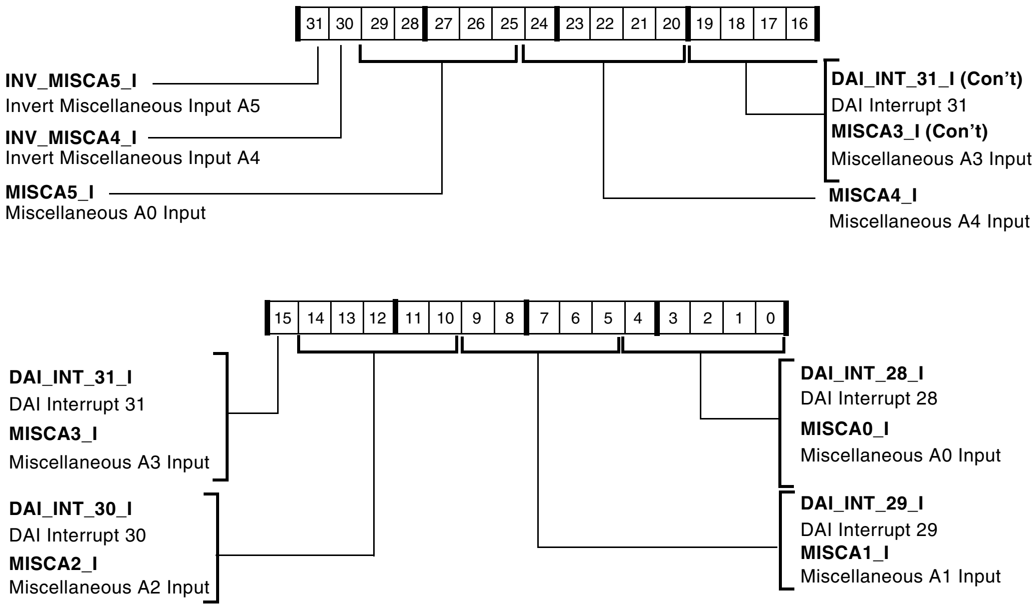

Figure A-84. SRU_EXT_MISCA Register (RW)

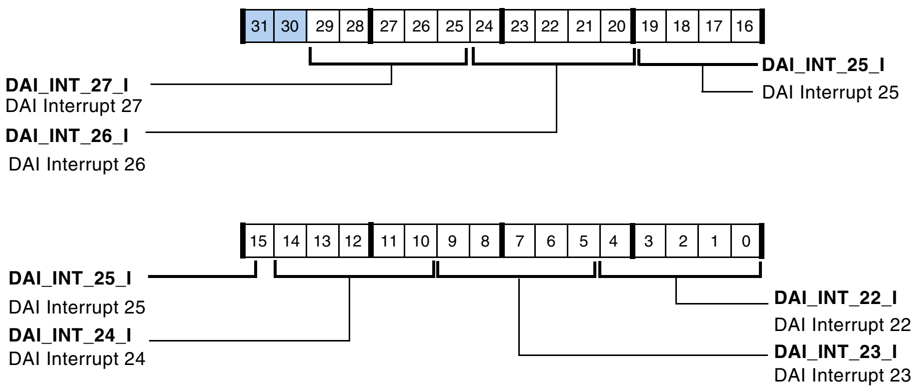

Figure A-85. SRU_EXT_MISCB Register (RW)

Group F – DAI Pin Buffer Enable

The 6-bit source encodings are listed in Table A-81.

Source Signals

Table A-81. Group F Sources – Pin Output Enable

Selection Code	Source Signal	Description (Output Source Selection)
000000 (0x0)	LOW	Logic Level Low (0)
000001 (0x1)	HIGH	Logic Level High (1)
000010 (0x2)	MISCA0_O	Miscellaneous Control A0 Output
000011 (0x3)	MISCA1_O	Miscellaneous Control A1 Output
000100 (0x4)	MISCA2_O	Miscellaneous Control A2 Output
000101 (0x5)	MISCA3_O	Miscellaneous Control A3 Output
000110 (0x6)	MISCA4_O	Miscellaneous Control A4 Output
000111 (0x7)	MISCA5_O	Miscellaneous Control A5 Output
001000 (0x8)	SPORT0_CLK_PBEN_O	SPORT 0 Clock Output Enable
001001 (0x9)	SPORT0_FS_PBEN_O	SPORT 0 Frame Sync Output Enable
001010 (0xA)	SPORT0_DA_PBEN_O	SPORT 0 Data Channel A Output Enable
001011 (0xB)	SPORT0_DB_PBEN_O	SPORT 0 Data Channel B Output Enable
001100 (0xC)	SPORT1_CLK_PBEN_O	SPORT 1 Clock Output Enable
001101 (0xD)	SPORT1_FS_PBEN_O	SPORT 1 Frame Sync Output Enable
001110 (0xE)	SPORT1_DA_PBEN_O	SPORT 1 Data Channel A Output Enable
001111 (0xF)	SPORT1_DB_PBEN_O	SPORT 1 Data Channel B Output Enable
010000 (0x10)	SPORT2_CLK_PBEN_O	SPORT 2 Clock Output Enable
010001 (0x11)	SPORT2_FS_PBEN_O	SPORT 2 Frame Sync Output Enable
010010 (0x12)	SPORT2_DA_PBEN_O	SPORT 2 Data Channel A Output Enable
010011 (0x13)	SPORT2_DB_PBEN_O	SPORT 2 Data Channel B Output Enable
010100 (0x14)	SPORT3_CLK_PBEN_O	SPORT 3 Clock Output Enable

Table A-81. Group F Sources – Pin Output Enable (Cont'd)

Selection Code	Source Signal	Description (Output Source Selection)
010101 (0x15)	SPORT3_FS_PBEN_O	SPORT 3 Frame Sync Output Enable
010110 (0x16)	SPORT3_DA_PBEN_O	SPORT 3 Data Channel A Output Enable
010111 (0x17)	SPORT3_DB_PBEN_O	SPORT 3 Data Channel B Output Enable
011000 (0x18)	SPORT4_CLK_PBEN_O	SPORT 4 Clock Output Enable
011001 (0x19)	SPORT4_FS_PBEN_O	SPORT 4 Frame Sync Output Enable
011010 (0x1A)	SPORT4_DA_PBEN_O	SPORT 4 Data Channel A Output Enable
011011 (0x1B)	SPORT4_DB_PBEN_O	SPORT 4 Data Channel B Output Enable
011100 (0x1C)	SPORT5_CLK_PBEN_O	SPORT 5 Clock Output Enable
011101 (0x1D)	SPORT5_FS_PBEN_O	SPORT 5 Frame Sync Output Enable
011110 (0x1E)	SPORT5_DA_PBEN_O	SPORT 5 Data Channel A Output Enable
011111 (0x1F)	SPORT5_DB_PBEN_O	SPORT 5 Data Channel B Output Enable
100000 (0x20)	SPORT6_CLK_PBEN_O	SPORT 6 Clock Output Enable
100001 (0x21)	SPORT6_FS_PBEN_O	SPORT 6 Frame Sync Output Enable
100010 (0x22)	SPORT6_DA_PBEN_O	SPORT 6 Data Channel A Output Enable
100011 (0x23)	SPORT6_DB_PBEN_O	SPORT 6 Data Channel B Output Enable
100100 (0x24)	SPORT7_CLK_PBEN_O	SPORT 7 Clock Output Enable
100101 (0x25)	SPORT7_FS_PBEN_O	SPORT 7 Frame Sync Output Enable
100110 (0x26)	SPORT7_DA_PBEN_O	SPORT 7 Data Channel A Output Enable
100111 (0x27)	SPORT7_DB_PBEN_O	SPORT 7 Data Channel B Output Enable
101000 (0x28)	SPORT0_TDV_PBEN_O	SPORT 0 Transmit Data Valid Output
101001 (0x29)	SPORT1_TDV_PBEN_O	SPORT 1 Transmit Data Valid Output
101010 (0x2A)	SPORT2_TDV_PBEN_O	SPORT 2 Transmit Data Valid Output
101011 (0x2B)	SPORT3_TDV_PBEN_O	SPORT 3 Transmit Data Valid Output
101100 (0x2C)	SPORT4_TDV_PBEN_O	SPORT 4 Transmit Data Valid Output
101101 (0x2D)	SPORT5_TDV_PBEN_O	SPORT 5 Transmit Data Valid Output
100111 (0x2E)	SPORT6_TDV_PBEN_O	SPORT 6 Transmit Data Valid Output

Table A-81. Group F Sources – Pin Output Enable (Cont'd)

Selection Code	Source Signal	Description (Output Source Selection)
101110 (0x2F)	SPORT7_TDV_PBEN_O	SPORT 7 Transmit Data Valid Output
110000 (0x30)–1111111 (0x3F)		Reserved

Destination Signal Control Registers (SRU_PBENx)

The pin buffer enable registers are shown in Figure A-86 through Figure A-89.

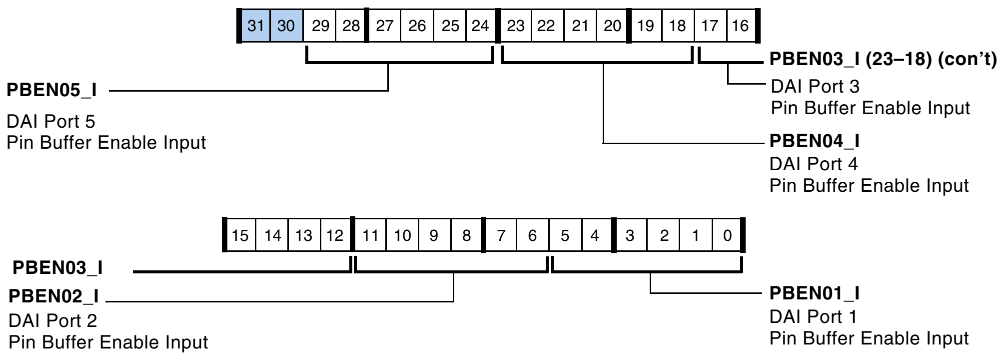

Figure A-86. SRU_PBEN0 (RW)

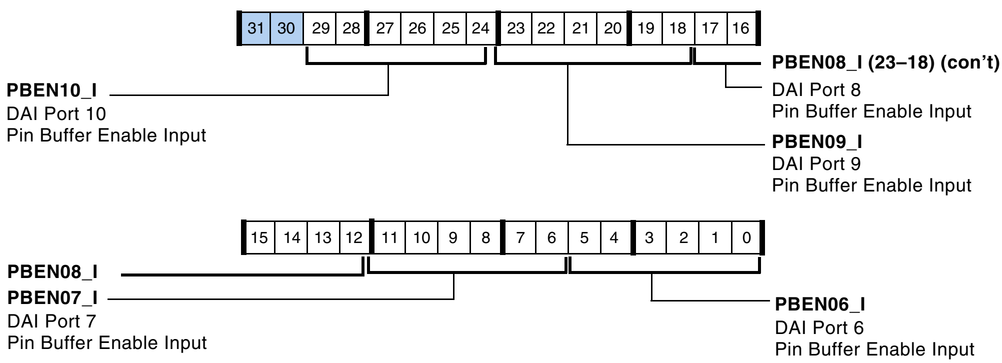

Figure A-87. SRU_PBEN1 (RW)

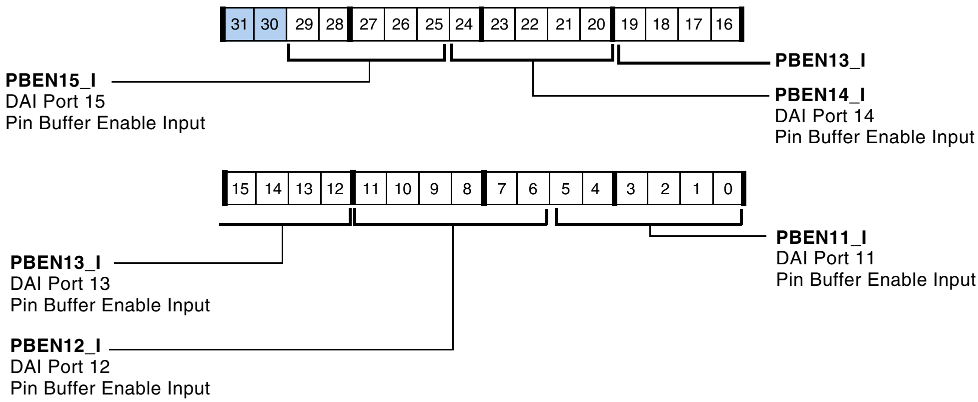

Figure A-88. SRU_PBEN2 (RW)

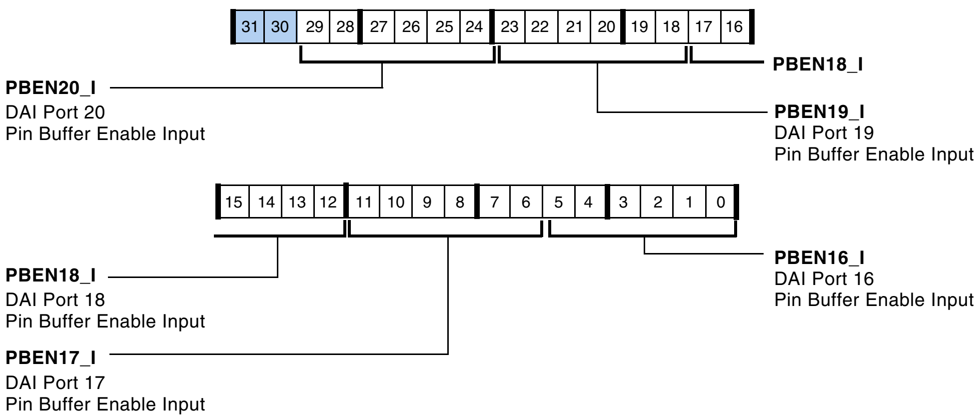

Figure A-89. SRU_PBEN3 (RW)

Group H – Shift Register Clock Routing (ADSP-2147x)

Table A-82 shows the list of sources for the SR_SCLK_I and SR_LAT_I input signals.

Source Signals

Table A-82. Group H Sources – Shift Register Clock Routing

Selection Code	Source Signal	Description (Output Source Selection)
00000 (0x0)	LOW	Logic Level Low (0)
00001 (0x1)	HIGH	Logic Level High (1)
00010 (0x2)	SPORT0_CLK_O	SPORT 0 Clock Output
00011 (0x3)	SPORT1_CLK_O	SPORT 1 Clock Output
00100 (0x4)	SPORT2_CLK_O	SPORT 2 Clock Output
00101 (0x5)	SPORT3_CLK_O	SPORT 3 Clock Output
00110 (0x6)	SPORT4_CLK_O	SPORT 4 Clock Output
00111 (0x7)	SPORT5_CLK_O	SPORT 5 Clock Output
01000 (0x8)	SPORT6_CLK_O	SPORT 6 Clock Output
01001 (0x9)	SPORT7_CLK_O	SPORT 7 Clock Output
01010 (0xA)	SPORT0_FS_O	SPORT 0 Frame Sync Output
01011 (0xB)	SPORT1_FS_O	SPORT 1 Frame Sync Output
01100 (0xC)	SPORT2_FS_O	SPORT 2 Frame Sync Output
01101 (0xD)	SPORT3_FS_O	SPORT 3 Frame Sync Output
01110 (0xE)	SPORT4_FS_O	SPORT 4 Frame Sync Output
01111 (0xF)	SPORT5_FS_O	SPORT 5 Frame Sync Output
10000 (0x10)	SPORT6_FS_O	SPORT 6 Frame Sync Output
10001 (0x11)	SPORT7_FS_O	SPORT 7 Frame Sync Output
10010 (0x12)	SR_SCLK_O	Dedicated SR_SCLK Pin

Table A-82. Group H Sources – Shift Register Clock Routing (Cont'd)

Selection Code	Source Signal	Description (Output Source Selection)
10011 (0x13)	SR_LAT_O	Dedicated SR_LAT Pin
10100 (0x14)	PCG_CLKA_O	PCG Clock A Output
10101 (0x15)	PCG_CLKB_O	PCG Clock B Output
10110 (0x16)	PCG_FSA_O	PCG Frame Sync A Output
10111 (0x17)	PCG_FSB_O	PCG Frame Sync B Output
11000 (0x18)	DAI_PB01_O	Pin Buffer 1
11001 (0x19)	DAI_PB02_O	Pin Buffer 2
11010 (0x1A)	DAI_PB03_O	Pin Buffer 3
11011 (0x1B)	DAI_PB04_O	Pin Buffer 4
11100 (0x1C)	DAI_PB05_O	Pin Buffer 5
11101 (0x1D)	DAI_PB06_O	Pin Buffer 6
11110 (0x1E)	DAI_PB07_O	Pin Buffer 7
11111 (0x1F)	DAI_PB08_O	Pin Buffer 8

Destination Control Signal Register (SR_CLK_SHREG)

Figure A-90 shows the programmable options for SR_SCLK_I and SR_LAT_I input signals.

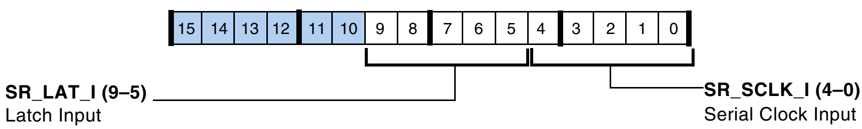

Figure A-90. SR_CLK_SHREG Register (RW)

Group I – Shift Register Serial Data Routing Register (ADSP-2147x)

Table A-83 show the list of sources for the SR_SDI_I input signal.

Source Signals

Table A-83. Group I Sources – Shift Register Serial Data Routing

Selection Code	Source Signal	Description (Output Source Selection)
00000 (0x0)	LOW	Logic Level Low (0)
00001 (0x1)	HIGH	Logic Level High (1)
00010 (0x2)	SPORT0_DA_O	SPORT 0 Data Channel A
00011 (0x3)	SPORT0_DB_O	SPORT 0 Data Channel B
00100 (0x4)	SPORT1_DA_O	SPORT 1 Data Channel A
00101 (0x5)	SPORT1_DB_O	SPORT 1 Data Channel B
00110 (0x6)	SPORT2_DA_O	SPORT 2 Data Channel A
00111 (0x7)	SPORT2_DB_O	SPORT 2 Data Channel B
01000 (0x8)	SPORT3_DA_O	SPORT 3 Data Channel A
01001 (0x9)	SPORT3_DB_O	SPORT 3 Data Channel B
01010 (0xA)	SPORT4_DA_O	SPORT 4 Data Channel A
01011 (0xB)	SPORT4_DB_O	SPORT 4 Data Channel B
01100 (0xC)	SPORT5_DA_O	SPORT 5 Data Channel A
01101 (0xD)	SPORT5_DB_O	SPORT 5 Data Channel B
01110 (0xE)	SPORT6_DA_O	SPORT 6 Data Channel A
01111 (0xF)	SPORT6_DB_O	SPORT 6 Data Channel B
10000 (0x10)	SPORT7_DA_O	SPORT 7 Data Channel A
10001 (0x11)	SPORT7_DB_O	SPORT 7 Data Channel B
10010 (0x12)	SR_SDAT_O	Dedicated SR_SDI Pin
10011 (0x13)	DAI_PB01_O	Pin Buffer 1

Table A-83. Group I Sources – Shift Register Serial Data Routing (Cont'd)

Selection Code	Source Signal	Description (Output Source Selection)
10100 (0x14)	DAI_PB02_O	Pin Buffer 2
10101 (0x15)	DAI_PB03_O	Pin Buffer 3
10110 (0x16)	DAI_PB04_O	Pin Buffer 4
10111 (0x17)	DAI_PB05_O	Pin Buffer 5
11000 (0x18)	DAI_PB06_O	Pin Buffer 6
11001 (0x19)	DAI_PB07_O	Pin Buffer 7
11010 (0x1A)	DAI_PB08_O	Pin Buffer 8
11011 (0x1B) – 11111 (0x1F)	Reserved	

Destination Control Signal Register (SR_DAT_SHREG)

Figure A-91 shows the programmable options for the SR_SDI_I input signal.

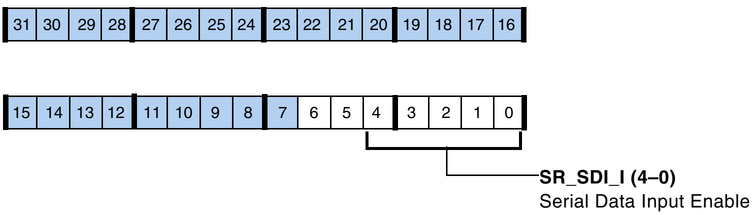

Figure A-91. SR_DAT_SHREG Register (RW)

DAI Pin Buffer Registers (DAI_PIN_STAT)

The register shown in Figure A-92 returns the status of DAI_PB20-1 pin buffers. This register is updated at PCLK/2 rate.

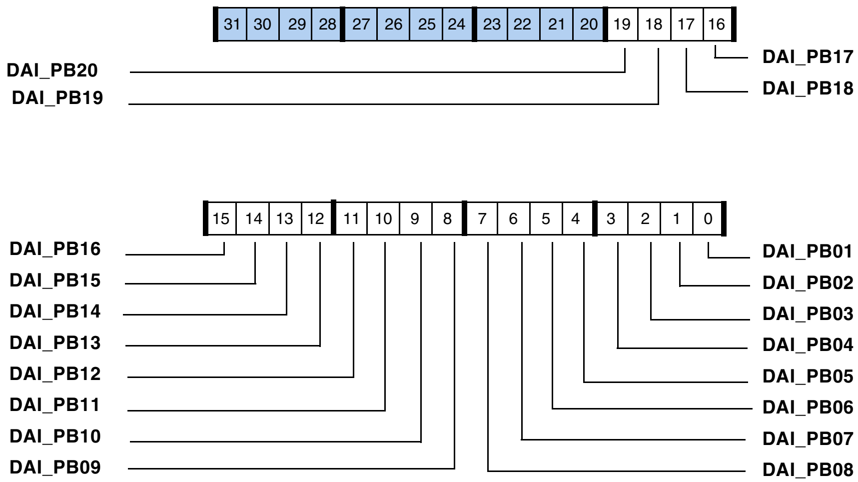

Figure A-92. DAI_PIN_STAT Register (RO)

Peripherals Routed Through the DAI

The following sections provide information on the peripherals that are explicitly routed through the digital applications interface. For more information, see "DAI Signal Routing Unit Registers" on page A-124.

Serial Port Registers

The following section describes serial port (SPORT) registers.

SPORT Divisor Registers (DIVx)

These registers, shown in Figure A-93, allow programs to set the frame sync divisor and clock divisor.

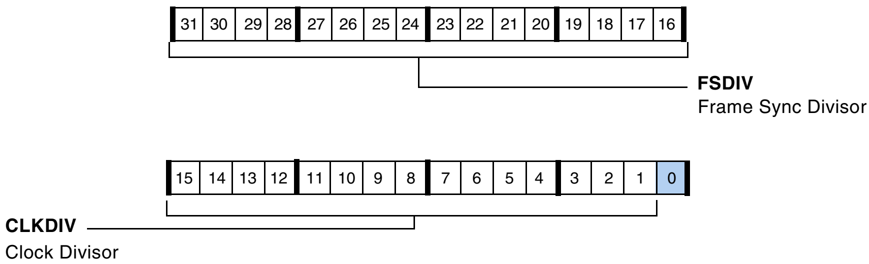

Figure A-93. DIVx Register (RW)

Serial Control Registers (SPCTLx)

The SPCTLx registers (Figure A-94, Figure A-95 and Figure A-96 and Table A-84 on page A-159) are transmit and receive control registers for the corresponding serial ports (SPORT 0 through 7). These registers change depending on operating mode.

For more information, see"Operating Modes" on page 11-29 especially Table 11-9 and Table 11-10.

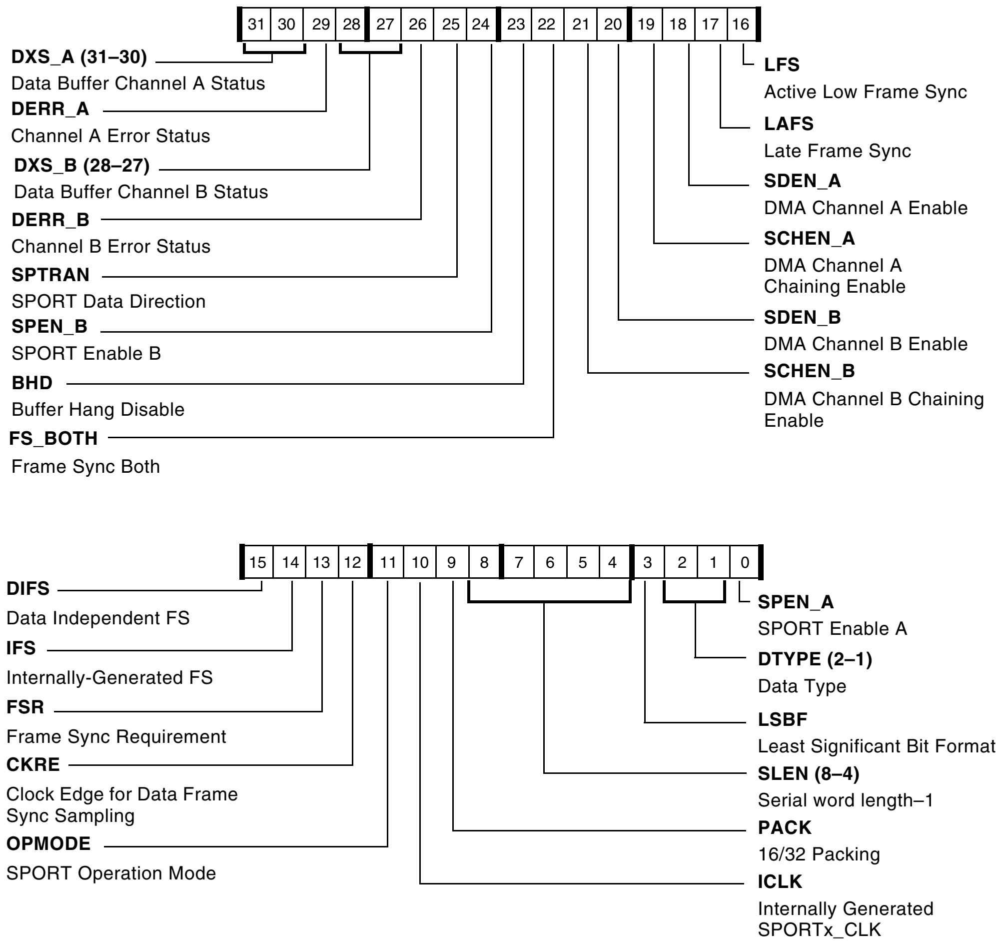

Figure A-94. SPCTLx Register for Standard Serial Mode

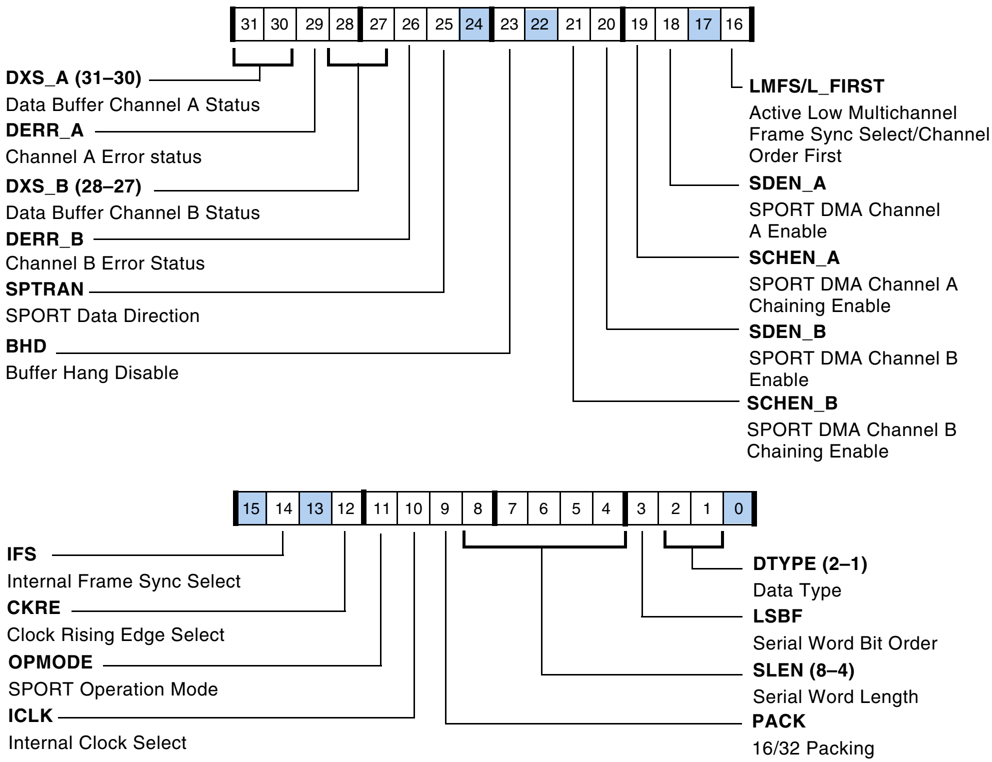

Figure A-95. SPCTLx Register – Packed and Multichannel Mode

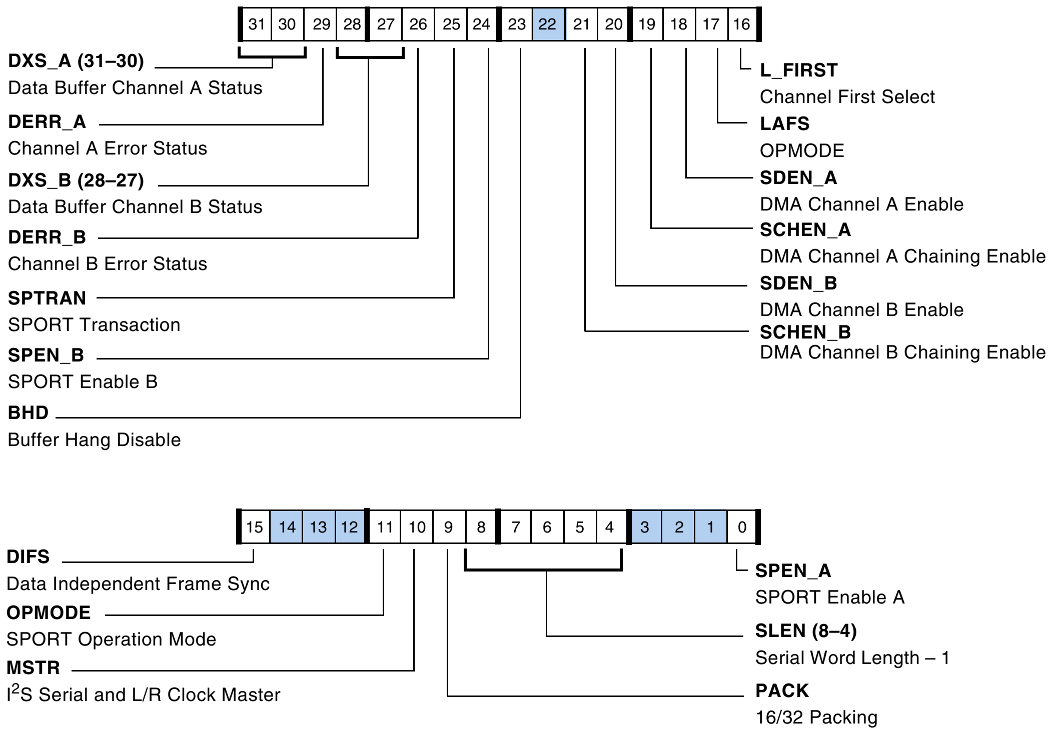

Figure A-96. SPCTLx Register for I^{2}S and Left-Justified Modes

Table A-84. SPCTLx Register Bit Descriptions (All Modes, RW)

Bit	Name	Description
0	SPEN_A	**Enable Channel A Serial Port.** 0 = Serial port A channel disabled 1 = Serial port A channel enabled Note if the bit changes from one (=1) to zero (=0) the data buffers are automatically flushed which takes 6 core cycles. This bit gets cleared if the RW1C error bits in SPERRCTL are cleared.
		This bit is reserved when the SPORT is in packed or multichannel modes.

Table A-84. SPCTLx Register Bit Descriptions (All Modes, RW) (Cont'd)

Bit	Name	Description
2–1	DTYPE	**Data Type Select.** Selects the data type formatting for standard serial mode transmissions. For standard serial mode A channels, selection of companding mode and MSB format are exclusive: 00 = Right-justify, zero-fill unused MSBs 01 = Right-justify, sign-extend unused MSBs 10 = Compand using μ-law 11 = Compand using A-law For standard serial mode B channels: 0 = Right-justify, zero-fill unused MSBs 1 = Right-justify, sign-extend unused MSBs The transmit buffer does not zero-fill or sign-extend transmit data words; this only takes place for the receive buffer.
		For multichannel/packed mode A channels, selection of companding mode and MSB format are inclusive: x0 = Right-justify, zero-fill unused MSBs x1 = Right-justify, sign-extend unused MSBs 1x = Compand using μ-law 1x = Compand using A-law For multichannel/packed mode B channels: 0 = Right-justify, zero-fill unused MSBs 1 = Right-justify, sign-extend unused MSBs The transmit buffer does not zero-fill or sign-extend transmit data words; this only takes place for the receive buffer. For all B channels, companding is not available.
		This bit is reserved when the SPORT is in I^2S or left-justified modes.
3	LSBF	**Serial Word Endian Select.** 0 = Big endian (MSB first) 1 = Little endian (LSB first)
		This bit is internally set when the SPORT is in I^2S or left-justified modes

Table A-84. SPCTLx Register Bit Descriptions (All Modes, RW) (Cont'd)

Bit	Name	Description
8–4	SLEN	**Serial Word Length Select.** Selects the word length in bits where the SLEN field = serial word length –1 For standard/packed and multichannel modes SLEN = 2–31 (3 to 32 bits) For I^2S and left-justified modes SLEN = 7–31 (8 to 32 bits)
9	PACK	**16-Bit to 32-Bit Word Packing Enable.** When PACK = 1, two successive received words are packed into a single 32-bit word, and each 32-bit word is unpacked and transmitted as two 16-bit words. 0 = Disable 16- to 32-bit word packing 1 = Enable 16- to 32-bit word packing
10	ICLK/ MSTR	**Internal Clock Select.** 0 = Select external transmit clock. The clock signal is accepted as an input on the SPORTx_CLK_I signals and the serial clock divisors in the DIVx registers are ignored. The externally-generated serial clock does not need to be synchronous with the processor's system clock. 1 = Select internal transmit clock. The SPORTx_CLK_O signals are outputs and the clock frequency is determined by the value of the serial clock divisor (CLKDIV bit) in the DIVx registers.
		Master Select. For I^2S and left-justified mode, the MSTR bit selects the source for clock and frame sync. 0 = External clock and frame sync 1 = Internal clock and frame sync Note the externally-generated serial clock and FS does not need to be synchronous with the processor's system clock.
11	OPMODE	**SPORT Operation Mode.** 0 = DSP standard /multichannel mode 1 = I^2S, packed, left-justified mode

Table A-84. SPCTLx Register Bit Descriptions (All Modes, RW) (Cont'd)

Bit	Name	Description
12	CKRE/ Reserved	**Clock Edge Select.** Determines the clock signal to sample data and selects the frame sync. For sampling receive data and frame syncs: 1 = Selects the rising edge of SPORTx_CLK. 0 = The processor selects the falling edge of SPORTx_CLK for sampling receive data and frame syncs. Note that transmit data and frame sync signals change their state on the clock edge that is not selected. For example, the transmit and receive functions of any two SPORTs connected together should always select the same value for CKRE so internally-generated signals are driven on one edge and received signals are sampled on the opposite edge.
		This bit is internally set when the SPORT is in I^{2}S or left-justified mode.
13	FSR/ Reserved	**Frame Sync Required Select.** Selects whether the serial port requires (if set, = 1) or does not require a transfer frame sync (if cleared, = 0).
		This bit is internally set when the SPORT is in I^{2}S or left-justified, multichannel or packed mode.
14	IFS/ Reserved	**Internal Frame Sync Select.** Selects whether the serial port uses an internally generated frame sync (if set, = 1) or uses an external frame sync (if cleared, = 0).
		This bit is reserved when the SPORT is in I^{2}S or left-justified mode.
15	DIFS/ Reserved	**Data Independent Frame Sync Select.** 1 = Serial port uses a data-independent frame sync (sync at selected interval) 0 = Serial port uses a data-dependent frame sync (sync when TX FIFO is not empty or when RX FIFO is not full).
		This bit is internally set when the SPORT is in packed or multichannel modes.

Table A-84. SPCTLx Register Bit Descriptions (All Modes, RW) (Cont'd)

Bit	Name	Description
16	LFS/L_FIRST	**Polarity Level Frame Sync.** This bit selects the logic level of the (transmit or receive) frame sync signals for standard and multichannel modes if the FSED bit in SPCTLNx register is cleared (=0). 0 = Active high frame sync 1 = Active low frame sync
		Polarity Edge Frame Sync. This bit selects the logic edge of the (transmit or receive) frame sync signals for multichannel mode if the FSED bit in SPCTLNx register is set (=1). 0 = Rising edge frame sync 1 = Falling edge frame sync
		Channel Order First Select. Selects left/right channel first for Left-justified/(I^2S/packed protocol after frame sync edge. 0 = Left channel first (left justified) 1 = Right channel first (left justified) 0 = Right channel first (I^2S/packed) 1 = Left channel first ((I^2S/packed)
17	LAFS/OPMODE /Reserved	**Late Transmit Frame Sync Select.** This bit selects when to generate the frame sync signal. This bit selects a late frame sync if set (= 1) during the first bit of each data word. This bit selects an early frame sync if cleared (= 0) during the serial clock cycle immediately preceding the first data bit 0 = Early frame sync (FS before first bit) 1 = Late frame sync (FS during first bit)
		OPMODE Protocol (I2S or Left-Justified Protocol Select). 0 = I^2S mode 1 = Left-justified mode This bit is reserved when the SPORT is in packed or multichannel modes.
18	SDEN_A	**Enable Channel A Serial Port DMA.** 0 = Disable serial port channel A DMA 1 = Enable serial port channel A DMA
19	SCHEN_A	**Enable Channel A Serial Port DMA Chaining.** 0 = Disable serial port channel A DMA chaining 1 = Enable serial port channel A DMA chaining

Table A-84. SPCTLx Register Bit Descriptions (All Modes, RW) (Cont'd)

Bit	Name	Description
20	SDEN_B	**SPORT DMA Enable Channel B.** 0 = Disable serial port channel B DMA 1 = Enable serial port channel B DMA
21	SCHEN_B	**SPORT DMA Chaining Channel B Enable.** 0 = Disable serial port channel B DMA chaining 1 = Enable serial port channel B DMA chaining
22	FS_BOTH/ Reserved	**FS Both Enable.** If both channels (A/B) are enabled in standard serial mode: 0 = Issue FS if data is present in either transmit buffer 1 = Issue FS if data is present in both transmit buffers
		This bit is internally cleared when the SPORT is in packed or multichannel modes. For I^2S and Left justified this bit is internally cleared for one enabled channel and set for both enabled channels.
23	BHD	**Buffer Hang Disable.** 0 = Causes the processor core to hang when it attempts to write to a full buffer or read from an empty buffer. 1 = Disables the core-hang and a core read from an empty receive buffer returns previously-read (invalid) data and core writes to a full transmit buffer to overwrite (valid) data that has not yet been transmitted.
24	SPEN_B/ Reserved	**SPORT Channel B Enable.** 0 = Serial port B channel disabled 1 = Serial port B channel enabled Note that if the bit changes from one (=1) to zero (=0) the data buffers are automatically flushed which takes 6 core cycles. This bit gets cleared if the RW1C error bits in SPERRCTL are cleared.
		Reserved when the SPORT is in packed or multichannel modes.

Table A-84. SPCTLx Register Bit Descriptions (All Modes, RW) (Cont'd)

Bit	Name	Description
25	SPTRAN	**Data Transfer Direction.** This bit controls the data direction of the serial port channel A and B signals. 0 = Receive on both channels A and B. The RXSPxA and RXSPxB buffers are activated, while the receive shift registers are controlled by SPORTx_CLK and SPORTx_FS. The TXSPxA and TXSPxB buffers are inactive. 1 = Transmit on both channels A and B. The TXSPxA and TXSPxB buffers are activated, while the transmit shift registers are controlled by SPORTx_CLK and SPORTx_FS. The RXSPxA and RXSPxB buffers are inactive.
26 (RO)	DERR_B	**Channel B Error Status.** This bit provides transmit underflow or receive overflow status. If FSR = 1, the DERR_x bit indicates whether the SPORTx_FS signal (from an internal or external source) occurred while the DXS_x buffer was empty or full. 0 = No SPORTx_FS signal occurred while TXSPxA/B buffer is empty/full. 1 = SPORTx_FS signal occurred while TXSPxA/B buffer is empty/full. This bit gets cleared if the RW1C error channel bit in SPERRCTL is cleared.
28–27 (RO)	DXS_B	**Channel B Data Buffer Status.** Indicates the status of the serial port's channel B data buffer (RXSPxB or TXSPxB) as follows: 00 = Empty 10 = Partially full 11 = Full
29 (RO)	DERR_A	**Channel A Error Status (sticky).** Refer to DERR_B
31–30 (RO)	DXS_A	**Channel A Data Buffer Status.** Refer to DXS_B

SPORT Control 2 Registers (SPCTLNx)

These registers (where x signifies SPORT 0 through 7) allow programs to set frame sync edge detection for I^2S compatibility. These registers also allow interrupts to be generated when transmit DMA count is expired or when the last bit of last word is shifted out.

Note that these registers do not exist on previous SHARC processors (ADSP-212xx, ADSP-213xx).

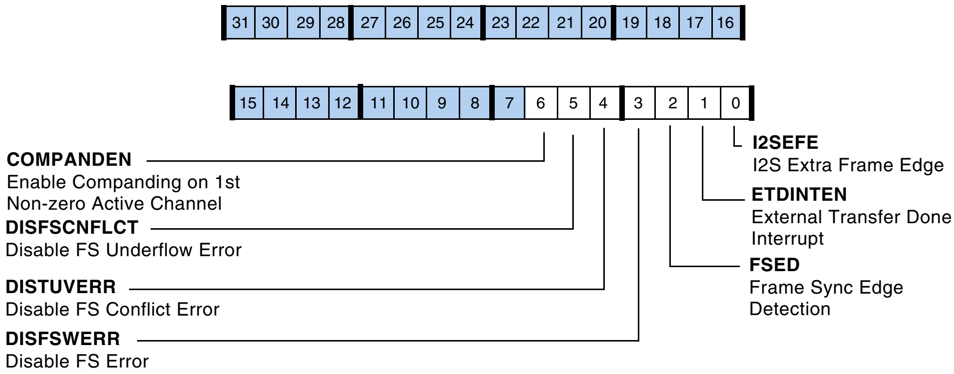

Figure A-97. SPCTLNx Register

Table A-85. SPCTLNx Register Bit Descriptions (RW)

Bit	Name	Description
0	I2SEFE	**I2S Extra Frame Edge.** If set, SPORT generates the last LRCLK if configured as I^2S master (valid only for DMA). If cleared, (reset value), behaves similar to previous SHARCs.
1	ETDINTEN	**External Transfer Done Interrupt.** If set, interrupt occurs only after the last bit of last word in the DMA is shifted out. If cleared, interrupt occurs when the DMA counter expires. For chain pointer DMA, if set, interrupt occurs: 1) If PCI=0 only after that last word of the last DMA block in the chain is shifted out. 2) If PCI=1, when DMA counter expires for the initial DMA blocks (CP is nonzero) in the chain and the last bit of the last word is shifted out for the last DMA block in the chain. For receive DMA, interrupt behaves in the same way independent of the bit setting.

Table A-85. SPCTLNx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
2	FSED	**Frame Sync Edge Detection.** In multichannel mode: 1 = Start transmitting/receiving only after the SPORTs detect an active edge of an external frame sync (even if the SPORTs are enabled at any instant of an active frame sync). This is done only when SPORTs are programmed for external FS mode (IFS = 0). 0 = Behaves similar to previous SHARCs (default).
3	DISFSWERR (Applies to ADSP-2147x, ADSP-2148x)	**Disable Frame Sync Error.** Serial Mode—If an external frame sync is of more than one SCLK width, a frame sync error is generated. If this bit is set, the frame sync error is generated only on an active edge of premature frame sync during valid data transmission/reception. Late Frame Sync Mode—If a frame sync is not active during the whole transmission/reception a frame sync error is generated. An error is not generated even if the frame sync is of more than one SCLK width or if it is not active throughout the transmit/receive.
4	DISTUVERR (Applies to ADSP-2147x, ADSP-2148x)	**Disable Underflow Error.** If single channel is enabled in multichannel mode, and if a premature frame sync occurs (for example: word length = 16 bits, frame sync duration 14 SCLK) during the transmission of last word in a DMA, then the TX underflow error bit is set (DERR_x bit), even though this premature frame sync does not cause any underflow and the SPORT does not try to drive data for this premature frame sync. If this bit is set, no spurious TX underflow error is generated for this premature frame sync.

Table A-85. SPCTLNx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
5	DISFSCNFLCT (Applies to ADSP-2147x, ADSP-2148x)	**Disable Frame Sync Conflict Error.** If single channel is enabled in multichannel mode, and if the frame sync duration is one less than the word length (example word length is 16 bits and frame sync duration is 15 SCLK cycles), then every second frame sync should be taken as an invalid frame sync and no data should be transmitted/received for that frame sync. However data is also transmitted/received for premature frame syncs. If this bit is set, no data is transmitted/received for premature frame syncs.
6	COMPANDEN (Applies to ADSP-2147x, ADSP-2148x)	**Companding on First Active Channel Enable.** If companding for any active channel is enabled in multichannel mode, and the first active channel is not the zeroth channel, and companding is enabled for the first active channel (for example channel 2), then from second frame onwards, companding for the first active channel (channel 2) does not occur. If this bit is set, companding occurs for the first active channel, even if it is not the zeroth channel. For more information, see "Companding Limitations (ADSP-2146x)" on page 11-26.

SPORT Multichannel Control Registers (SPMCTLx)

The serial ports in the ADSP-214xx processors work individually, not in pairs. Therefore, each SPORT has its own multichannel control register. These registers are shown in Figure A-98 (where x = SPORTs 0, 2, 4, and 6 and y = SPORTs 1, 3, 5, and 7) and described in Table A-86.

Note that in ADSP-2136x SHARC processors there is one `SPMCTLxy` register for each TDM pair, therefore programs can write to one register for both SPORTs.

On the ADSP-214xx SHARC processors, each sport has its own `SPMCTLx` register, so only one write into both `SPMCTLx` registers is required to operate the SPORTs as pairs. Since there is no change in `SPMCTLx` register bit definitions, the same value can be written into both `SPMCTLx` registers in order to make legacy programs for the ADSP-2136x processors operate correctly.

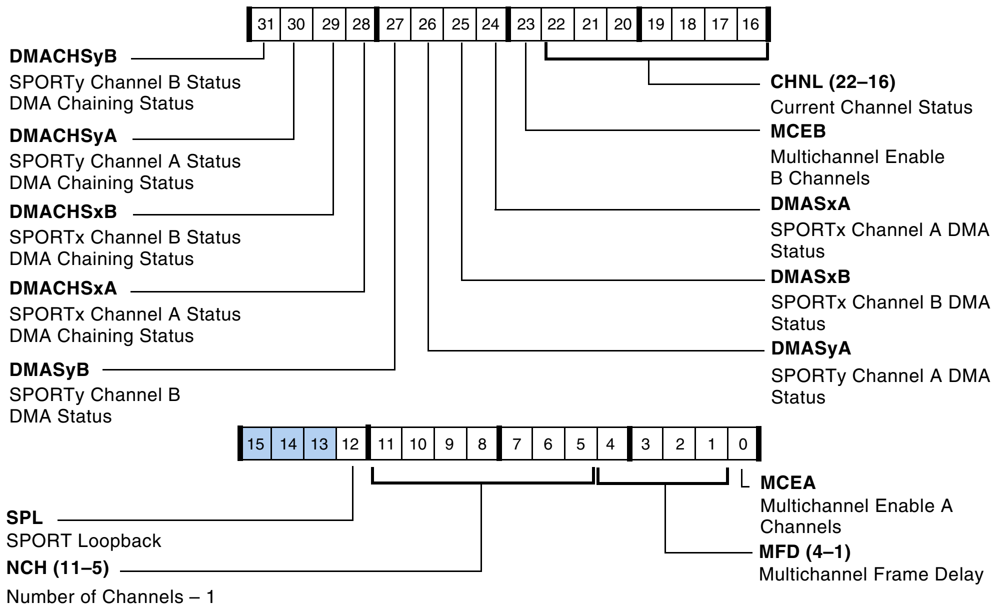

Figure A-98. SPMCTLx Registers – Multichannel/Packed Mode

Table A-86. SPMCTLx Register Bit Descriptions (RW)

Bit	Name	Description
0	MCEA	**Multichannel Mode Enable, A Channels.** Packed and multichannel A modes only. One of two configuration bits that enable and disable multichannel mode on serial port channels. See OPMODE bit (17). 0 = Disable multichannel A operation 1 = Enable multichannel A operation/packed mode. The corresponding SPEN_A bit in the SPCTL register should be cleared. If the bit transitions from high to low, the buffer (TDM/packed) is cleared which takes 6 CCLK cycles. The DERR_A bit is also cleared.
4–1	MFD	**Multichannel Frame Delay.** The interval, in number of serial clock cycles, between the multichannel frame sync pulse and the first data bit. These bits provide support for different types of T1 interface devices. Valid values range from 0 to 15 with bits 4–1. Values of 1 to15 correspond to the number of intervening serial clock cycles. A value of 0 corresponds to no delay. The multichannel frame sync pulse is concurrent with first data bit.

Table A-86. SPMCTLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
11–5	NCH	**Number of Multichannel Slots (minus one).** Select the number of channel slots (maximum of 128) to use for multichannel operation. Valid values for actual number of channel slots range from 1 to 128. Use this formula to calculate the value for NCH: NCH = Actual number of channel slots – 1.
12	SPL	**SPORT Loopback Mode**. Enables if set (= 1) or disables if cleared (= 0) the channel loopback mode. Loopback mode enables debug capabilities. Loopback works under the configurations show in "Loopback Routing" on page 10-40 where either of the two paired SPORTs can be set up to transmit or receive, depending on their SPTRAN bit setting. Only the transmitter acts as master to generate the clock and frame sync. The SPL bit applies to all non multichannel modes.
15–13	Reserved	
22–16 (RO)	CHNL	**Current Channel Selected.** Identify the currently selected transmit channel slot (0 to 127).
23	MCEB	**Multichannel B Mode Enable.** Packed and multichannel B modes only. One of two configuration bits that enable and disable multichannel mode on serial port channels. See OPMODE bit (17). 0 = Disable multichannel B operation 1 = Enable multichannel B operation/packed mode the corresponding SPEN_B bit in the SPCTL register should be cleared. If the bit transitions from high to low, the buffer (TDM/packed) is cleared which takes 6 core clock cycles. The DERR_B bit is also cleared.
24 (RO)	DMASxA	**DMAxA DMA Channel Status.** 0 = Inactive 1 = Active
25 (RO)	DMASxB	**DMAxB DMA Channel Status.** 0 = Inactive 1 = Active
26 (RO)	DMASyA	**DMAyA DMA Channel Status.** 0 = Inactive 1 = Active
27 (RO)	DMASyB	**DMAyB DMA Channel Status.** 0 = Inactive 1 = Active

Table A-86. SPMCTLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
28 (RO)	DMACHSxA	**DMAxA DMA Chaining Status.** 0 = Inactive 1 = Active
29 (RO)	DMACHSxB	**DMAxB DMA Chaining Status.** 0 = Inactive 1 = Active
30 (RO)	DMACHSyA	**DMAyA DMA Chaining Status.** 0 = Inactive 1 = Active
31 (RO)	DMACHSyB	**DMAyB DMA Chaining Status.** 0 = Inactive 1 = Active

SPORT Active Channel Select Registers (SPxCSy)

Each bit, 31–0, set (=1) in one of the four `SPxCS3-0` registers corresponds to the active channel, 127–0, on a multichannel mode serial port. When these registers activate a channel (by setting the respective bits in these registers to 1, the serial port transmits or receives the word in that channel's position of the data stream. When a channel's bit in these registers is cleared (=0), the serial port's data transmit pin three-states during the channel's transmit time slot if the serial port is configured as transmitter. If the serial port is configured as the receiver it ignores the incoming data.

SPORT Compand Registers (SPxCCSy)

Each bit, 31–0, set (=1) in one of the four `SPxCCS3-0` registers corresponds to the active companding channel, 127–0, on a multichannel mode serial port. Only SPORT0/2/4/6/A supports transmit directions and SPORT1/3/5/7/A supports receive directions. When these registers activate companding for a channel, the SPORT applies the companding from the serial port's `DTYPE` selection to the word transmitted or received in that channel's position of the data stream. When a channel's bit in these registers is cleared (=0), the SPORT does not compand the outgoing or incoming data during the channel's time slot.

Error Control Register (SPERRCTLx)

The SPERRCTLx registers control and report the status of the interrupts generated by each SPORT (see Figure A-99, Table A-87).

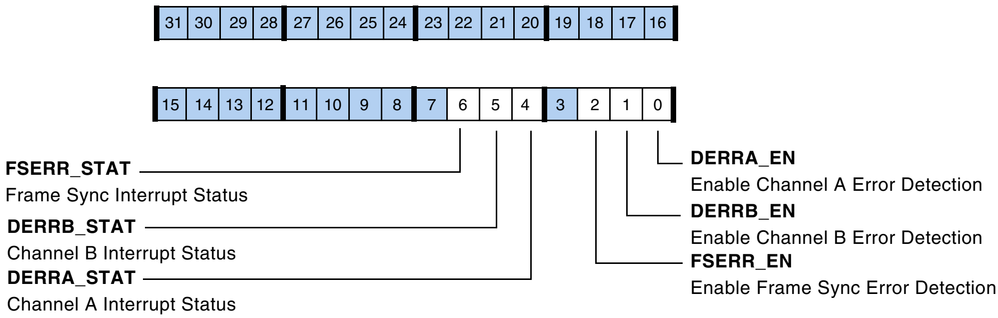

Figure A-99. SPERRCTLx Register

Table A-87. SPERRCTLx Register Bit Descriptions (RW)

Bit	Name	Description
0	DERRA_EN	**Enable Channel A Error Detection.** 0 = Disable 1 = Enable
1	DERRB_EN	**Enable Channel B Error Detection.** 0 = Disable 1 = Enable
2	FSERR_EN	**Enable Frame Sync Error Detection.** 0 = Disable 1 = Enable
3	Reserved	
4 (RW1C)	DERRA_STAT	**Channel A Interrupt Status.** SPTRAN = 0 Receive overflow status SPTRAN = 1 Transmit underflow status

Table A-87. SPERRCTLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
5 (RW1C)	DERRB_STAT	**Channel B Interrupt Status.** SPTRAN = 0 Receive overflow status SPTRAN = 1 Transmit underflow status
6 (RW1C)	FSERR_STAT	**Frame Sync Interrupt Status.** 0 = No frame sync error 1 = Frame sync error detected

SPORT Error Status Register (SPERRSTAT)

The SPERRSTAT register combines the status of all SPORT interrupts (see Figure A-100).

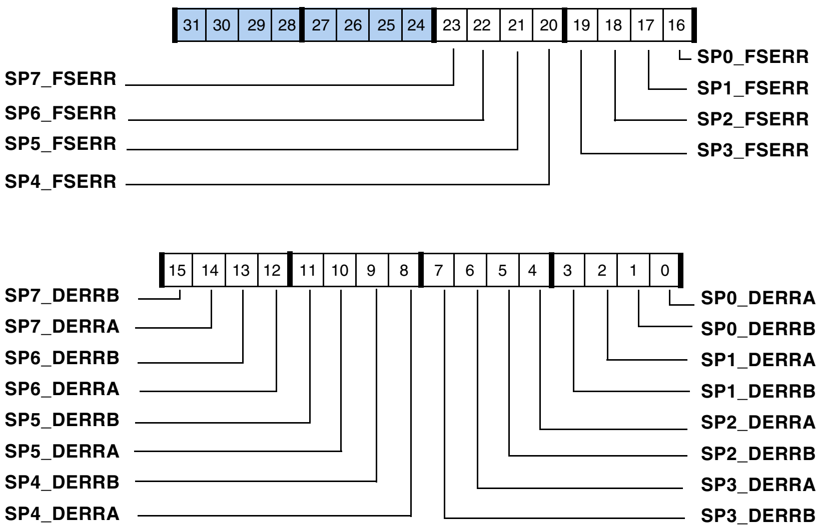

Figure A-100. SPERRSTAT Register (RO)

Input Data Port Registers

The input data port (IDP) provides an additional input path to the processor core. The IDP can be configured as 8 channels of serial data or 7 channels of serial data and a single channel of up to a 20-bit wide parallel data.

Input Data Port Control Register 0 (IDP_CTL0)

Use this register to configure and enable the IDP and each of its channels. The register is shown in Figure A-101 and described in Table A-88.

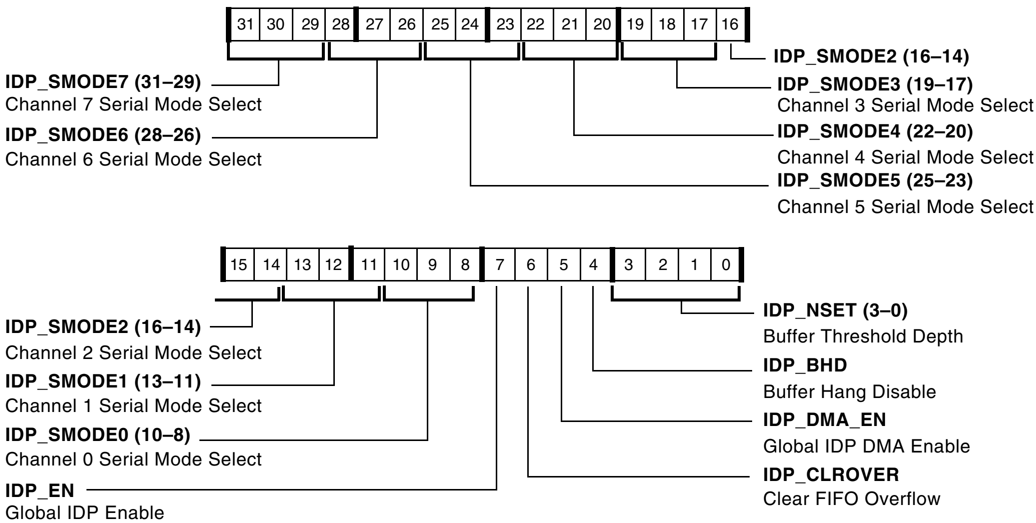

Figure A-101. IDP_CTL0 Register

Table A-88. IDP_CTL0 Register Bit Descriptions (RW)

Bits	Name	Description
3–0	IDP_NSET	**Buffer Threshold Depth.** The setting of these bits represent a threshold number of entries (N) in the FIFO. When the FIFO fills to a point where it has greater (N+1) than N words (data in FIFO exceeds the value set in the IDP_NSET bit field), a DAI interrupt is latched with the IDP_FIFO_GTN_INT bit in the DAI_IRPTL_x register. Only the core can use this feature to detect when data needs to be read. The maximum IDP_NSET= 7, otherwise no interrupt is generated.
4	IDP_BHD	**IDP Buffer Hang Disable.** Reads of an empty FIFO or writes to a full FIFO to cause a core hang. This condition continues until the FIFO has valid data (in the case of reads) or the FIFO has at least one empty location (in the case of writes). This can be used in debug operations. 0 = Core hang is enabled 1 = Core hang is disabled
5	IDP_DMA_EN	**DMA Enable.** Enables DMA on all IDP channels. This bit is the global control for standard and ping-pong DMA. 0 = Channel disabled 1 = Channel enabled
6 (RW1S)	IDP_CLROVR	**FIFO Overflow Clear Bit.** Clears the FIFO and the SRU_OVFx bits in the DAI_STAT register.
7	IDP_EN	**Enable IDP.** This bit enables the IDP. This is a global control bit. This bit needs to be set for all operations modes including DMA. When this bit transitions from high to low the buffer is flushed which takes 2 core clock cycles. Writing a 1 to bit 31 of the IDP_CTL1 register also flushes the FIFO. To enable the IDP for DMA, two separate bits in two different registers must be set. The first are the global IDP_EN and IDP_DMA_EN bits in the IDP_CTL0 register and the second are the specific channel enable bits, located in the IDP_CTL1 register.

Table A-88. IDP_CTL0 Register Bit Descriptions (RW) (Cont'd)

Bits	Name	Description
10–8	IDP_SMODE0	**Serial Input Data Format Mode Select.** These eight inputs (0–7), each of which contains 3 bits, indicate the mode of the serial input for each of the eight IDP channels. Input format: 000 = Left-justified 24 bits 001 = I^2S mode 24 bits 010 = Left-justified 32 bits 011 = I^2S 32 bits 100 = Right-justified 24 bits 101 = Right-justified 20 bits 110 = Right-justified 18 bits 111 = Right-justified 16 bits Note the SMODEx bits define the IDP buffer input format for core access. For I2S and left-justified single channel modes, it receives 32 bits of data from the SDATA pins. No L/R bit is added in these modes.
13–11	IDP_SMODE1	
16–14	IDP_SMODE2	
19–17	IDP_SMODE3	
22–20	IDP_SMODE4	
25–23	IDP_SMODE5	
28–26	IDP_SMODE6	
31–29	IDP_SMODE7	

Input Data Port Control Register 1 (IDP_CTL1)

Use the `IDP_CTL1` register to configure and enable individual IDP channels. The register is shown in Figure A-102 and described in Table A-89.

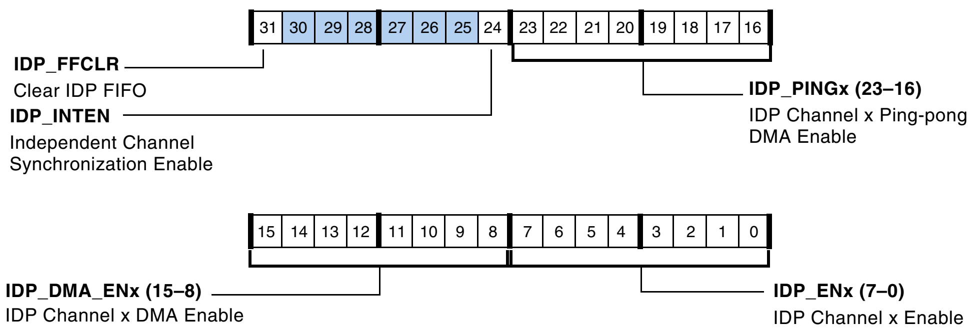

Figure A-102. IDP_CTL1 Register

Table A-89. IDP_CTL1 Register Bit Descriptions (RW)

Bit	Name	Description
7–0	IDP_ENx	**IDP Channel x Enable.** These are the enable bits for accepting data from individual channels. Corresponding IDP_ENx must be set with IDP_EN bit to get data from channel x. If IDP_EN bit is not set then this bit has no effect. 0x00 = all channels cleared 0xFF = all channels enabled (default)
15–8	IDP_DMA_ENx	**IDP DMA Enable.** These are the DMA enable bits for individual channels. Corresponding IPD_DMA_ENx must be set with IDP_DMA_EN bit for DMA transfer of data from channel x. If the global DMA_EN bit is not set then this bit has no effect. 0x00 = all channels cleared 0xFF = all channels enabled (default)
23–16	IDP_PINGx	**IDP Ping-Pong DMA Channel x Enable.** These are the Ping-Pong DMA enable bits for individual channels. Corresponding IDP_PINGx must be set to start ping-pong DMA from channel x. This bit requires the IDP_DMA_ENx bit and IDP_DMA_EN bit are set.
24	IDP_INTEN	**Independent Channel Synchronization Enable.** This is the enable bit for independent channel synchronization. If this bit is set, the IDP channels will start shifting in data from the first active edge of the LRCLK based on the setting of FAEx. If this bit is cleared (reset value), then the ADSP-214xx behaves like previous SHARC processors.
30–25	Reserved	
31 (RW1S)	IDP_FFCLR	**Clear IDP FIFO.** Setting this bit to 1 clears the IDP FIFO and the IDP_FIFOSZ bits. This bit can be set together with the enable bit.

Input Data Port Control Register 2 (IDP_CTL2)

This register controls the first active edge selection for channel synchronization. The register is shown in Figure A-103 and described in Table A-90.

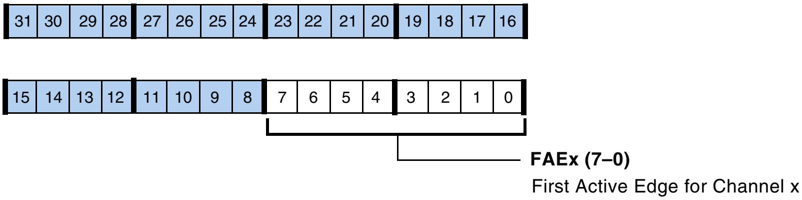

Figure A-103. IDP_CTL2 Register

Table A-90. IDP_CTL2 Register Bit Descriptions (RW)

Bit	Name	Description
7–0	FAEx	**First Active Edge for Channel x.** 1= nth IDP channel starts shifting in data from the first rising edge of LRCLK after IDP is enabled. This data is latched after the next falling edge of LRCLK. 0 = nth IDP channel starts shifting in data from the first falling edge of LRCLK after IDP is enabled. This data is latched after the next rising edge of LRCLK. Reset value of all these bits is 0. These bits are used only if IDP_INTEN bit (IDP_CTL1[24]) is set.
8–31	Reserved	

Parallel Data Acquisition Port Control Register (IDP_PP_CTL)

The `IDP_PP_CTL` register (shown in Figure A-104 and described in Table A-91) provides 20 mask bits that allow the input from any of the 20 pins to be ignored.

For more information on the operation of the parallel data acquisition port, see Chapter 12, Input Data Port (SIP, PDAP). For information on the pin multiplexing that is used in conjunction with this module, see "Pin Multiplexing" on page 24-28.

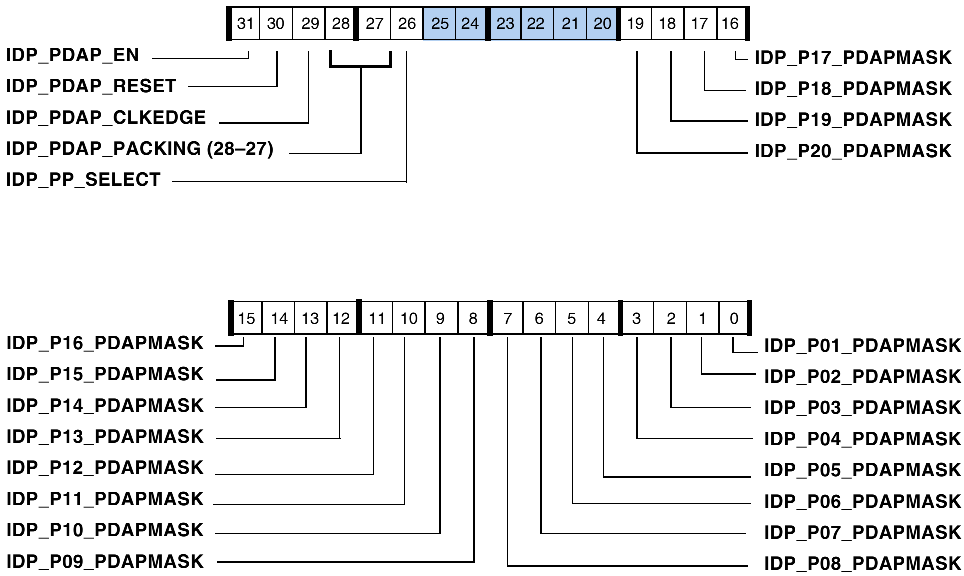

Figure A-104. IDP_PP_CTL Register

Table A-91. IDP_PP_CTL Register Bit Descriptions (RW)

Bit	Name	Description
19–0	IDP_P20–1_ PDAPMASK	**Parallel Data Acquisition Port Mask.** For each of the parallel inputs: 0 = Input data from PDAP_20-1 are masked 1 = Input data from PDAP_20-1 are unmasked After this masking process, data gets passed along to the packing unit.
25–20	Reserved	
26	IDP_PP_ SELECT	**PDAP Port Select.** This bit selects which peripheral is connected to the PDAP unit. 0 = Data/control bits are read from DAI pins 1 = Data/control bits are read from AMI_ADDR pins

Table A-91. IDP_PP_CTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
28–27	IDP_PDAP_ PACKING	**Packing.** Selects PDAP packing mode. These bits mask parallel sub words from the 20 parallel input signals and packs them into a 32-bit word. The bit field indicates how data is packed. Selection of packing mode is made based on the application. 00 = 8- to 32-bit (packing by 4) 01 = (11, 11, 10) to 32-bit (packing by 3) 10 = 16- to 32-bit (packing by 2) 11 = 20- to 32-bit (no packing). 12 LSBs are cleared For input data width less than 20-bits, inputs are aligned to MSB pins.
29	IDP_PDAP_ CLKEDGE	**PDAP Sampling Clock Edge Select.** Setting this bit (= 1) causes the data to latch on the falling edge (PDAP_CLK_I signal). Clearing this bit (= 0) causes data to latch on the rising edge (default). Notice that in all four packing modes, data is read on a clock edge, but the specific edge used (rising or falling) is not indicated. 0 = Data is latched on the rising edge 1 = Data is latched on the falling edge
30 (RW1S)	IDP_PDAP_ RESET	**PDAP Reset.** A reset clears any data in the packing unit waiting to get latched into the IDP FIFO. This bit resets the counter of the PDAP for packing alignment. This bit always returns a value of zero when read.
31	IDP_PDAP_ EN	**PDAP Enable.** 0 = Disconnects all PDAP inputs (data/control) from use as parallel input channel 1 = Connects all PDAP inputs (data/control) from use as parallel input channel. IDP channel 0 cannot be used as a serial input port with this setting

IDP Status Register (DAI_STAT0)

The IDP DMA status register shown in Figure A-105 and described in Table A-92 reflects the status of the standard and ping-pong DMA channels.

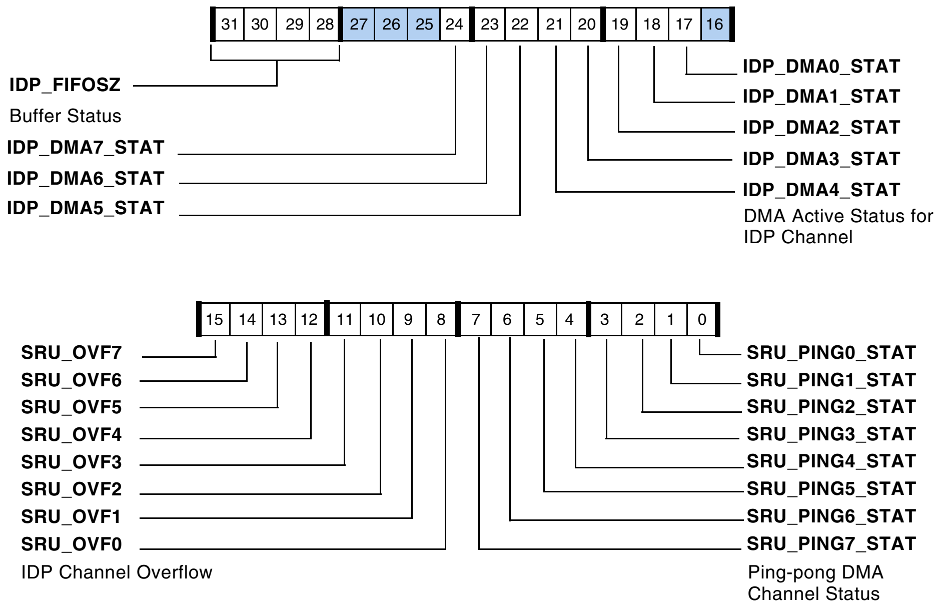

Figure A-105. DAI_STAT0 Register

Table A-92. DAI_STAT0 Register Bit Descriptions (RO)

Bit	Name	Description
7–0	SRU_PINGx_ STAT	**Ping-Pong DMA Channel A/B Status.** Indicates the status of ping-pong DMA in each respective channel (7–0). 0 = Ping DMA (channel A) is active 1 = Pong DMA (channel B) is active
15–8	SRU_OVFx	**Overflow Channel Status (sticky).** Provides overflow status information for each channel (bit 8 for channel 0 through bit 15 for channel 7). 0 = IDP channel input no overflow 1 = IDP channel input overflow has occurred These bits get cleared by reading the DAI_IRPTL register
16	Reserved	

Table A-92. DAI_STAT0 Register Bit Descriptions (RO) (Cont'd)

Bit	Name	Description
24–17	IDP_DMAx_STAT	**Input Data Port DMA Channel Status.** These bits reflect the state of all eight DMA channels and are set once IDP_DMA_EN is set and remain set until the last data of that channel is transferred. Even if IDP_DMA_EN is set (=1), this bit goes low once the required number of data transfers occur. Note that if DMA through some channel is not intended, this bit goes high. 0 = DMA is not active 1 = DMA is active
27–25	Reserved	
31–28	IDP_FIFOSZ	**IDP FIFO Size Status.** Indicates valid number of samples in the IDP FIFO. 0000 = IDP FIFO empty 1000 = IDP FIFO full

IDP Status Register 1 (DAI_STAT1)

Since the core allows writes to the `IDP_FIFO`, the `DAI_STAT1` register stores the different read or writes indexes with a maximum of 8 entries each.

Table A-93. DAI_STAT1 Register Bit Descriptions (RO)

Bit	Name	Description
3–0	FIFO_WRI	**Write Index Pointer.** Reflects the write index status during core writes to the IDP_FIFO. 0000 = No write done 1000 = 8 writes done
7–4	FIFO_RDI	**Read Index Pointer.** Reflects the read index status during core reads from the IDP_FIFO. 0000 = No read done 1000 = 8 reads done
31–8	Reserved	

Asynchronous Sample Rate Converter Registers

The asynchronous sample rate converter (ASRC) is composed of five registers which are described in the following sections.

Control Registers (SRCCTLx)

The `SRCCTLn` control registers (read/write) control the operating modes, filters, and data formats used in the sample rate converter. For n = 0, the register controls the SRC0 and SRC1 modules and for n = 1 it controls the SRC2 and SRC3 modules (x = 0, 2 and y = 1, 3). The bit settings for these registers are shown in Figure A-106 and described in Table A-94.

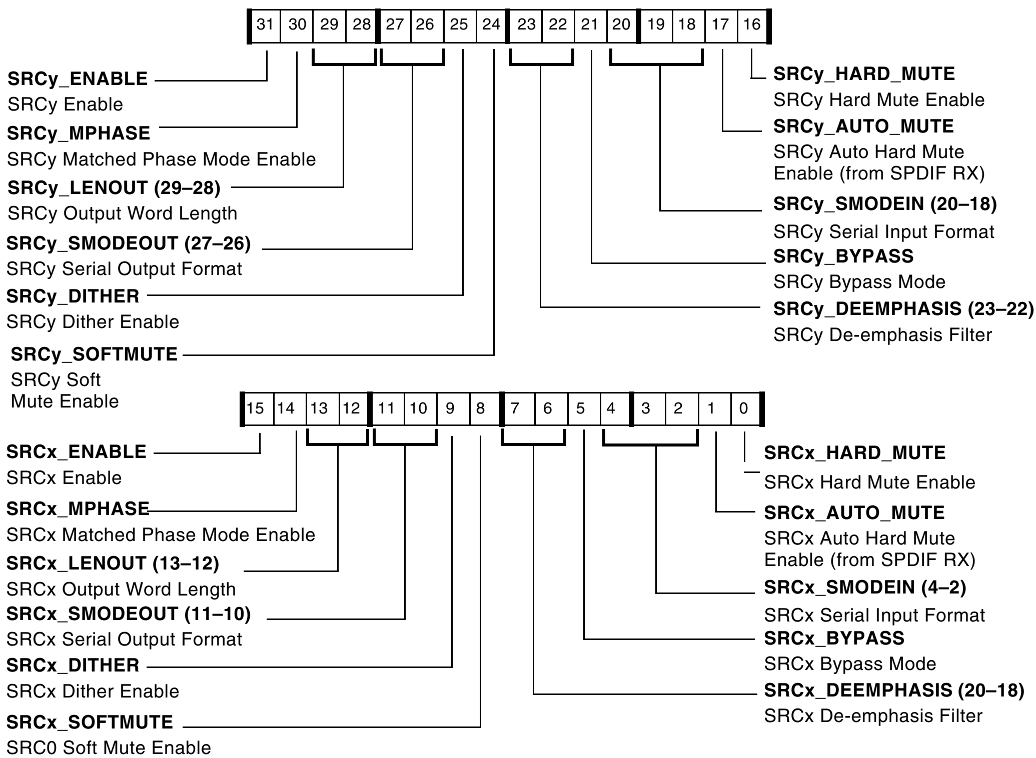

Figure A-106. SRCCTLx Register

Table A-94. SRCCTLx Register Bit Descriptions (RW)

Bit	Name	Description
0	SRCx_HARD_MUTE	**Hard Mute.** Hard mutes SRC 0, 2. 1 = Mute (default)
1	SRCx_AUTO_MUTE	**Auto Hard Mute.** Auto hard mutes SRC 0, 2 when non audio is asserted by the SPDIF receiver. 0 = No mute 1 = Mute (default)
2–4	SRCx_SMODEIN	**Serial Input Format.** Selects the serial input format for SRC 0, 2 as follows: 000 = Default, format is left-justified 001 = I^2S 010 = TDM 100 = 24-bit right-justified 101 = 20-bit right-justified 110 = 18-bit right-justified 111 = 16-bit right-justified
5	SRCx_BYPASS	**Bypass SRCx.** Output of SRC 0, 2 is the same as the input.
6–7	SRCx_DEEMPHASIS	**De-emphasis Filter Select.** Used to de-emphasize audio data that has been emphasized. The type of de-emphasis filter is selected by the SRCx_DEEMPHASIS bits and is based on the input sample rate (SRCx_FS_IP_I signal) as follows: enables de-emphasis on incoming audio data for SRC 0, 2. 00 = No de-emphasis 01 = 32 kHz 10 = 44.1 kHz 11 = 48 kHz
8	SRCx_SOFTMUTE	**Soft Mute.** Enables soft mute on SRC 0, 2. 0 = No mute 1 = Mute (default)
9	SRCx_DITHER	**Dither Enable.** Enables dithering before truncation on SRC 0, 2 when a word length less than 24 bits is selected. 0 = Truncation only 1 = Dithering before truncation

Table A-94. SRCCTLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
10–11	SRCx_SMODEOUT	**Serial Output Format.** Selects the serial output format on SRC 0, 2 as follows: 00 = Left-justified 01 = I^2S 10 = TDM mode 11 = Right-justified. The right-justified serial data out mode assumes 64 SCLK cycles per frame, divided evenly for left and right. For the other modes these 8 LSBs contain zeros.
12–13	SRCx_LENOUT	**Output Word Length Select.** Selects the serial output word length on SRC 0, 2 as follows: 00 = 24 bits 01 = 20 bits 10 = 18 bits 11 = 16 bits Any word length less than 24 bits has dither added to the unused LSBs if SRCx_DITHER is enabled (= 1).
14	MPHASE	**SRCx_MPHASE Match Phase Mode Select.** Configures the SRC 0, 2 modules to not use their own internally-generated sample rate ratio but use an externally-generated ratio. Used with TDM data. (ADSP-21488 only) 0 = Matched phase slave disabled 1 = Matched phase slave enabled Note this setting must be cleared for the phase master.
15	SRCx_ENABLE	**SRCx Enable.** Enables SRC 0, 2. When (set = 1), or when the sample rate (frame sync) between the input and output changes, the SRC begins its initialization routine where; 1) MUTE_OUT is asserted, 2) soft mute control counter for input samples is set to maximum attenuation (–144 dB). Note that SRC power-up completion is finished by clearing the SRCx_MUTEOUT bit in SRCRATx register. Writes to the SRCCTLx register should be at least one cycle before setting the SRCx_ENABLE. When setting and clearing this bit, it should be held low for a minimum of 5 PCLK cycles. Programs should disable the SRC when changing modes.

Table A-94. SRCCTLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
16	SRCy_HARD_MUTE	**Hard Mute.** Hard mutes SRC 1, 3. 1 = Mute (default)
17	SRCy_AUTO_MUTE	**Auto Hard Mute.** Auto hard mutes SRC 1, 3 when non audio is asserted by the SPDIF receiver. 0 = No mute 1 = Mute (default)
18–20	SRCy_SMODEIN	**Serial Input Format.** Selects the serial input format for SRC 1, 3 as follows: 000 = Default, format is left-justified 001 = I^2S 010 = TDM 100 = 24-bit right-justified 101 = 20-bit right-justified 110 = 18-bit right-justified 111 = 16-bit right-justified
21	SRCy_BYPASS	**Bypass Mode Enable.** Output of SRC 1, 3 is the same as input.
22–23	SRCy_DEEMPHASIS	**De-emphasis Filter Select.** Enables de-emphasis on incoming audio data for SRC 1, 3. 00 = No de-emphasis 01 = 32 kHz 10 = 44.1 kHz 11 = 48 kHz
24	SRCy_SOFTMUTE	**Soft Mute.** Enables soft mute on SRC 1, 3. 0 = No mute 1 = Mute (default)
25	SRCy_DITHER	**Dither Enable.** Enables dithering before truncation on SRC 0, 2 when a word length less than 24 bits is selected. 0 = Truncation only 1 = Dithering before truncation
26–27	SRCy_SMODEOUT	**Serial Output Format.** Selects the serial output format for SRC 1, 3 as follows. 00 = Left-justified 01 = I^2S 10 = TDM mode 11 = Right-justified

Table A-94. SRCCTLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
28–29	SRCy_LENOUT	**Output Word Length Select.** Selects the serial output word length for SRC 1, 3 as follows: 00 = 24 bits 01 = 20 bits 10 = 18 bits 11 = 16 bits Any word length less than 24 bits has dither added to the unused LSBs if SRCx_DITHER is enabled (= 0).
30	MPHASE	**SRCy_MPHASE Match Phase Mode Select.** Configures the SRC 0, 2 modules to not use their own internally-generated sample rate ratio but use an externally-generated ratio. Used with TDM data. (ADSP-21488 only) 0 = Matched phase slave disabled 1 = Matched phase slave enabled Note this setting must be cleared for the phase master.
31	SRCy_ENABLE	**SRCy Enable.** Enables SRC 1, 3. When set (= 1), or when the sample rate (frame sync) between the input and output changes, the SRC begins its initialization routine where; 1) MUTE_OUT is asserted, 2) soft mute control counter for input samples is set to maximum attenuation (–144 dB). Note that SRC power-up completion is finished by clearing the SRCx_MUTEOUT bit in SRCRATx register. Writes to the SRCCTLx register should be at least one cycle before setting the SRCx_ENABLE. When setting and clearing this bit, it should be held low for a minimum of 5 PCLK cycles. Programs should disable the SRC when changing modes.

Mute Register (SRCMUTE)

This register connects an SRCx mute input and output when the `SRC0_MUTE_ENx` bit is cleared (=0). This allows SRCx to automatically mute input while the SRC is initializing (0 = automatic muting and 1 = manual muting). Bit 0 controls SRC0, bit 1 controls SRC1, bit 2 controls SRC2, and bit 3 controls SRC3.

Ratio Registers (SRCRATx)

These registers report the mute and I/O sample ratio as follows: the SRCRAT0 register reports for SRC0 and SRC1 and the SRCRAT1 register reports the mute and I/O sample ratio for SRC2 and SRC3 (X = SRC0 and SRC1, Y = SRC2 and SRC3). The registers are shown in Figure A-107 and Figure A-108 and described in Table A-95.

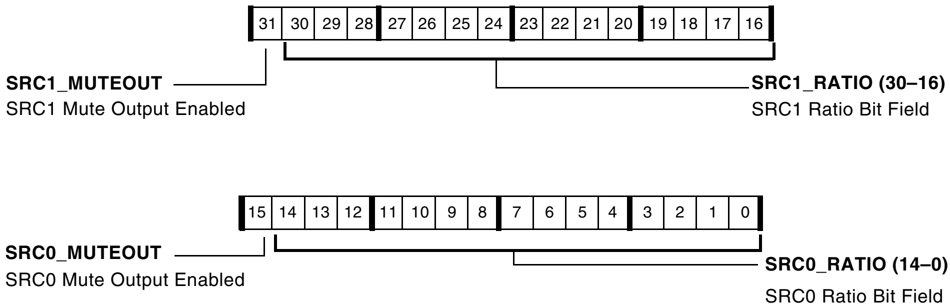

Figure A-107. SRCRAT0 Register

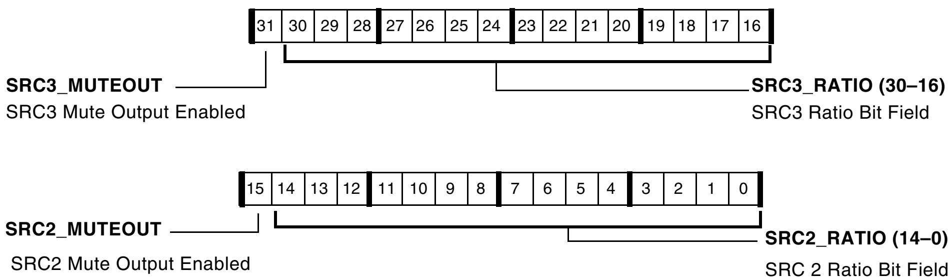

Figure A-108. SRC SRCRAT1 Register

Table A-95. SRCRATx Register Bit Descriptions (RO)

Bit	Name	Description
14–0	SRCx_RATIO	**Sampling Ratio of Frame Syncs.** These bits can be read to find the ratio of output to input sampling frequency (SRCx_FS_OP_I/SRCx_FS_IP_I). This ratio is reported in 4.11 (integer.fraction) format where the 15-bit value of the normal binary number is comprised of 4 bits for the integer and 11 bits for the fraction.
15	SRCx_MUTEOUT	**Mute Status.** The SRCx_MUTEOUT bits in SRCRATx register report the status of the MUTE_OUT signal. Once the SRCx_MUTEOUT signal is cleared, the ratio can be read. When the SRCx_ENABLE is set or there is a change in the sample ratio, the MUTE_OUT signal is asserted. The MUTE_OUT signal remains asserted until the digital servo loop's internal fast settling mode is complete. When the digital servo loop has switched to slow settling mode, the MUTE_OUT signal is deasserted. Reset = 0x80008000.
30–16	SRCy_RATIO	**Sampling Ratio of Frame Syncs.** See bits 14–0.
31	SRCy_MUTEOUT	**Mute Status.** See bit 15.

Precision Clock Generator Registers

The precision clock generator (PCG) consists of four identical units. Each of these units (A, B, C, and D) generates one clock (`CLKA_0` `CLKB_0`, `CLKC_0` or `CLKD_0`) and one frame sync (`FSA_0`, `FSB_0`, `FSC_0` or `FSD_0`) output.

Control Registers (PCG_CTLxy)

Two control registers (y=0, 1) operate for each unit. The control registers enable clocks, frame syncs, and select divisors for each unit. These registers are shown in Figure A-109 and Figure A-110 and described in Table A-96 and Table A-97. Note the different units (x = A, B, C, D) any clock unit can be chosen.

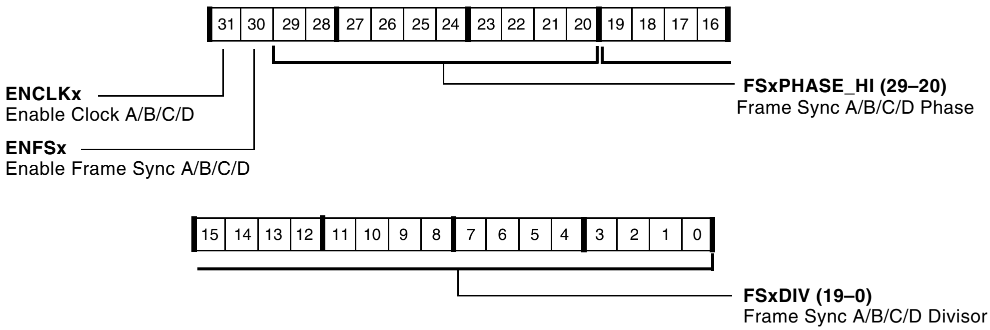

Figure A-109. PCG_CTLx0 Registers

Table A-96. PCG_CTLx0 Register Bit Descriptions (RW)

Bit	Name	Description
19–0	FSxDIV	**Divisor for Frame Sync A/B/C/D.** This 20-bit field frame sync divider is multiplexed: FSxDIV >1 PCGx is in normal mode FSxDIV =0,1 PCGx is in bypass mode Fore more information on bypass mode, refer to the STROBEx and INVFSx bits of the PCG_PWx register.
29–20	FSxPHASE_HI	**Phase for Frame Sync A/B/C/D.** This field represents the upper half of the 20-bit value for the channel A/B/C/D frame sync phase. See also FSXPHASE_LO (Bits 29-20) in Table A-97.
30	ENFSx	**Enable Frame Sync A/B/C/D.** 0 = Specified frame sync generation disabled 1 = Specified frame sync generation enabled
31	ENCLKx	**Enable Clock A/B/C/D.** 0 = Specified clock generation disabled 1 = Specified clock generation enabled

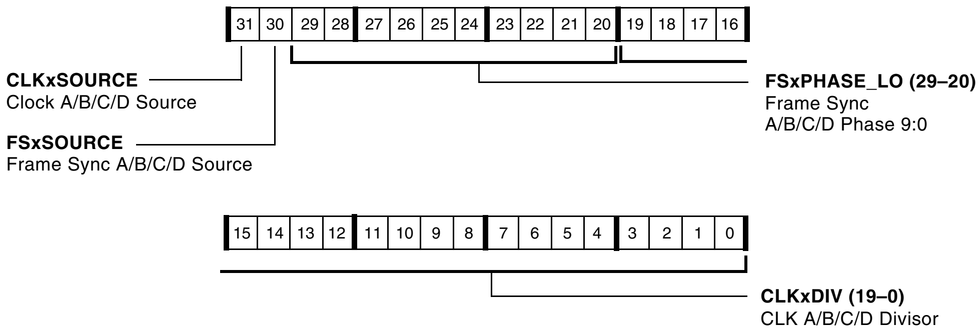

Figure A-110. PCG_CTLx1 Register

Table A-97. PCG_CTLx1 Register Bit Descriptions (RW)

Bit	Name	Description
19–0	CLKxDIV	**Divisor for Clock A/B/C/D.**
29–20	FSxPHASE_LO	**Phase for Frame Sync A/B/C/D.** This field represents the lower half of the 20-bit value for the channel A/B/C/D frame sync phase. See also FSX-PHASE_HI (Bits 29-20) in PCG_CTLx1 described on on page A-190.
30	FSxSOURCE	**Frame Sync Source.** Master clock source for frame sync A/B/C/D. 0 = CLKIN pin selected for specified frame sync 1 = PCG_EXTX_I selected for specified frame sync This frame sync period is also a reference for the strobe period in one shot mode.
31	CLKxSOURCE	**Clock Source.** Master clock source for clock A/B/C/D. 0 = CLKIN pin selected for specified clock 1 = PCG_EXTx_I selected for specified clock

Clock Inputs

The CLKxSOURCE bit (bit 31 in the PCG_CTLx1 registers) specifies the input source for the clock of the respective units (A, B, C, and D). When this bit is cleared (= 0), the input is sourced from the external oscillator/crystal, as shown in Figure 15-1 on page 15-6. When set (= 1), the input is sourced from DAI. The CLKxSOURCE bit is overridden if CLKx_SOURCE_IOP bit in the PCG_SYNCx register is set. If the CLKx_SOURCE_IOP bit is set, the input is sourced from the peripheral clock (PCLK).

Pulse Width Registers (PCG_PWx)

Pulse width is the number of input clock periods for which the frame sync output is high. Pulse width should be less than the divisor of the frame sync. The pulse width control registers are shown in Figure A-111 and Figure A-112 and described in Table A-98 and Table A-99. Note that where letters and slashes appear, for example A/B/C/D, any clock unit can be chosen.

If the STROBEA/B/C/D bits of the pulse width control register (PCG_PW, PCG_PW2) is reset to 0, then the input is directly passed to the frame sync output, either not inverted or inverted, depending on the INVFSA, INVFSB, INVFSC and INVFSD bits of the PCG_PW and PCG_PW2 registers.

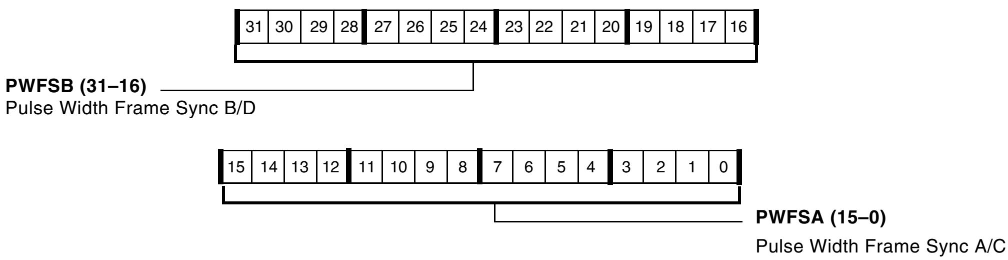

Figure A-111. PCG_PWx Registers (in Normal Mode)

Table A-98. PCG_PWx Register Bit Descriptions (in Normal Mode) (RW)

Bit	Name	Description
15–0	PWFSA	**Pulse Width for Frame Sync A/C.** Note: This is valid when not in bypass mode
31–16	PWFSB	**Pulse Width for Frame Sync B/D.** Note: This is valid when not in bypass mode

In bypass mode, if the least significant bit (LSB) of the PCG_PW register is set to 1, then a one-shot pulse is generated. This one-shot-pulse has a duration equal to the period of MISCA2_I for unit A, MISCA3_I for unit B, MISCA4_I for unit C, and MISCA5_I for unit D (see "DAI Routing Capabilities" on page 10-23). This pulse is generated either at the rising or at the falling edge of the input clock, depending on the value of the INVFSA, INVFSB, INVFSC, and INVFSD bits of the PCG_PW and PCG_PW2 registers.

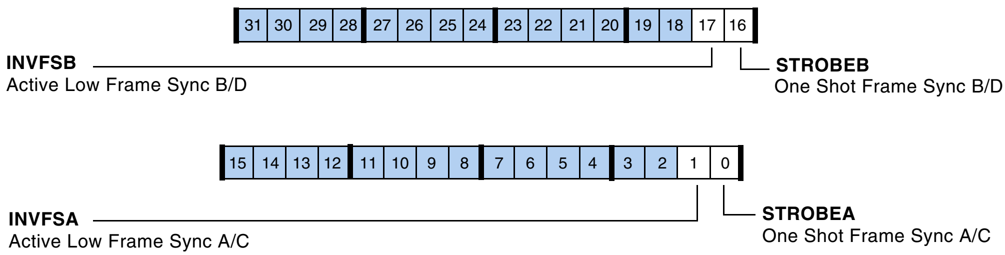

Figure A-112. PCG_PWx Registers (in Bypass Mode)

Table A-99. PCG_PWx Register Bit Descriptions (in Bypass Mode) (RW)

Bit	Name	Description
0	STROBEx	**One Shot Frame Sync A/C.** Frame sync is a pulse with duration equal to one period of the MISCA2_I signal (PCG A) MISCA4_I signal (PCG C) repeating at the beginning of every frame. Note: This is valid in bypass mode only.
1	INVFSx	**Active Low Frame Sync Select for Frame Sync A/C.** 0 = Active high frame sync 1 = Active low frame sync
15–2	Reserved (In bypass mode, bits 15-2 are ignored.)	
16	STROBEx	**One Shot Frame Sync B/D.** Frame sync is a pulse with duration equal to one period of the MISCA3_I signal (PCG B) MISCA5_I signal (PCG D) repeating at the beginning of every frame. Note: This is valid in bypass mode only.

Table A-99. PCG_PWx Register Bit Descriptions
(in Bypass Mode) (RW) (Cont'd)

Bit	Name	Description
17	INVFSx	**Active Low Frame Sync Select.** 0 = Active high frame sync 1 = Active low frame sync
31–18	Reserved (In bypass mode, bits 31–18 are ignored.)	

PCG Frame Synchronization Registers (PCG_SYNCx)

These registers (x = 0, 1), shown in Figure A-113, and Figure A-114 and described in Table A-100 and Table A-101, allow programs to synchronize the clock frame syncs units with external frame syncs.

Note the CLKxSOURCE bits (PCG_CTLx1 register) are overridden if CLKx_SOURCE_IOP bits (bit 2) in the PCG_SYNCx registers are set.

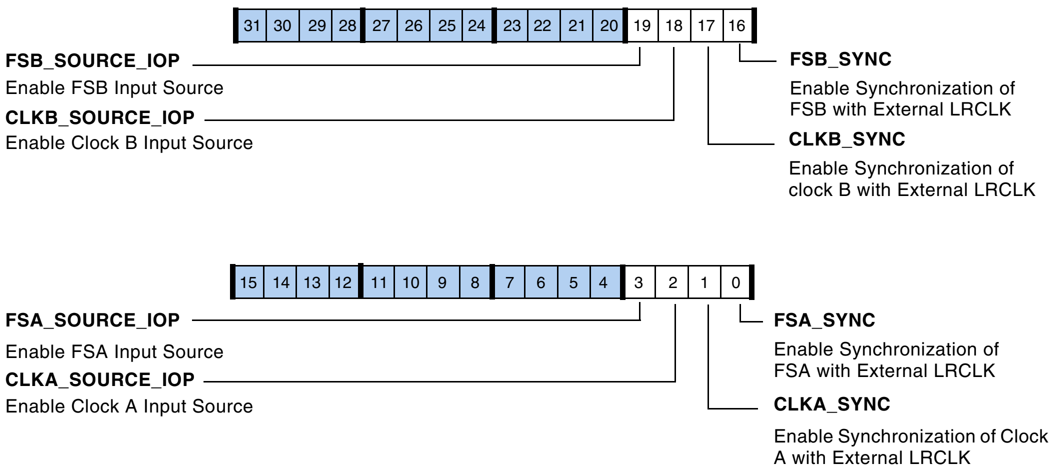

Figure A-113. PCG_SYNC1 Register

Table A-100. PCG_SYNC1 Register Bit Descriptions (RW)

Bit	Name	Description
0	FSA_SYNC	**Enable Synchronization of Frame Sync A With External Frame Sync.** 0 = Frame sync disabled 1 = Frame sync enabled
1	CLKA_SYNC	**Enable Synchronization of Clock A With External Frame Sync.** 0 = Clock disabled 1 = Clock enabled
2	CLKA_SOURCE_IOP	**Enable Clock A Input Source.** 0 = Output selected by CLKASOURCE bit 1 = PCLK (PCLK=CCLK/2 derived from core PLL) selected for clock A.
3	FSA_SOURCE_IOP	**Enable Frame Sync A Input Source.** 0 = Output selected by FSASOURCE bit 1 = PCLK (PCLK=CCLK/2 derived from core PLL) selected for frame sync A.
16	FSB_SYNC	**Enable Synchronization of Frame Sync B With External Frame Sync.** 0 = Frame sync disabled 1 = Frame sync enabled
17	CLKB_SYNC	**Enable Synchronization of Clock B With External Frame Sync.** 0 = Clock disabled 1 = Clock enabled
18	CLKB_SOURCE_IOP	**EnabLe Clock B Input Source.** 0 = Output selected by CLKBSOURCE bit 1 = PCLK (PCLK=CCLK/2 derived from core PLL) selected for clock B.
19	FSB_SOURCE_IOP	**Enable Frame Sync B Input Source.** 0 = Output selected by FSBSOURCE bit 1 = PCLK (PCLK=CCLK/2 derived from core PLL) selected for frame sync B.

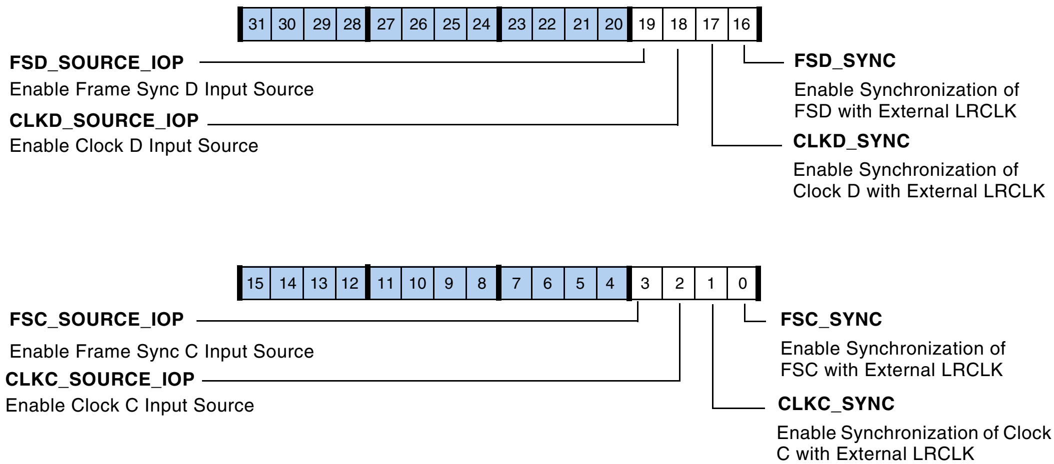

Figure A-114. PCG_SYNC2 Register

Table A-101. PCG_SYNC2 Register Bit Descriptions (RW)

Bit	Name	Description
0	FSC_SYNC	**Enable Synchronization of Frame Sync C With External Frame Sync.** 0 = Frame sync disabled 1 = Frame sync enabled
1	CLKC_SYNC	**Enable Synchronization of Clock C With External Frame Sync.** 0 = Clock disabled 1 = Clock enabled
2	CLKC_SOURCE_IOP	**Enable Clock C Input Source.** 0 = Output selected by CLKCSOURCE bit 1 = PCLK selected for clock C
3	FSC_SOURCE_IOP	**Enable Frame Sync C Input Source.** 0 = Output selected by FSCSOURCE bit 1 = PCLK selected for frame sync C
16	FSD_SYNC	**Enable Synchronization of Frame Sync D With External Frame Sync.** 0 = Frame sync disabled 1 = Frame sync enabled

Table A-101. PCG_SYNC2 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
17	CLKD_SYNC	**Enable Synchronization of Clock D With External Frame Sync.** 0 = Clock disabled 1 = Clock enabled
18	CLKD_SOURCE_IOP	**Enable Clock D Input Source.** 0 = Output selected by CLKDSOURCE bit 1 = PCLK selected for clock D
19	FSD_SOURCE_IOP	**Enable Frame Sync D Input Source.** 0 = Output selected by FSDSOURCE bit 1 = PCLK selected for frame sync D

Sony/Philips Digital Interface Registers

The following sections describe the registers that are used to configure, enable, and report status information for the S/PDIF transceiver.

Transmitter Registers

The following sections describe the S/PDIF transmitter registers.

Transmit Control Register (DITCTL)

This 32-bit register's bits are shown in Figure A-115 and described in Table A-102.

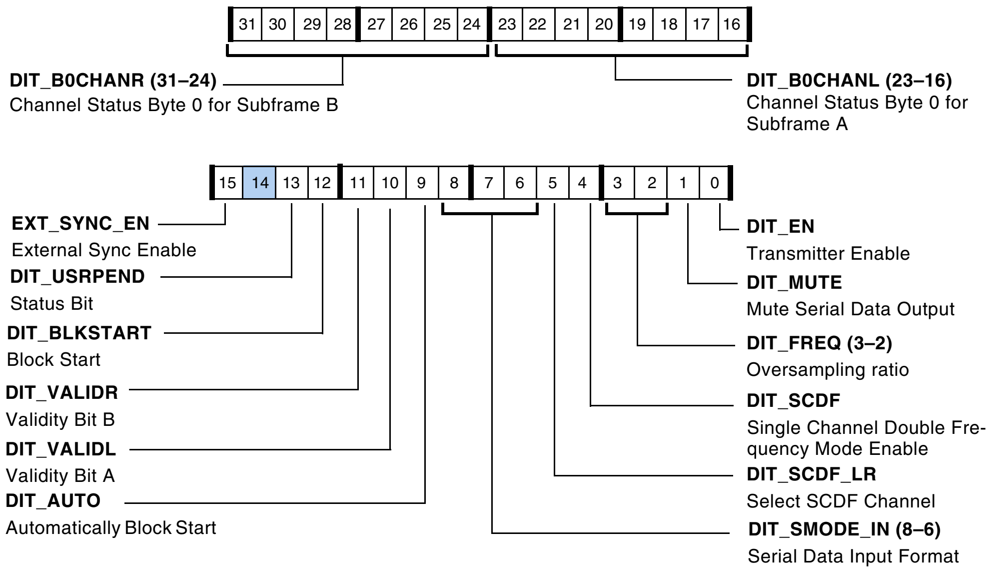

Figure A-115. DITCTL Register

Table A-102. DITCTL Register Bit Descriptions (RW)

Bit	Name	Description
0	DIT_EN	**Transmitter Enable.** Enables the transmitter and resets the control registers to their defaults. 0 = Transmitter disabled 1 = Transmitter enabled
1	DIT_MUTE	**Mute.** Mutes the serial data output.
3–2	DIT_FREQ	**Frequency Multiplier.** Sets the over sampling ratio to the following: 00 = 256 × frame sync 01 = 384 × frame sync
4	DIT_SCDF	**Single-Channel, Double-Frequency Mode Enable.** 0 = 2 channel mode 1 = SCDF mode
5	DIT_SCDF_LR	**Select Single-Channel, Double-Frequency Mode.** 0 = Left channel 1 = Right channel

Table A-102. DITCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
8–6	DIT_SMODEIN	**Serial Data Input Format.** Selects the input format as follows: 000 = Left-justified 001 = I^2S 010 = Reserved 011 = Reserved 100 = Right-justified, 24-bits 101 = Right-justified, 20-bits 110 = Right-justified, 18-bits 111 = Right-justified, 16-bits
9	DIT_AUTO	**Automatically Generate Block Start.** Automatically generate block start. When enabled, the transmitter is in standalone mode where it inserts block start, channel status, and validity bits on its own. If the channel status or validity buffer needs to be enabled (after the SRU programming is complete), first write to the buffers with the required data and then enable the buffers by setting the DIT_AUTO bit. 0 = Manually start block transfer according to input stream status bits 1 = Automatically start block transfer.
10	DIT_VALIDL	**Validity Bit A.** Use with channel status buffer.
11	DIT_VALIDR	**Validity Bit B.** Use with channel status buffer.
12 (RO)	DIT_BLKSTART	**Block Start.** Status bit that indicates block start (when bit 9, DIT_AUTO = 1). 0 = Current word is not block start 1 = Current word is block start
13 (RO)	DIT_USRPEND	**User Bits Pending.** This bit is set if the update of the internal buffer from the DITUSRBITA/Bx registers has not completed yet.
14	Reserved	
15	EXT_SYNC_EN	**External Sync Enable.** When set (Regardless of bit 9) the internal frame counter is set to zero at an internal LRCLK rising edge followed by an DIT_EXTSYNC_I rising edge.
23–16	DIT_B0CHANL	**Channel Status Byte 0 for Subframe A.**
31–24	DIT_B0CHANR	**Channel Status Byte 0 for Subframe B.**

Transmit Status Bit Registers for Subframe A/B (DITCHANAx/Bx)

These registers provide status information for transmitter subframe A and B. The first five bytes of the channel status may be written all at once to the control registers for both A and B channels. As the data is serialized and transmitted, the appropriate bit is inserted into the channel status area of the 192-word frame. Note that these registers are used in standalone mode only.

There are six channel status registers associated with subframe A (left channel) and six user bits buffer registers associated with subframe B (right channel). Since a block owns 2 x 192 frames, 24 bytes per frame are required for storage. Note that status byte 0 is available in the `DITCTL` register. These registers are listed with their locations in Table A-103 and Table A-104.

Table A-103. DITCHANAx Registers (RW)

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DITCTL			BYTE0	
DITCHANA0	BYTE1	BYTE2	BYTE3	BYTE4
DITCHANA1	BYTE5	BYTE6	BYTE7	BYTE8
DITCHANA2	BYTE9	BYTE10	BYTE11	BYTE12
DITCHANA3	BYTE13	BYTE14	BYTE15	BYTE16
DITCHANA4	BYTE17	BYTE18	BYTE19	BYTE20
DITCHANA5	BYTE21	BYTE22	BYTE23	Reserved

Table A-104. DITCHANBx Registers (RW)

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DITCTL				BYTE0
DITCHANB0	BYTE1	BYTE2	BYTE3	BYTE4
DITCHANB1	BYTE5	BYTE6	BYTE7	BYTE8

Table A-104. DITCHANBx Registers (RW) (Cont'd)

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DITCHANB2	BYTE9	BYTE10	BYTE11	BYTE12
DITCHANB3	BYTE13	BYTE14	BYTE15	BYTE16
DITCHANB4	BYTE17	BYTE18	BYTE19	BYTE20
DITCHANB5	BYTE21	BYTE22	BYTE23	Reserved

Transmit User Bits Buffer Registers for Subframe A/B Registers (DITUSRBITAx/Bx)

Once programmed, these registers are used only for the next block of data. This allows programs to change the user bit information with every block of data. After writing to the appropriate registers to change the user bits for the next block, `DITUSRBITAx` and `DITUSRBITBx` must be written to enable the use of these bits. Note these registers are used in standalone mode only.

There are six user bits buffer registers associated with subframe A (left channel) and six user bits buffer registers associated with subframe B (right channel). These registers are listed with their locations in Table A-105 and Table A-106.

Table A-105. DITUSRBITAx Registers (RW)

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DITUSRBITA0	BYTE0	BYTE1	BYTE2	BYTE3
DITUSRBITA1	BYTE4	BYTE5	BYTE6	BYTE7
DITUSRBITA2	BYTE8	BYTE9	BYTE10	BYTE11
DITUSRBITA3	BYTE12	BYTE13	BYTE14	BYTE15
DITUSRBITA4	BYTE16	BYTE17	BYTE18	BYTE19
DITUSRBITA5	BYTE20	BYTE21	BYTE22	BYTE23

Table A-106. DITUSRBITBx Registers (RW)

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DITUSRBITB0	BYTE0	BYTE1	BYTE2	BYTE3
DITUSRBITB1	BYTE4	BYTE5	BYTE6	BYTE7
DITUSRBITB2	BYTE8	BYTE9	BYTE10	BYTE11
DITUSRBITB3	BYTE12	BYTE13	BYTE14	BYTE15
DITUSRBITB4	BYTE16	BYTE17	BYTE18	BYTE19
DITUSRBITB5	BYTE20	BYTE21	BYTE22	BYTE23

User Bit Update Register (DITUSRUPD)

This register is a 1-bit wide register (RW). After writing to the user bits registers (`DITURSBITAx` and `DITUSRBITBx`), a value of 0x1 must be written into `DITUSRUPD` register to enable the use of these bits in the next block of transfer.

Receiver Registers

The following sections describe the receiver registers.

Receive Control Register (DIRCTL)

This 32-bit register, described in Table A-107 is used to set up error control and single-channel double-frequency mode.

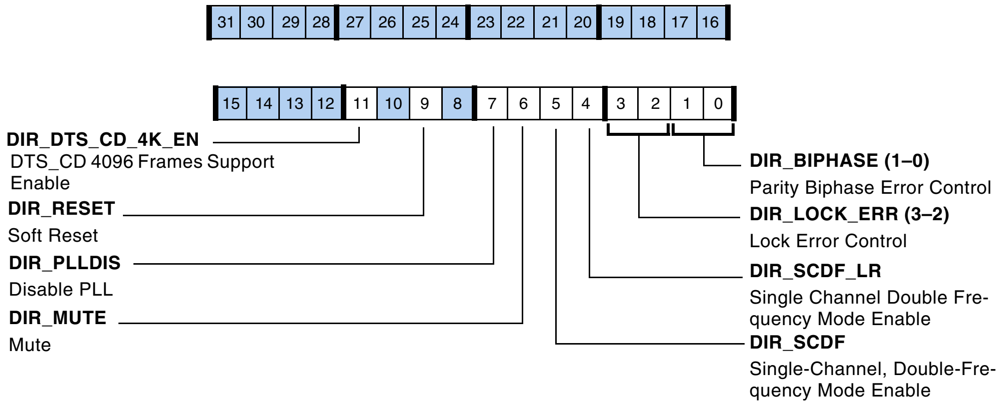

Figure A-116. DIRCTL Register

Table A-107. DIRCTL Register Bit Descriptions (RW)

Bit	Name	Description
1–0	DIR_BIPHASE	**Parity Biphase Error Control.** When a parity or biphase error occurs, the audio data will be handled according to these bits. 00 = No action taken 01 = Hold last valid sample 10 = Replace invalid sample with zeros 11 = Reserved
3–2	DIR_LOCK_ERR	**Lock Error Control.** When the DIR_LOCK bit in the DIRSTAT register is deasserted, it means the PLL has become unlocked and the audio data is handled according to these bit settings. 00 = No action taken 01 = Hold last valid sample 10 = Send zeros after the last valid sample 11 = Soft mute of the last valid audio is performed (as if NOSTREAM is asserted). This is valid only when linear PCM audio data is in the stream. When non-linear audio data is in the stream, this mode defaults to the case of DIR_LOCK_ERR1–0 bits = 10.
4	DIR_SCDF_LR	**Single-Channel, Double-Frequency Channel Select.** 0 = Left channel 1 = Right channel

Table A-107. DIRCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
5	DIR_SCDF	**Single-Channel, Double-Frequency Mode Enable.** 0 = 2 channel mode enabled 1 = SCDF mode
6	DIR_MUTE	**Mute.** 0 = Mute disabled 1 = Mute serial data outputs, maintaining clocks (digital black)
8–7	Reserved	
9	DIR_RESET	**Reset S/PDIF Receiver.** By default, the S/PDIF receiver is always enabled. If this bit is set, the S/PDIF receiver and digital PLL are disabled.
10	Reserved	
11	DIR_DTS_CD_4K_EN	**DTS_CD 4096 Frames Support Enable.** If this bit is set, and if NON-AUDIO preamble is detected, then the DIR_NO-AUDIOLR bit is asserted high and remains high if another NON AUDIO preamble is detected within 4096 frames, otherwise it is cleared. The assertion and deassertion of DIR_NOAUDIO bit can generate the DIR_NOAUDIO_INT DAI interrupt, if unmasked in the DAI_IRPTL_FE/DAI_IRPTL_RE interrupt mask registers. This bit is supported with on-chip Digital PLL only. This bit is applicable only for the ADSP-2147x and ADSP-2148x processors.
31–12	Reserved	

Receive Status Register (DIRSTAT)

The Status register consists of status bits (VALIDITY, NONAUDIO, NOSTREAM, BIPHERR, PARITYERR and LOCK), indicate the status of various functions supported by S/PDIF Receiver. It also has the lower byte of the 40-bit channel status information. The VALIDITY, NOSTREAM, BIPHERR, PARITYERR and LOCK bits are sticky and cleared on read. This register also contains the lower byte of the 40-bit channel status information. The bit settings for these registers are shown in Figure A-117 and described in Table A-108.

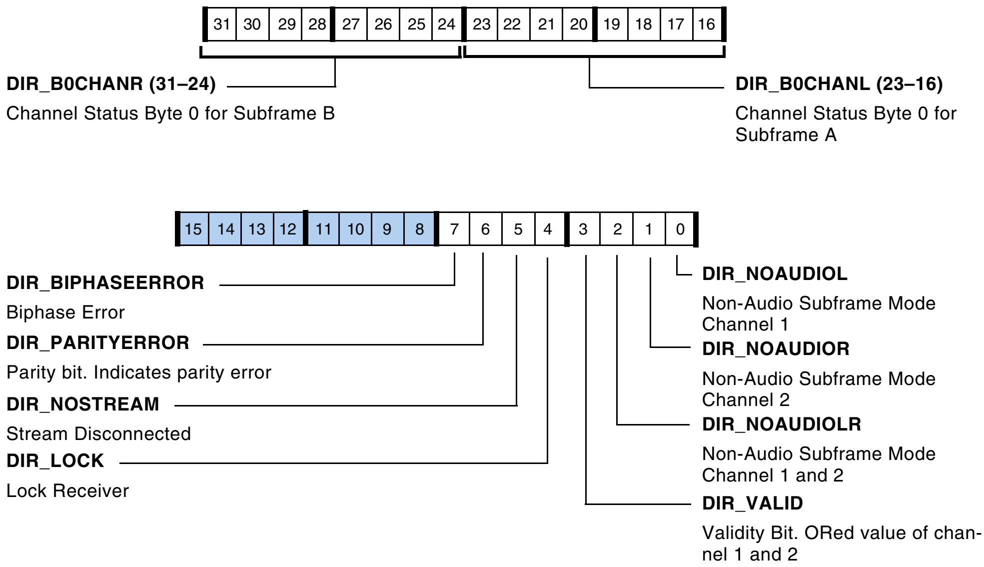

Figure A-117. DIRSTAT Register

Table A-108. DIRSTAT Register Bit Descriptions (RO)

Bit	Name	Description
0	DIR_NOAUDIOL	**Non-Audio Subframe Mode Channel 1.** Based on SMPTE 337M. 0 = Not non-audio subframe mode 1 = Non-audio subframe mode, channel 1
1	DIR_NOAUDIOR	**Non-Audio Subframe Mode Channel 2.** Based on SMPTE 337M. 0 = Not non-audio subframe mode 1 = Non-audio subframe mode, channel 2
2	DIR_NOAUDIOLR	**Non-Audio Frame Mode Channel 1 and 2.** Based on SMPTE 337M. 0 = Not non-audio frame mode 1 = Non-audio frame mode
3 (ROC)	DIR_VALID	**Validity Bit (sticky).** ORed bits of channel 1 and 2. 0 = Linear PCM data 1 = Non-linear audio data

Table A-108. DIRSTAT Register Bit Descriptions (RO) (Cont'd)

Bit	Name	Description
4 (ROC)	DIR_LOCK	**Lock Receiver (sticky).** When set (=1), the digital PLL of receiver is locked, the corresponding DIR_LOCK bit is set. This bit can be polled to detect the DIR_LOCK condition. After the receiver is locked, the other status bits in DIRSTAT and the channel status (DIRCHANL/R) registers can be read. Interrupts can be also used with some status bits. 0 = Receiver not locked 1 = Receiver locked
5 (ROC)	DIR_NOSTREAM	**No Stream Error (sticky).** Asserted when the AES3/SPDIF stream is disconnected. When this bit is asserted and the audio data in the stream is linear PCM, the receiver performs a soft mute of the last valid sample from the AES3/SPDIF stream. A soft mute consists of taking the last valid audio sample and slowly and linearly decrementing it to zero, over a period of 4096 frames. During this time, the PLL three-states the charge pump until the soft mute has been completed. If non-linear PCM audio data is in the AES3/SPDIF stream when the NOSTREAM bit is asserted, the receiver sends out zeros after the last valid sample. 0 = Stream not disconnected 1 = Stream disconnected (default)
6 (ROC)	DIR_PARITYERROR	**Parity Bit (sticky).** When cleared, (=0), indicates that the AES3/SPDIF stream was received with the correct parity, or even parity. When set (=1), indicates that an error has occurred, and the parity is odd. 0 = No parity error 1 = Parity error
7 (ROC)	DIR_BIPHASEERROR	**Biphase Error (sticky).** When set (=1), indicates that a bi-phase error has occurred and the data sampled from the biphase stream may not be correct. 0 = No biphase error 1 = Biphase error
15–8	Reserved	
23–16	DIR_B0CHANL	**Channel Status Byte 0 for Subframe A.**
31–24	DIR_B0CHANR	**Channel Status Byte 0 for Subframe B.**

Receive Status Registers for Subframe A (DIRCHANA)

The S/PDIF receiver stores a maximum of 5 bytes (40-bit) status information. Note that status byte 0 is available in the DIRCTL register. This 32-bit register is described in Table A-109.

Table A-109. DIRCHANAx Registers (RO)

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DIRSTAT			BYTE0	
DIRCHANA	BYTE1	BYTE2	BYTE3	BYTE4

Receive Status Registers for Subframe B (DIRCHANB)

The S/PDIF receiver stores a maximum of 5 bytes (40-bit) status information. Note that status byte 0 is available in the DIRCTL register. This 32-bit register is described in Table A-110.

Table A-110. DIRCHANBx Registers (RO)

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DIRSTAT				BYTE0
DIRCHANB	BYTE1	BYTE2	BYTE3	BYTE4

Shift Register Control Register

The following section provides bit information for the shift register module.

Control Register (SR_CTL)

This register, shown in Figure A-118 and described in Table A-111, enables the peripheral.

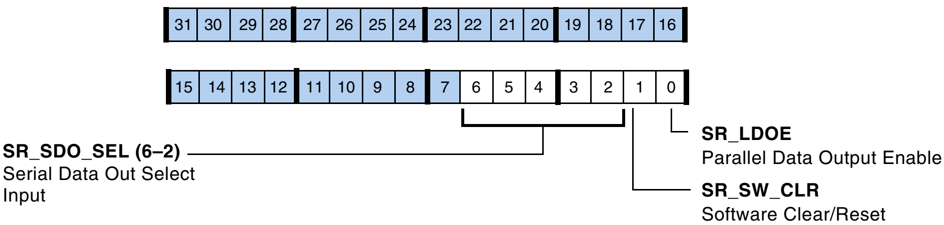

Figure A-118. SR_CTL Register

Table A-111. SR_CTL Register Bit Descriptions (RW)

Bit	Name	Description
0	SR_LDOE	**Parallel Data Output Enable.** This bit enables the parallel SR_LD017–0 output pins. It is cleared on chip reset (RESET) and/or asynchronously on dedicated SR_CLR pin.
1	SR_SW_CLR	**Software Clear/Reset.** If this bit is 0, then the reset is active. 0 = Shift register cleared 1 = Shift register enabled
6–2	SR_SDO_SEL	**Serial Data Out Multiplexer's Select Input.** These bits select which parallel word is shifted through the SR_SDO pin. 00000 = LSB selected. 10001 = MSB selected.
31–7	Reserved	

DPI Signal Routing Unit Registers

The digital peripheral interface is comprised of a group of peripherals and the signal routing unit 2 (SRU2).

Group A – Miscellaneous Signals

The registers and input signals for group A are summarized in Figure A-119 through Figure A-124 and Table A-112.

Source Signals

Table A-112. Group A Connections

Selection Code	Signal	Description (Source Selection)
00000 (0x0)	LOW	Logic Level Low (0)
00001 (0x1)	HIGH	Logic Level High (1)
00010 (0x2)	DPI_PB01_O	External Pin 1
00011 (0x3)	DPI_PB02_O	External Pin 2
00100 (0x4)	DPI_PB03_O	External Pin 3
00101 (0x5)	DPI_PB04_O	External Pin 4
00110 (0x6)	DPI_PB05_O	External Pin 5
00111 (0x7)	DPI_PB06_O	External Pin 6
01000 (0x8)	DPI_PB07_O	External Pin 7
01001 (0x9)	DPI_PB08_O	External Pin 8
01010 (0xA)	DPI_PB09_O	External Pin 9
01011 (0xB)	DPI_PB10_O	External Pin 10
01100 (0xC)	DPI_PB11_O	External Pin 11
01101 (0xD)	DPI_PB12_O	External Pin 12
01110 (0xE)	DPI_PB13_O	External Pin 13
01111 (0xF)	DPI_PB14_O	External Pin 14

Table A-112. Group A Connections (Cont'd)

Selection Code	Signal	Description (Source Selection)
10000 (0x10)	TIMER0_O	Timer0 Output
10001 (0x11)	TIMER1_O	Timer1 Output
10010 (0x12)	Reserved	
10011 (0x13)	UART0_TX_O	UART0 Transmitter Output
10100 (0x14)	Reserved	
10101-11111	Reserved	

Destination Control Signal Registers (SRU2_INPUTx)

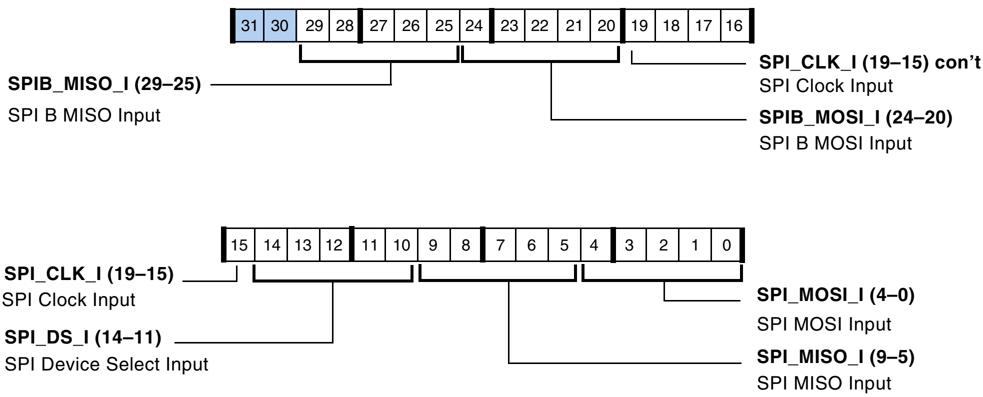

Figure A-119. SRU2_INPUT0 Register (RW)

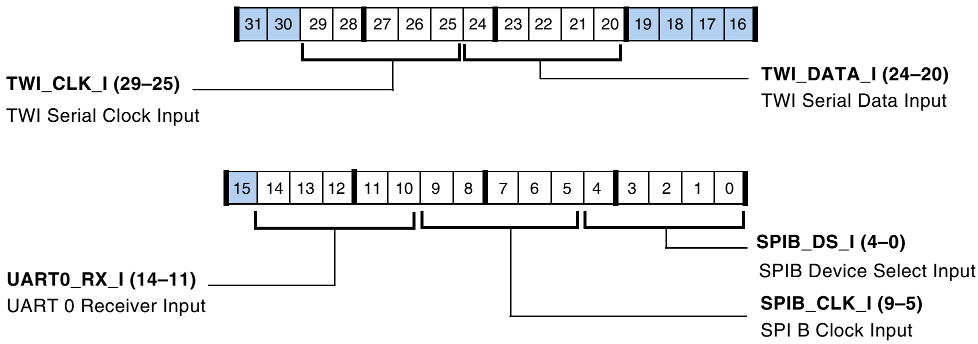

Figure A-120. SRU2_INPUT1 Register (RW)

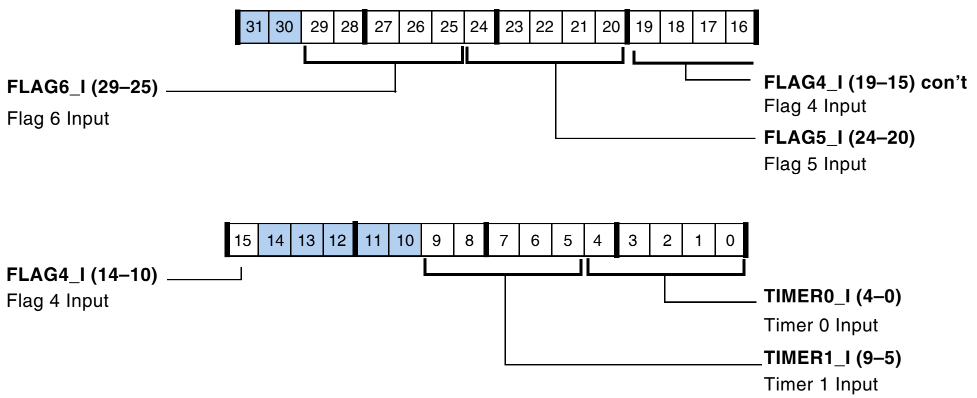

Figure A-121. SRU2_INPUT2 Register (RW)

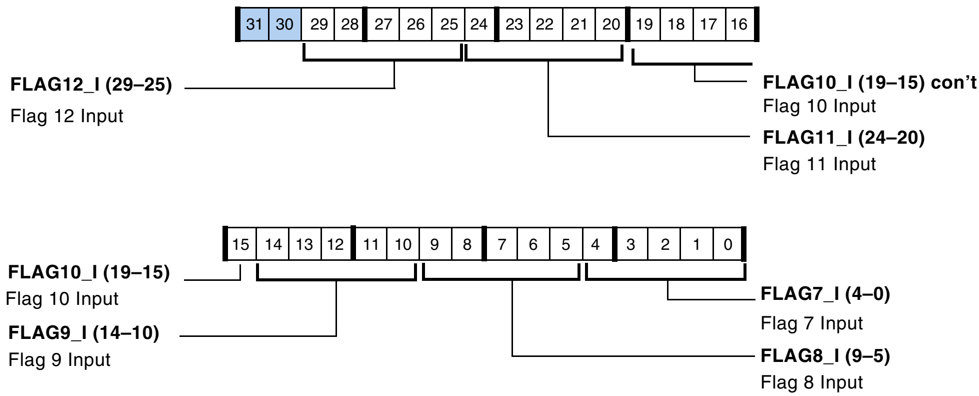

Figure A-122. SRU2_INPUT3 Register (RW)

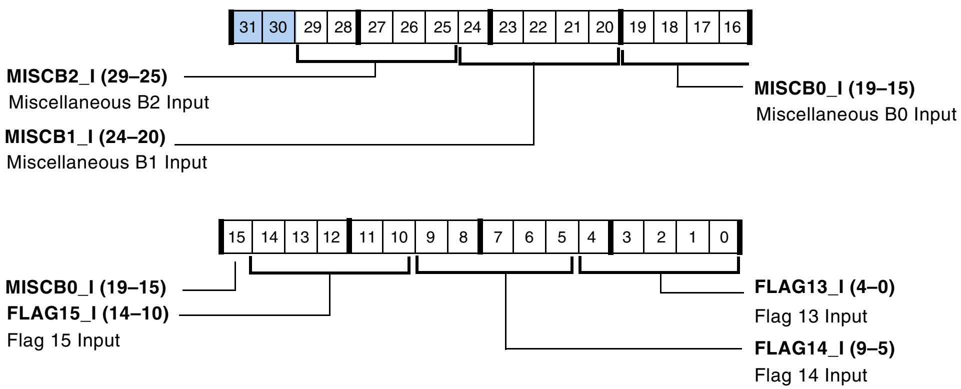

Figure A-123. SRU2_INPUT4 Register (RW)

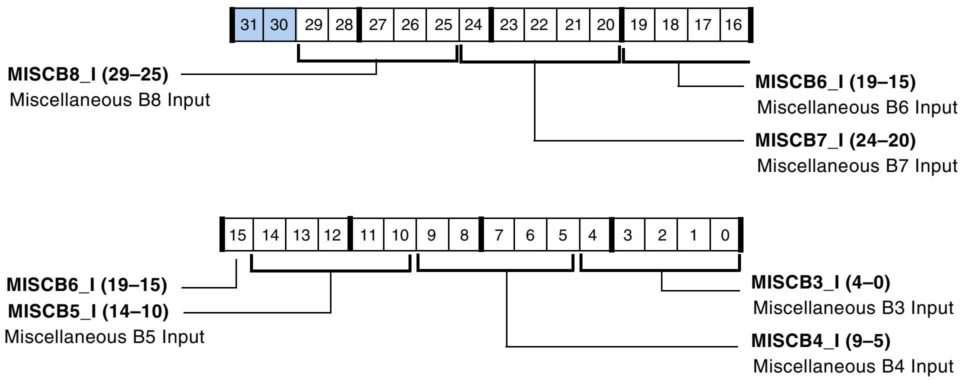

Figure A-124. SRU2_INPUT5 Register (RW)

Group B – Pin Assignment Signal

Group B connections, shown in Figure A-125 through Figure A-127 and Table A-113, are used to route output signals to the 14 DPI pins.

For the ADSP-2147x and ADSP-2148x processors, the outputs of PWM units 3–1 can be routed to the DPI pins. See locations 0x23 – 0x2E in Table A-113.

Source Signals

Table A-113. Group B Signals

Binary	Signal	Description (Source Selection)
000000 (0x0)	LOW	Logic Level Low (0)
000001 (0x1)	HIGH	Logic Level High (1)
000010 (0x2)	DPI_PB01_O	External Pin 1
000011 (0x3)	DPI_PB02_O	External Pin 2
000100 (0x4)	DPI_PB03_O	External Pin 3
000101 (0x5)	DPI_PB04_O	External Pin 4

Table A-113. Group B Signals (Cont'd)

Binary	Signal	Description (Source Selection)
000110 (0x6)	DPI_PB05_O	External Pin 5
000111 (0x7)	DPI_PB06_O	External Pin 6
001000 (0x8)	DPI_PB07_O	External Pin 7
001001 (0x9)	DPI_PB08_O	External Pin 8
001010 (0xA)	DPI_PB09_O	External Pin 9
001011 (0xB)	DPI_PB10_O	External Pin 10
001100 (0xC)	DPI_PB11_O	External Pin 11
001101 (0xD)	DPI_PB12_O	External Pin 12
001110 (0xE)	DPI_PB13_O	External Pin 13
001111 (0xF)	DPI_PB14_O	External Pin 14
010000 (0x10)	TIMER0_O	Timer0 Output
010001 (0x11)	TIMER1_O	Timer1 Output
010010 (0x12)	Reserved	
010011 (0x13)	UART0_TX_O	UART0 Transmitter Output
010100 (0x14)	Reserved	
010101 (0x15)	SPI_MISO_O	MISO from SPI
010110 (0x16)	SPI_MOSI_O	MOSI from SPI
010111 (0x17)	SPI_CLK_O	Clock Output from SPI
011000 (0x18)	SPI_FLG0_O	Slave Select 0 from SPI
011001 (0x19)	SPI_FLG1_O	Slave Select 1 from SPI
011010 (0x1A)	SPI_FLG2_O	Slave Select 2 from SPI
011011 (0x1B)	SPI_FLG3_O	Slave Select 3 from SPI
011100 (0x1C)	SPIB_MISO_O	MISO from SPIB
011101 (0x1D)	SPIB_MOSI_O	MOSI from SPIB
011110 (0x1E)	SPIB_CLK_O	Clock Output from SPIB
011111 (0x1F)	SPIB_FLG0_O	Slave Select 0 from SPIB

Table A-113. Group B Signals (Cont'd)

Binary	Signal	Description (Source Selection)
100000 (0x20)	SPIB_FLG1_O	Slave Select 1 from SPIB
100001 (0x21)	SPIB_FLG2_O	Slave Select 2 from SPIB
100010 (0x22)	SPIB_FLG3_O	Slave Select 3 from SPIB
100011 (0x23)	FLAG4_O	Flag/PWM 4 Output[1]
100100 (0x24)	FLAG5_O	Flag/PWM 5 Output
100101 (0x25)	FLAG6_O	Flag/PWM 6 Output
100110 (0x26)	FLAG7_O	Flag/PWM 7 Output
100111 (0x27)	FLAG8_O	Flag/PWM 8 Output
101000 (0x28)	FLAG9_O	Flag/PWM 9 Output
101001 (0x29)	FLAG10_O	Flag/PWM 10 Output
101010 (0x2A)	FLAG11_O	Flag/PWM 11 Output
101011 (0x2B)	FLAG12_O	Flag/PWM 12 Output
101100 (0x2C)	FLAG13_O	Flag/PWM 13 Output
101101 (0x2D)	FLAG14_O	Flag/PWM 14 Output
101110 (0x2E)	FLAG15_O	Flag/PWM 15 Output
101111 (0x2F)	PCG_CLKC_O	Precision Clock Generator Clock C Out
110000 (0x30)	PCG_CLKD_O	Precision Clock Generator Clock D Out
110001 (0x31)	PCG_FSC_O	Precision Clock Generator Frame Sync C Out
110010 (0x32)	PCG_FSD_O	Precision Clock Generator Frame Sync D Out
110011–111111	Reserved	

1 PWM not available on ADSP-2146x models.

Destination Signal Control Registers (SRU2_PINx)

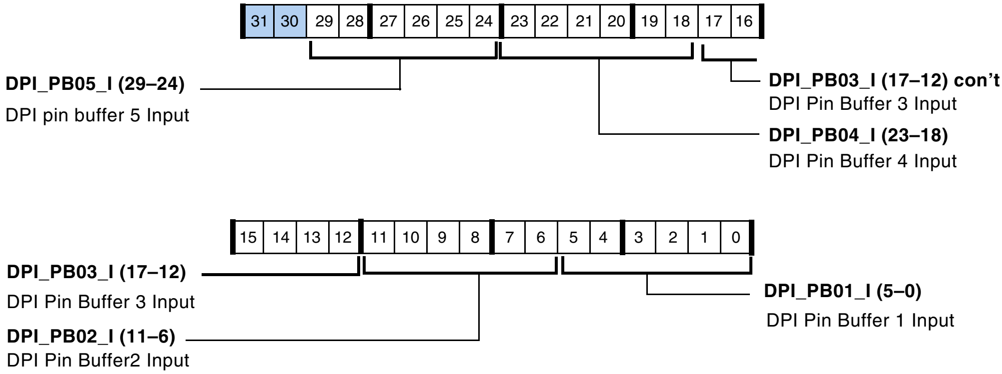

Figure A-125. SRU2_PIN0 Register

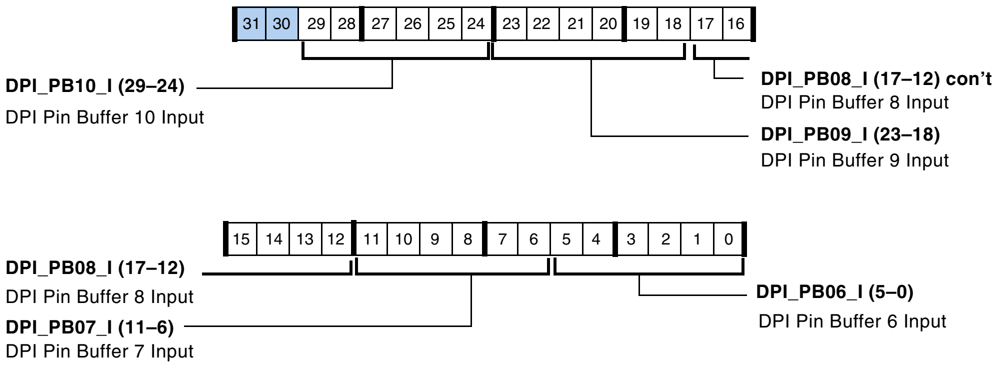

Figure A-126. SRU2_PIN1 Register

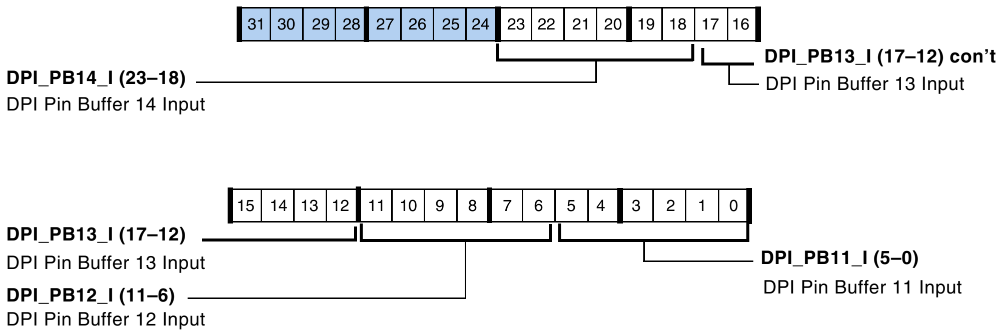

Figure A-127. SRU2_PIN2 Register

Group C – Pin Enable

Group C signals, shown in Table A-114, are used to specify whether each DPI pin is used as an output or an input by setting the source for the pin buffer enable. When a pin buffer enable (`DPI_PBENxx_I`) is set (= 1), the signal present at the corresponding pin buffer input (`DPI_PBxx_I`) is driven off chip as an output. When a pin buffer enable is cleared (= 0), the signal present at the corresponding pin buffer input is ignored.

The pin enable control registers activate the drive buffer for each of the 14 DPI pins. When the pins are not enabled (driven), they can be used as inputs.

The registers that control group C settings are shown in Figure A-128 through Figure A-130.

The TWI output must operate as an open-drain output, the DPI input pins used for TWI data and clock should be connected to logic level 0.

Source Signals

Table A-114. Group C Signals

Binary	Signal	Description (Source Selection)
000000 (0x0)	LOW	Logic Level Low (0)
000001 (0x1)	HIGH	Logic Level High (1)
000010 (0x2)	MISCB0_O	Miscellaneous Control 0
000011 (0x3)	MISCB1_O	Miscellaneous Control 1
000100 (0x4)	MISCB2_O	Miscellaneous Control 2
000101 (0x5)	TIMER0_PBEN_O	Enable for Timer 0 Output
000110 (0x6)	TIMER1_PBEN_O	Enable for Timer 1 Output
000111 (0x7)	Reserved	
001000 (0x8)	UART0_TX_PBEN_O	Pin Enable for UART 0 Transmitter
001001 (0x9)	Reserved	
001010 (0xA)	SPIMISO_PBEN_O	Pin Enable for MISO from SPI
001011 (0xB)	SPIMOSI_PBEN_O	Pin Enable for MOSI from SPI
001100 (0xC)	SPICLK_PBEN_O	Pin Enable for CLK from SPI
001101 (0xD)	SPIFLG0_PBEN_O	Pin Enable for Slave Select 0 from SPI
001110 (0xE)	SPIFLG1_PBEN_O	Pin Enable for Slave Select 1 from SPI
001111 (0xF)	SPIFLG2_PBEN_O	Pin Enable for Slave Select 2 from SPI
010000 (0x10)	SPIFLG3_PBEN_O	Pin Enable for Slave Select 3 from SPI
010001 (0x11)	SPIBMISO_PBEN_O	Pin Enable for MISO from SPIB
010010 (0x12)	SPIBMOSI_PBEN_O	Pin Enable for MOSI from SPIB
010011 (0x13)	SPIBCLK_PBEN_O	Pin Enable for CLK from SPIB
010100 (0x14)	SPIBFLG0_PBEN_O	Pin Enable for Slave Select 0 from SPIB
010101 (0x15)	SPIBFLG1_PBEN_O	Pin Enable for Slave Select 1 from SPIB
010110 (0x16)	SPIBFLG2_PBEN_O	Pin Enable for Slave Select 2 from SPIB
010111 (0x17)	SPIBFLG3_PBEN_O	Pin Enable for Slave Select 3 from SPIB

Table A-114. Group C Signals (Cont'd)

Binary	Signal	Description (Source Selection)
011000 (0x18)	FLAG4_PBEN_O	Pin Enable for Flag 4 Output
011001 (0x19)	FLAG5_PBEN_O	Pin Enable for Flag 5 Output
011010 (0x1A)	FLAG6_PBEN_O	Pin Enable for Flag 6 Output
011011 (0x1B)	FLAG7_PBEN_O	Pin Enable for Flag 7 Output
011100 (0x1C)	FLAG8_PBEN_O	Pin Enable for Flag 8 Output
011101 (0x1D)	FLAG9_PBEN_O	Pin Enable for Flag 9 Output
011110 (0x1E)	FLAG10_PBEN_O	Pin Enable for Flag 10 Output
011111 (0x1F)	FLAG11_PBEN_O	Pin Enable for Flag 11 Output
100000 (0x20)	FLAG12_PBEN_O	Pin Enable for Flag 12 Output
100001 (0x21)	FLAG13_PBEN_O	Pin Enable for Flag 13 Output
100010 (0x22)	FLAG14_PBEN_O	Pin Enable for Flag 14 Output
100011 (0x23)	FLAG15_PBEN_O	Pin Enable for Flag 15 Output
100100 (0x24)	TWI_DATA_PBEN_O	Data Output Enable from TWI
100101 (0x25)	TWI_CLK_PBEN_O	Clock Output Enable from TWI
100110 (0x26)	MISCB3_O	Miscellaneous Control 3
100111 (0x27)	MISCB4_O	Miscellaneous Control 4
101000 (0x28)	MISCB5_O	Miscellaneous Control 5
101001 (0x29)	MISCB6_O	Miscellaneous Control 6
101010 (0x2A)	MISCB7_O	Miscellaneous Control 7
101011 (0x2B)	MISCB8_O	Miscellaneous Control 8
101100–111111	Reserved	

Destination Control Signal Registers (SRU2_PBENx)

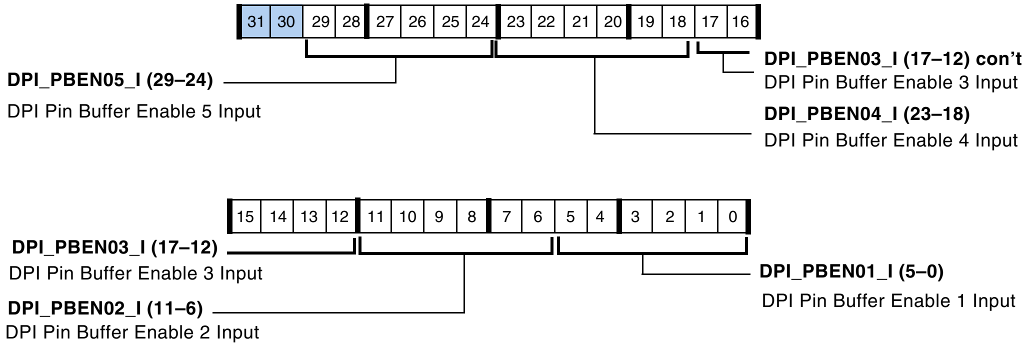

Figure A-128. SRU2_PBEN0 Register

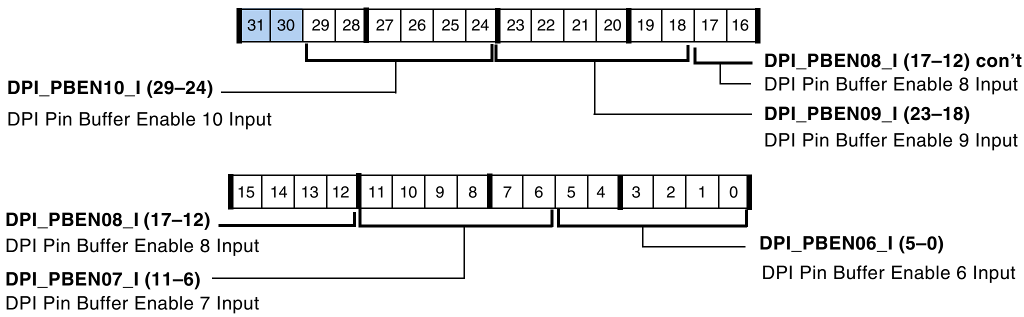

Figure A-129. SRU2_PBEN1 Register

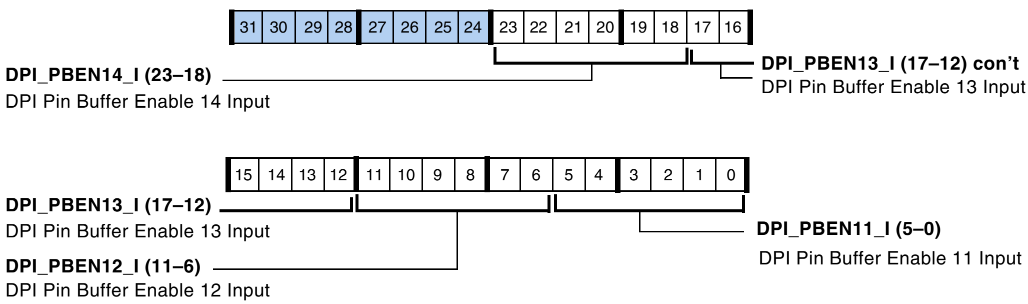

Figure A-130. SRU2_PBEN2 Register

DPI Pin Buffer Status Register (DPI_PIN_STAT)

The register is shown in Figure A-131 returns the status of `DPI_PB14-1` pin buffers. This register is updated at PCLK/2 rate.

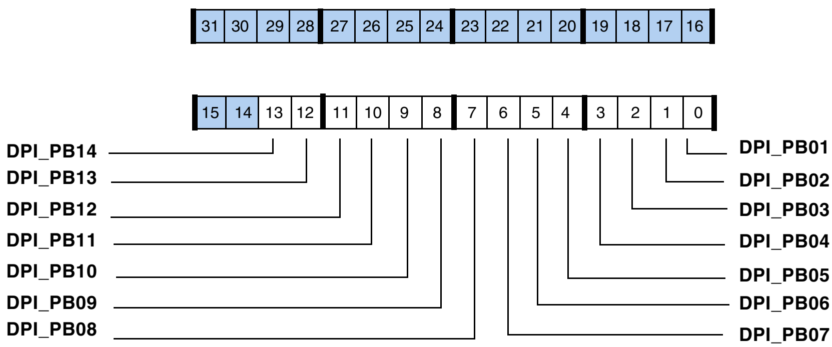

Figure A-131. DPI_PIN_STAT Register

Peripherals Routed Through the DPI

The following sections provide information on the peripherals that are explicitly routed through the digital peripheral interface.

Serial Peripheral Interface Registers

The following sections describe the registers associated with the two serial peripheral interfaces (SPIs).

Control Registers (SPICTL, SPICTLB)

The SPI control (SPICTL) registers are used to configure and enable the SPI system. The bit settings for these registers are shown in Figure A-132 and described in Table A-115.

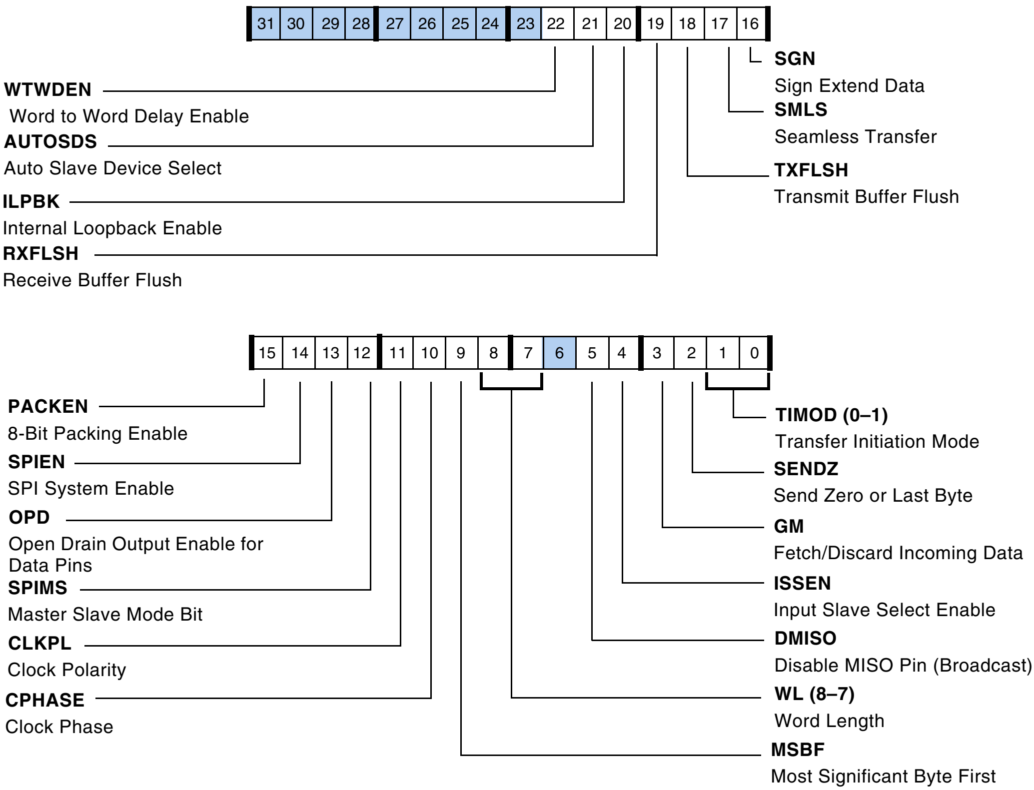

Figure A-132. SPICTL, SPICTLB Registers (Bits 15–0)

Table A-115. SPICTL Register Bit Descriptions (RW)

Bit	Name	Description
1–0	TIMOD	**Transfer Initiation Mode.** Defines transfer initiation mode and interrupt generation. 00 = Initiate transfer by read of receive buffer. Interrupt active when receive buffer is full. 01 = Initiate transfer by write to transmit buffer. Interrupt active when transmit buffer is empty. 10 = Enable DMA transfer mode. Interrupt configured by DMA. 11 = Reserved
2	SENDZ	**Send Zero.** Send zero or the last word when TXSPI is empty. 0 = Send last word 1 = Send zeros
3	GM	**Get Data.** When RXSPI is full, get data or discard incoming data. 0 = Discard incoming data 1 = Get more data, overwrites the previous data
4	ISSEN	**Input Slave-Select Enable.** Enables slave-select input ($\overline{\texttt{SPI\_DS\_I}}$ pin) for the master. $\overline{\texttt{SPI\_DS\_I}}$ operation depends on the SPI configuration. If the SPI is a slave, $\overline{\texttt{SPI\_DS\_I}}$ acts as the slave-select input. The state of this input pin is observable in bit 7 of the SPIFLGx register. As master, $\overline{\texttt{SPI\_DS\_I}}$ can serve as an error-detection input in a multimaster environment. The ISSEN-bit enables this feature. When ISSEN =1, the $\overline{\texttt{SPI\_DS\_I}}$ input is the master mode error input; otherwise, $\overline{\texttt{SPI\_DS\_I}}$ is ignored. 0 = Disable 1 = Enable
5	DMISO	**Disable MISO Pin.** Disables MISO as an output. This is needed in an environment where a master wishes to transmit to various slaves at one time (broadcast). However, only one slave is allowed to transmit data back to the master. This bit should be set for all slaves, except the one from whom the master wishes to receive data. Different CPUs or processors can take turns being master, 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. The (DMISO) bit disables MISO as an output. 0 = MISO enabled 1 = MISO disabled

Table A-115. SPICTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
6	Reserved	
8–7	WL	**Word Length.** SPI port can transmit and receive three word widths: 00 = 8 bits 01 = 16 bits 10 = 32 bits **8-bit word.** 8-bit word. When transmitting, the SPI port sends out only the lower eight bits of the word written to the SPI buffer. When receiving, the SPI port packs the 8-bit word to the lower 32 bits of the RXSPI buffer while the upper bits in the registers are zeros. **16-bit word.** When transmitting, the SPI port sends out only the lower 16 bits of the word written to the SPI buffer. When receiving, the SPI port packs the 16-bit word to the lower 32 bits of the RXSPI buffer while the upper bits in the register are zeros. **32-bit word.** No packing of the RXSPI or TXSPI registers is necessary as the entire 32-bit register is used for the data word.
9	MSBF	**Most Significant Byte First.** 0 = LSB sent/received first 1 = MSB sent/received first
10	CPHASE	**Clock Phase.** Selects the transfer format. 0 = SPICLK starts toggling at the middle of 1st data bit 1 = SPICLK starts toggling at the start of 1st data bit (default setting)
11	CLKPL	**Clock Polarity.** 0 = Active high SPICLK (SPICLK low is the idle state) 1 = Active low SPICLK (SPICLK high is the idle state) Note that the CLKPL/CPHASE bits define the SPI mode.
12	SPIMS	**SPI Master Select.** Configures SPI module as master or slave. 0 = Device is a slave device 1 = Device is a master device

Table A-115. SPICTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
13	OPD	**Open Drain Output Enable.** Enables open drain data output enable for MOSI and MISO pins. 0 = Drive (MOSI and MISO driven) 1 = Open drain (MOSI and MISO three-stated). In a multimaster/slave SPI system, the data output pins (MOSI and MISO) can be configured to behave as open drain drivers to prevent contention and possible damage to pin drivers. An external pull-up resistor is required on both the MOSI/MISO pins when this option is selected. When the OPD bit is set and the SPI ports are configured as masters, the SPI_MOSI_O pin is three-stated. Instead the SPI_MOSI_PBEN_O pin is driven and act as output enable pin. Note that the corresponding DPI input buffer pin should be tied to GND. Similarly, when OPD is set and the SPI ports are configured as slaves, the SPI_MISO_O pin is three-stated. Instead the SPI_MISO_PBEN_O pin is driven and act as output enable pin. Note that the corresponding DPI input buffer pin should be tied to GND. See "Pin Buffers as Open Drain" on page 10-13.
14	SPIEN	**SPI Port Enable.** Enables the SPI port. If configured as a master (SPIMS=1) and SPIEN=0, the MOSI and SPICLK outputs are disabled, and the MISO input is ignored. If configured as a slave (SPIMS=0) and SPIEN=0, the MOSI and SPICLK inputs are ignored, and the MISO output is disabled. The SPIEN and SPIMS bits can be cleared by hardware if the MME-bit is set. For SPI slaves, the slave-select input ($\overline{\text{SPI\_DS\_I}}$) acts like a reset for the internal SPI logic. For this reason, the $\overline{\text{SPI\_DS\_I}}$ line must be error free. The SPIEN bit can also be used as a software reset of the internal SPI logic. An exception to this is the RW1C-type (read-write 1-to-clear) bits in the SPISTATx registers. These bits remain set if they are already set. Note: always clear the RW1C-type bits in SPISTATx registers before re-enabling the SPI, as these bits do not get cleared even if the SPI is disabled. This can be done by writing 0xFF to the SPISTATx registers. In the case of an MME error, enable the SPI ports after $\overline{\text{SPI\_DS\_I}}$ is deasserted. 0 = SPI module is disabled 1 = SPI module is enabled. When this bit transitions from high to low the RX and TX buffers are flushed which takes 2 core clock cycles.

Table A-115. SPICTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
15	PACKEN	**Packing Enable.** The SPI unpacks data when it transmits and packs data when it receives. In order to communicate with 8-bit SPI devices and store 8-bit words in internal memory, a packed transfer feature is built into the SPI port. 0 = No packing 1 = 8 to 16-bit packing Note: This bit may be 1 only when WL = 00 (8-bit transfer). When in transmit mode, the PACKEN bit unpacks data. When packing is enabled, two 8-bit words are packed into one 32-bit word. When the SPI port is transmitting, two 8-bit words are unpacked from one 32-bit word. When receiving, words are packed into one 32-bit word from two 8-bit words. The value 0xXXLMXXJK (where XX is any random value and JK and LM are data words to be transmitted out of the SPI port) is written to the TXSPI register. The processor transmits 0xJK first and then transmits 0xLM. The receiver packs the two words received, 0xJK and then 0xLM, into a 32-bit word. They appear in the RXSPI register as: 0x00LM00JK => if SGN is configured to 0 or L, J < 7 0xFFLMFFJK => if SGN is configured to 1 and L, J > 7
16	SGN	**Sign Extend.** 0 = No sign extension 1 = Sign extension
17	SMLS	**Seamless Transfer**. 0 = Seamless transfer disabled. After each word transfer there is a delay before the next word transfer starts. The delay is 2.5 SPICLK cycles 1 = Seamless transfer enabled. There is no delay before the next word starts, a seamless operation. Not supported in mode TIMOD1-0 = 00 and CPHASE=0 for all modes.
18	TXFLSH	**Flush Transmit Buffer.** Write a 1 to clear TXSPI. 0 = TXSPI not cleared 1 = TXSPI cleared. This bit can be set together with the enable bit. Note this bit is not self-clearing
19	RXFLSH	**Clear RXSPI.** Write a 1 to clear RXSPI. 0 = RXSPI not cleared 1 = RXSPI cleared. This bit can be set together with the enable bit. Note this bit is not self-clearing

Table A-115. SPICTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
20	ILPBK	**Internal Loopback.** This mode interconnects the MOSI with the MISO pins internally. In this mode the SPIMS bit must be set. 0 = No internal loopback 1 = Internal loopback enabled
21	AUTOSDS	**Auto Slave device Select.** 0 = Auto slave device select controlled by SPI hardware only for CPHASE=0 1 = Auto slave device select controlled by SPI hardware regardless for CPHASE setting Feature not supported if SMLS-bit is set
22	WTWDEN	**Word to Word Delay Enable.** According to AUTOSDS bit, the length of the chip de-select pulse is programmable. 0 = Word to word delay is fixed. (See the product-specific data sheet.) 1 = Word to word delay programmable by SPIBAUD[25:20]. This feature not supported if the SMLS bit is set.
31–23	Reserved	

DMA Configuration Registers (SPIDMAC, SPIDMACB)

These 17-bit SPI registers are used to control DMA transfers and are shown in Figure A-133 and described in Table A-116.

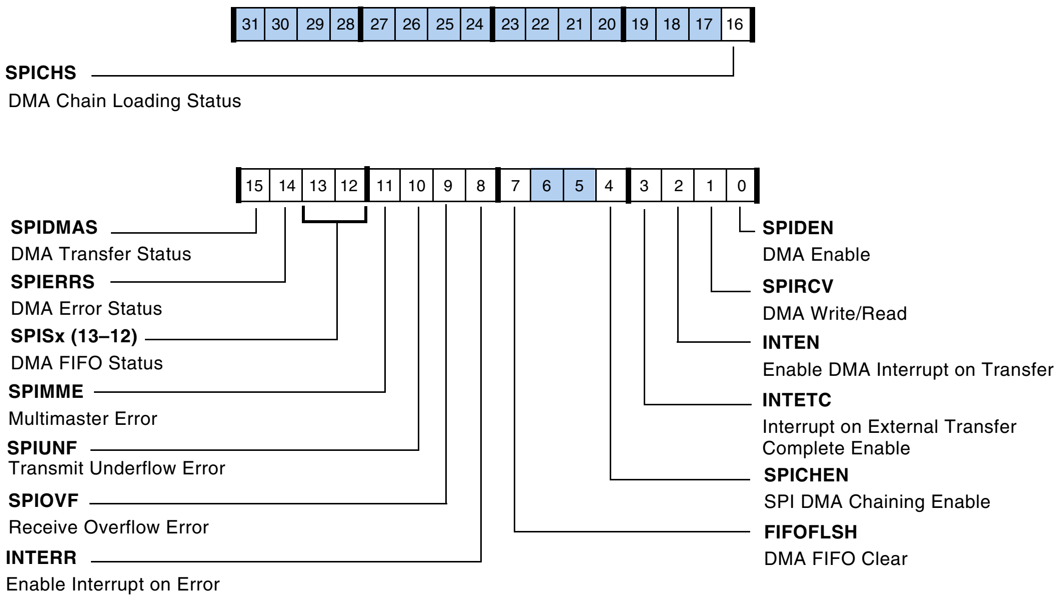

Figure A-133. SPIDMAC, SPIDMACB Registers

Table A-116. SPIDMAC, SPIDMACB Register Bit Descriptions (RW)

Bit	Name	Description
0	SPIDEN	**DMA Enable.** 0 = Disable 1 = Enable
1	SPIRCV	**DMA Write/Read.** 0 = SPI transmit (read from internal memory) 1 = SPI receive (write to internal memory)
2	INTEN	**Enable DMA Interrupt on Transfer.** 0 = Disable 1 = Enable

Table A-116. SPIDMAC, SPIDMACB Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
3	INTETC	**Interrupt on External Transfer Complete Enable.** Selects interrupt event for transmit DMA 0 = DMA interrupt generated when DMA count reaches zero. 1 = DMA interrupt generated when last bit of last word is shifted out. Note: both INTEN and INTETC bits, when enabled, generate an interrupt for INTETC.
4	SPICHEN	**SPI DMA Chaining Enable.** 0 = Disable 1 = Enable
6–5	Reserved	
7	FIFOFLSH	**DMA FIFO Clear.** If this bit is set, it takes 2 core clock cycles to flush the buffer. It clears the SPIS bit. 0 = Disable 1 = Enable Clearing the SPIEN/SPIDEN bits have no affect on the buffer Note this bit is not self-clearing
8	INTERR	**Enable Interrupt on Error.** 0 = Disable 1 = Enable
9 (RO)	SPIOVF	**Receive OverFlow Error (SPIRCV = 1).** 0 = Successful transfer 1 = Error – data received with RXSPI full
10 (RO)	SPIUNF	**Transmit Underflow Error (SPIRCV = 0).** 0 = Successful transfer 1 = Error occurred in transmission with no new data in TXSPI
11 (RO)	SPIMME	**Multimaster Error.** 0 = Successful transfer 1 = Error during transfer
13–12 (RO)	SPIS	**DMA FIFO Status.** 00 = FIFO empty 11 = FIFO full 10 = FIFO partially full 01 = Reserved

Table A-116. SPIDMAC, SPIDMACB Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
14 (RO)	SPIERRS	**DMA Error Status.** This bit is set if SPIOVF, SPIUNF or DPIMME bits are set. 0 = Successful DMA transfer 1 = Errors during DMA transfer
15 (RO)	SPIDMAS	**DMA Transfer Status.** 0 = DMA idle 1 = DMA in progress
16 (RO)	SPICHS	**DMA Chain Loading Status.** 0 = Chain idle 1 = Chain loading in progress
31–17	Reserved	

Baud Rate Registers (SPIBAUD, SPIBAUDB)

These SPI registers are used to set the bit transfer rate for a master device. When configured as slaves, the value written to these registers is ignored. The (SPIBAUDx) registers can be read from or written to at any time. Bit descriptions are provided in Table A-117. Note that the minimum value of BAUDR = 0x2, since the max SPICLK = PCLK/8 in master mode.

Table A-117. SPIBAUD, SPIBAUDB Register Bit Descriptions (RW)

Bit	Name	Description
0	Reserved	
15–1	BAUDR	**Baud Rate Enable.** Enables the master SPICLK per the equation: SPICLK baud rate = PCLK /(4 x BAUDR) Default = 0
19–16	Reserved	
25–20	STDC	**Sequential Transfer Delay.** The word to word delay(T4) = 1.5 SPI CLK Period + T3 and T3 = 0.5 SPICLK period for STDC = 0. T3 = STDC × SPICLK period for STDC > 0.
31–26	Reserved	

Status (SPISTAT, SPISTATB) Registers

The SPISTAT and SPISTATB registers are used to detect when an SPI transfer is complete, if transmission/reception errors occur, and the status of the TXSPI and RXSPI FIFOs. The bit settings for these registers are shown in Figure A-134 and described in Table A-118.

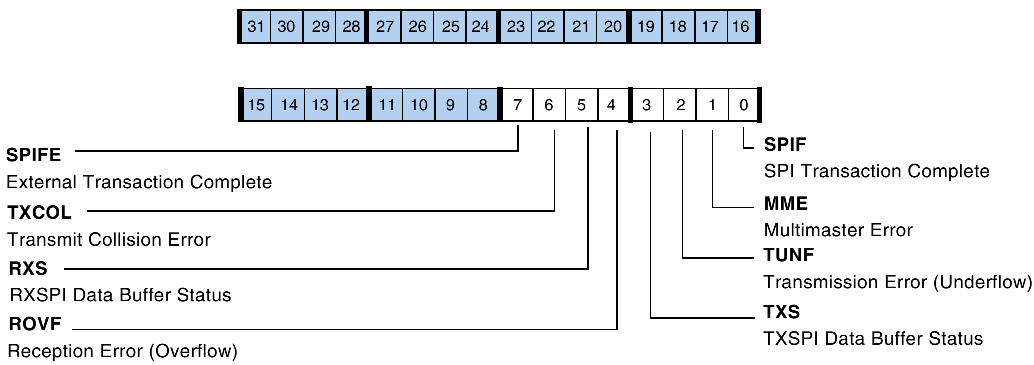

Figure A-134. SPISTAT, SPISTATB Registers

Table A-118. SPISTAT Register Bit Descriptions (RO)

Bit	Name	Description
0 (RO)	SPIF	**SPI Transmit or Receive Transfer Complete.** SPIF is set when an SPI single-word transfer is complete.
1 (RW1C)	MME	**Multimaster Error or Mode-Fault Error.** MME is set in a master device when some other device tries to become the master. In multimaster mode, if the $\overline{\text{SPI\_DS\_I}}$ input signal of a master is asserted (low) an error has occurred. This means that another device is also trying to be the master. Clears the SPIMME bit.

Table A-118. SPISTAT Register Bit Descriptions (RO) (Cont'd)

Bit	Name	Description
2 (RW1C)	TUNF	**Transmission Error.** TUNF is set when transmission occurred with no new data in TXSPI register. The TUNF bit (2) is set when all of the conditions of transmission are met and there is no new data in TXSPI (TXSPI is empty). In this case, the transmission contents depend on the state of the SENDZ bit in the SPICTL register. Clears the SPIUNF bit.
3 (RO)	TXS	**Transmit Data Buffer Status.** TXSPI data buffer status. 0 = Empty 1 = Full
4 (RW1C)	ROVF	**Reception Error.** ROVF is set when data is received with receive buffer full.
5 (RO)	RXS	**Receive Data Buffer Status.** The ROVF flag (bit 4) is when a new transfer has completed before the previous data is read from the RXSPI register. This bit indicates that a new word was received while the receive buffer was full. The ROVF flag is cleared by a RW1C-type software operation. The state of the GM bit in the SPICTL register determines whether the RXSPI register is updated with the newly received data or whether that new data is discarded. 0 = Empty 1 = Full
6 (RW1C)	TXCOL	**Transmit Collision Error.** When TXCOL is set, it is possible that corrupt data was transmitted. The TXCOL flag (bit 6) is set when a write to the TXSPI register coincides with the load of the shift register. The write to TXSPI can be via the software or the DMA. This bit indicates that corrupt data may have been loaded into the shift register and transmitted. In this case, the data in TXSPI may not match what was transmitted. This error can easily be avoided by proper software control. The TXCOL bit is cleared by a RW1C-type software operation. Note that this bit is never set when the SPI is configured as a slave with CPHASE = 0. The collision may occur, but it cannot be detected.

Table A-118. SPISTAT Register Bit Descriptions (RO) (Cont'd)

Bit	Name	Description
7 (RO)	SPIFE	**External Transaction Complete.** Set (= 1) when the SPI transaction is complete on the external interface. This bit is very useful in DMA mode showing that the peripheral has completed all the external transfers corresponding to the DMA programmed. For more information, see "Transfer Initiate Mode" on page 16-14 and "DMA Transfers" on page 16-26.
31–8	Reserved	

SPI Port Flags Registers (SPIFLG, SPIFLGB)

The SPIFLG and SPIFLGB registers are used to enable individual SPI slave-select lines when the SPI is enabled as a master. This register is ignored if the SPI is programmed as s slave. The bit settings for these registers are shown in Figure A-135 and described in Table A-119.

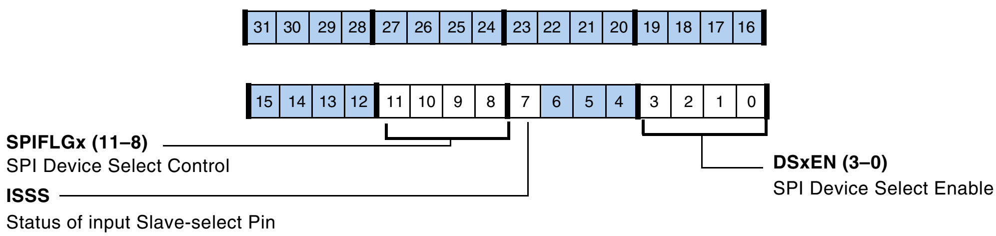

Figure A-135. SPIFLG, SPIFLGB Registers

Table A-119. SPIFLG, SPIFLGB Register Bit Descriptions (RW)

Bit	Name	Description
3–0	DSxEN	**SPI Device Select Enable.** Enables or disables the corresponding output signal to the SRU2 be used for SPI slave-select. 0 = Disable SPIFLGx output enable 1 = Enable SPIFLGx output enable Note DS0EN bit is set in SPI master mode only.
6–4	Reserved	

Table A-119. SPIFLG, SPIFLGB Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
7 (RO)	ISSS	**Input Slave Service Select.** Reflects the service selection for the slave-select input pin ($\overline{\texttt{SPI\_DS\_I}}$). 0 = $\overline{\texttt{SPI\_DS\_I}}$ pin ignored 1 = $\overline{\texttt{SPI\_DS\_I}}$ pin used as multimaster error detection (default for SPI only)
11–8	SPIFLGx	**SPI Device Select Control.** Selects (if cleared, = 0) a corresponding DPI pin (depending on pin routing) output to be asserted for an SPI slave-select. For AUTOSDS=1, there is automatic HW control regardless of CPHASE setting 0000 = All SPIFLGx cleared 1111 = All SPIFLGx set (default)
12–31	Reserved	

UART Control and Status Registers

The processor provides a set of PC-style, industry-standard control and status registers for each UART. These IOP registers are byte-wide registers that are mapped as half-words with the most significant byte zero-filled.

Global Control Registers (UART0TXCTL, UART0RXCTL)

Use these global registers (described in Table A-120 and Table A-121) to enable the UART receive or transmit controllers for core data transfers and both standard DMA or chained DMA.

Table A-120. UART0TXCTL Register Descriptions (RW)

Bit	Name	Description
0	UARTEN	**Transmit Control Enable.** This global bit must be set to enable the transmit path. 0 = Disable UART TX 1 = Enable UART TX If this bit transitions from high to low the TX buffer is flushed which takes 7 core clock cycles.
1	UARTDEN	**DMA Enable.** 0 = Disable DMA 1 = Enable DMA on the specified channel
2	UARTCHEN	**Chain Pointer DMA Enable.** 0 = Disable chained DMA 1 = Enable chained DMA on the specified channel

Table A-121. UART0RXCTL Register Descriptions (RW)

Bit	Name	Description
0	UARTEN	**Receive Control Enable.** This global bit must be set to enable the receive path. 0 = Disable UART RX 1 = Enable UART RX If this bit transitions from high to low the RX buffer is flushed which takes 7 core clock cycles.
1	UARTDEN	**DMA Enable.** 0 = Disables DMA 1 = Enables DMA on the specified channel
2	UARTCHEN	**Chain Pointer DMA Enable.** 0 = Disable chained DMA 1 = Enable chained DMA on the specified channel

Divisor Latch Registers (UART0DLL, UART0DLH)

The bit rate is characterized by the peripheral clock (PCLK) and the 16-bit divisor (2 x 8-bit. The divisor is split into the UART divisor latch low byte register (UART0DLL) and the UART divisor latch high byte register (UART0DLH), both shown in Figure A-136.

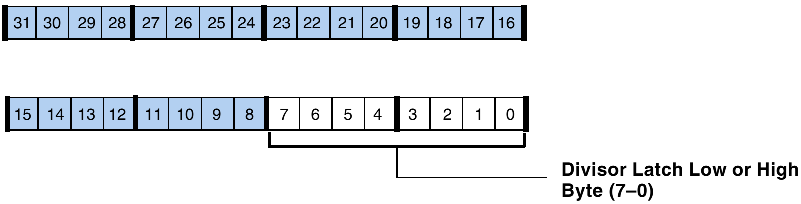

Figure A-136. UART Divisor Latch Registers (UART0DLL, UART0DLH)

Mode Register (UART0MODE)

The UART mode register controls miscellaneous settings as shown in Figure A-137 and described Table A-122.

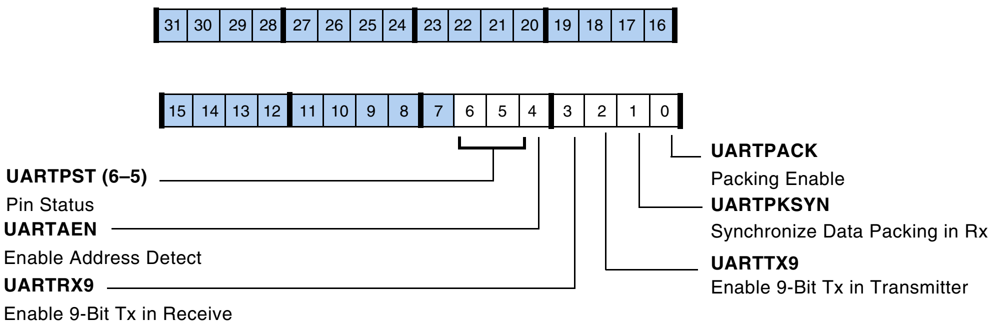

Figure A-137. UART0MODE Register

Table A-122. UART0MODE Register Bit Descriptions (RW)

Bit	Name	Description
0	UARTPACK	**Packing Enable.** 0 = No pack 1 = Packing enabled. Consecutive data words (example 0xAB and 0xCD) are packed as 0x00CD 00AB in the receiver, and 0x00CD 00AB is transmitted as two words of 0xAB and 0xCD successively from the transmitter. For more information, see “Data Packing/Unpacking” on page 21-8.
1 (RW1S)	UARTPKSYN	**Synchronize Data Packing in RX.** When written with a 1, the next data byte goes to the lower byte position of the RBR register. This bit always returns zero on reads.
2	UARTTX9	**Enable 9-Bit Data in Transmitter.** 0 = Word length select (WLS) 1 = 9-bit
3	UARTRX9	**Enable 9-Bit Data in Receiver.** 0 = Word length select (WLS) 1 = 9-bit
4	UARTAEN	**Enable Address Detect (If UARTRX9 = 1).** 0 = Disable address detection; all bytes are received 1 = Enable address detection; interrupt and load of RBR when RX9D is set
6–5	UARTPST	**Transmit Pin Status Control.** Changes the status/level of the transmit pin for a disabled UART TX control (UARTEN=0 in UART0TXCTL). 00 = UART0_TX_O is low 01 = UART0_TX_O is three-stated (default) 10 = UART0_TX_O is three-stated 11 = UART0_TX_O is high

Line Control Register (UART0LCR)

The UART line control register (shown in Figure A-138 and described in Table A-123) controls the format of received and transmitted character frames.

Some UART registers share the same IOP address. The UART0DLL registers are mapped to the same address as the UART0THR and UART0RBR registers. The UART0DLH registers are mapped to the same address as the interrupt enable registers (UART0IER). Note that the UARTDLAB bit in the UART0LCR register must be set before the UART divisor latch registers can be accessed. If the UARTDLAB bit is cleared, access to the UART0THR and UART0RBR or UART0IER registers occurs.

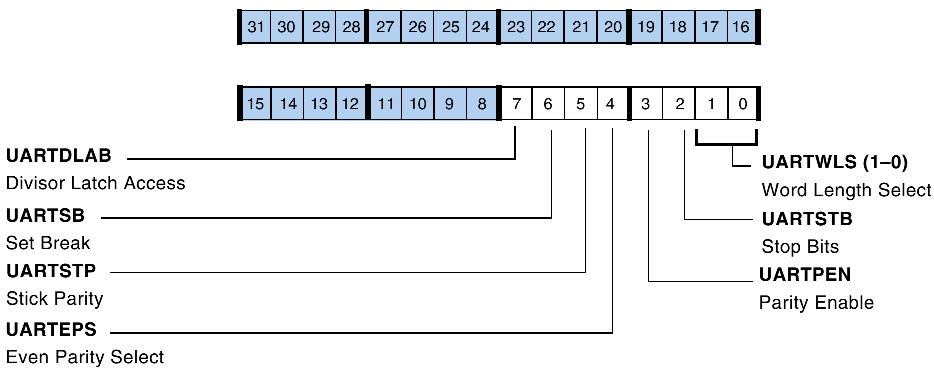

Figure A-138. UART0LCR Register

Table A-123. UART0LCR Register Bit Descriptions (RW)

Bit	Name	Description
1–0	UARTWLS	**Word Length Select.** 00 = 5-bit word(UARTWLS5) 01 = 6-bit word(UARTWLS6) 10 = 7-bit word(UARTWLS7) 11 = 8-bit word(UARTWLS8)
2	UARTSTB	**Stop Bits.** 0 = 1 stop bit 1 = 2 stop bits for non-5-bit word length or 1 1/2 stop bits for 5-bit word length
3	UARTPEN	**Parity Enable.** 0 = Parity not transmitted or checked 1 = Transmit and check parity
4	UARTEPS	**Even Parity Select.** 0 = Odd parity when PEN = 1 and STP = 0 1 = Even parity when PEN = 1 and STP = 0
5	UARTSTP	**Stick Parity.** Forces parity to defined value if set and PEN = 1. 0 = Parity transmitted and checked as 1 1 = Parity transmitted and checked as 0
6	UARTSB	**Set Break.** The UART transmit pin is three-state after reset. This bit is used to force the transmit pin to zero if the UARTEN bit set in UART0TXCTL register. Using this bit the UART TX pin can be used as a flag pin when the UART is not used. 0 = UART0_TX_O pin is high 1 = UART0_TX_O pin is low
7	UARTDLAB	**Divisor Latch Access Bit.** Because some IOP registers share the same address, this bit provides access as follows. 0 = Enable access to UART0THR, UART0RBR, and UART0IER registers 1 = Enable access to UART0DLL and UART0DLH registers

Line Status Register (UART0LSR)

The UART line status register (UART0LSR) contains UART status information as shown in Figure A-139 and described in Table A-124. There is also a shadow register, UART0LSRSH, that allows programs to read the contents of the corresponding main register without affecting the status the UART.

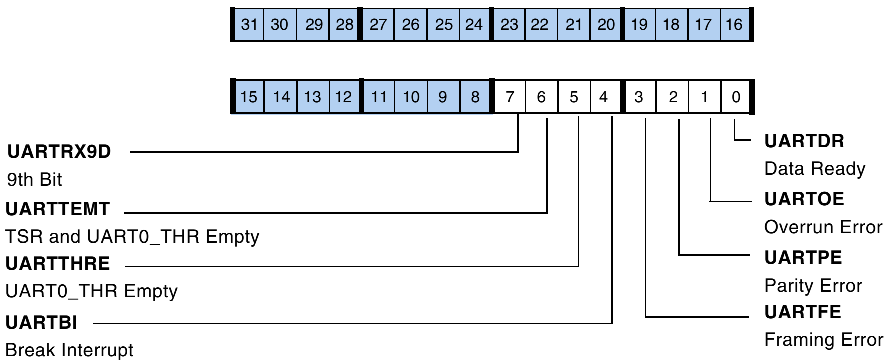

Figure A-139. UART0LSR Register

Table A-124. UART0LSR Register Bit Descriptions (RO)

Bit	Name	Description
0	UARTDR	**Data Ready.** This bit is cleared when the UART receive buffer (UART0RBR) is read. 0 = No new data 1 = UART0_RBR holds new data
1 (ROC[1, 2])	UARTOE	**Overrun Error.** 0 = No overrun 1 = UART0RBR overwritten before read
2 (ROC[1, 2])	UARTPE	**Parity Error.** 0 = No parity error 1 = Parity error
3 (ROC[1, 2])	UARTFE	**Framing Error.** 0 = No error 1 = Invalid stop bit error

Table A-124. UART0LSR Register Bit Descriptions (RO) (Cont'd)

Bit	Name	Description
4 (ROC[2])	UARTBI	**Break Interrupt.** Indicates that Rx pin was held low for more than the max word length. 0 = No break interrupt 1 = Break interrupt.
5	UARTTHRE	**UART0_THR Empty.** The UARTTHRE bit indicates that the UART transmit channel is ready for new data, and software can write to the UART0THR register. Writes to UART0THR clear the UARTTHRE bit. It is set again when data is copied from UART0THR to the transmit shift register (UART0TSR). The UARTTEMT bit can be evaluated to determine whether a recently initiated transmit operation has been completed. 0 = Not empty 1 = Empty (default)
6	UARTTEMT	**TSR and UART0_THR Empty.** 0 = Full 1 = Both empty
7	UARTRX9D	9th bit of the received character-address detect

1 These bits are read-only in the UART0LSRSH (shadow) register.
2 This bit is cleared by a read of the UART0LSR register.

Interrupt Enable Register (UART0IER)

The interrupt enable register (shown in Figure A-140) 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 UARTRBFIE and/or UARTTBEIE bits in this register are normally set.

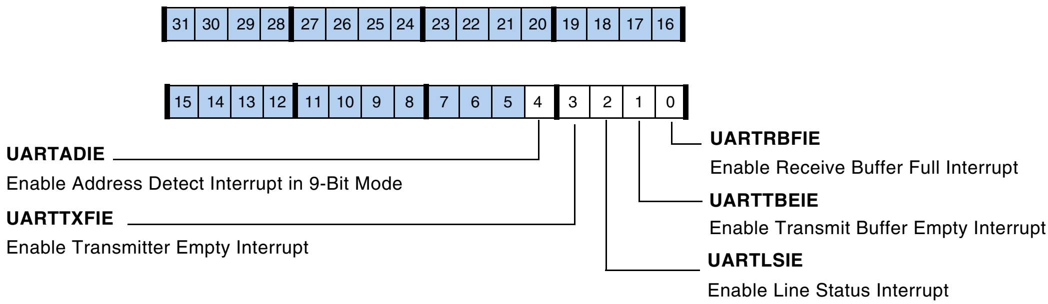

Figure A-140. UART0IER Register

Table A-125. UART0IER Register Bit Descriptions (RW)

Bit	Name	Description
0	UARTRBFIE	**Enable Receive Buffer Full Interrupt.** 0 = No interrupt 1 = Generate RX interrupt if UARTDR bit in UART0LSR is set
1	UARTTBEIE	**Enable Transmit Buffer Empty Interrupt.** 0 = No interrupt 1 = Generate TX interrupt if UARTTHRE bit in UART0LSR is set
2	UARTLSIE	**Enable Line Status Interrupt.** 0 = No interrupt 1 = Generate line status interrupt if any of UART0LSR[4–1] is set
3	UARTTXFIE	**Enable Transmit Empty Interrupt (TEMT = TSR + THR empty).** 0 = No interrupt 1 = Generate TX interrupt if UARTTEMT bit in UART0LSR is set
4	UARTADIE	**Enable Address Detect Interrupt in 9-Bit Mode.** 0 = No interrupt 1 = Generate RX interrupt when address is detected in 9-bit mode

Interrupt Identification Registers (UART0IIR, UART0IIRSH)

For legacy reasons, the UART interrupt identification register (`UART0IIR`, shown in Figure A-141) still reflects the UART interrupt status. Legacy operation may require bundling all UART interrupt sources to a single interrupt channel and servicing them all by the same software routine. For more information, see Appendix 2, Interrupt Control.

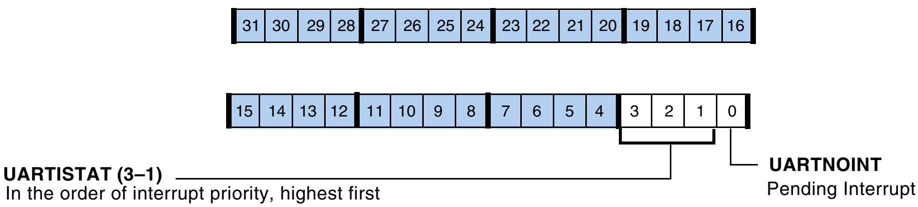

Figure A-141. UART0IIR Register

Table A-126. UART0IIR Register Bit Descriptions (RO)

Bit	Name	Description
0	UARTNOINT	**Pending Interrupt.** When UARTNOINT bit cleared it signals that an interrupt is pending. 0 = Interrupt pending 1 = No interrupt pending (default)
3–1 (ROC[1])	UARTISTAT	**In the Order of Interrupt Priority, Highest First.** 011 = Receive line status. Read UART0LSR to clear interrupt request. 100 = Address detect. Read RBR to clear interrupt request. 010 = Receive data ready. Read UART RBR to clear interrupt request. 001 = UART0THR empty. Write UART0THR or read UART0IIR to clear interrupt request, when priority = 4. 000 = TEMT = transmitter empty (UART THR & TSR empty). Write UART0THR or read UART0IIR to clear interrupt request, when priority = 5. In the case where both interrupts are signalling, the UART0IIR register reads 0x06. When a UART interrupt is pending, the interrupt service routine (ISR) needs to clear the interrupt latch explicitly.

1 These bits are read-only in the UART0IIRSH (shadow) register.

There is also a shadow register, `UART0IIRSH`. This register allows programs to read the contents of the corresponding main register without affecting the status of the UART.

Scratch Register (UART0SCR)

This register (Figure A-142) is used for general-purpose data storage and does not control the UART hardware in any way.

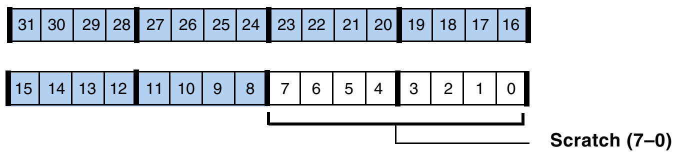

Figure A-142. UART0SCR Registers (RW)

DMA Status Registers (UART0TXSTAT, UART0RXSTAT)

These read-only registers (described in Table A-127 and Table A-128) provide DMA status information.

Table A-127. UART0TXSTAT Register Bit Descriptions (RO)

Bit	Name	Description
0	Reserved	
1	UARTDMASTAT	**DMA Status.** Provides DMA status. 0 = TX DMA is inactive 1 = TX DMA is active
2	UARTCHSTAT	**DMA Chaining Status.** Provides DMA chaining status. 0 = TX DMA chain loading is inactive 1 = TX DMA chain loading is active

Table A-128. UART0RXSTAT Register Bit Descriptions (RO)

Bit	Name	Description
0 (ROC)	UARTERRIRQ	**Receive Channel Error Interrupt.** 0 = No error interrupt 1 = Error interrupt generated due to receive error (parity/overrun/framing). This bit is cleared on a read of the UART0LSR register.
1	UARTDMASTAT	**DMA Status.** Provides DMA status. 0 = RX DMA is inactive 1 = RX DMA is active
2	UARTCHSTAT	**DMA Chaining Status.** Provides DMA chaining status. 0 = RX DMA chain loading is inactive 1 = RX DMA chain loading is active

Two-Wire Interface Registers

The two-wire interface (TWI) registers provide all control and status bits for this peripheral. Status bits can be updated by their respective functional blocks.

Master Internal Time Register (TWIMITR)

The `TWIMITR` register, shown in Figure A-143 and described in Table A-129), is used to enable the TWI module as well as to establish a relationship between the peripheral clock (`PCLK`) and the TWI controller's internally-timed events. The internal time reference is derived from `PCLK` using the prescaled value: PRESCALE = f_{PCLK}/10 MHz

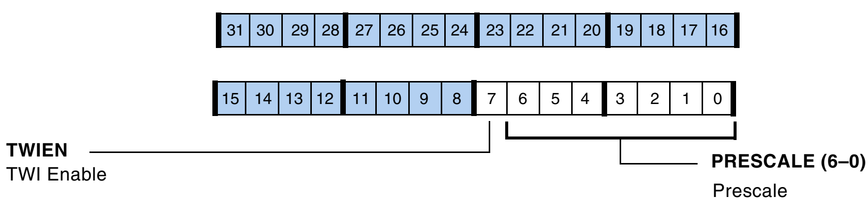

Figure A-143. TWIMITR Register

Table A-129. TWIMITR Register Bit Descriptions (RW)

Bit	Name	Description
0–6	PRESCALE	**Prescale.** The number of peripheral clock (PCLK) 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. This is represented as a 7-bit binary value.
7	TWIEN	**TWI Enable.** This bit must be set for slave or master mode operation. It is recommended that this bit be set at the time PRESCALE is initialized and remain set. This guarantees accurate operation of bus busy detection logic. 0 = Disable TWI 1 = Enable TWI master and slave mode operation. If this bit transitions from high to low the buffer is flushed which takes 2 clock cycles

Clock Divider Register (TWIDIV)

During master mode operation, the SCL clock divider register (shown in Figure A-144 and described in Table A-130) values are used to create the high and low durations of the serial clock (SCL). 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.

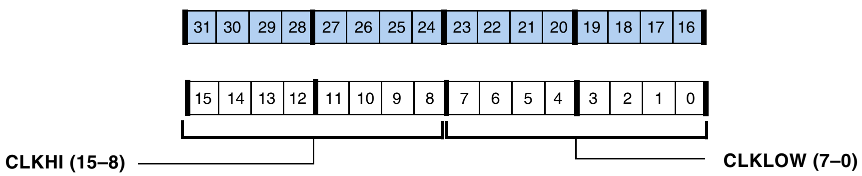

Figure A-144. TWIDIV Register

Table A-130. TWIDIV Register Bit Descriptions (RW)

Bit	Name	Description
7–0	CLKLOW	**Clock Low.** Number of internal time reference periods the serial clock (TWI_CLK) is held low. Represented as an 8-bit binary value.
15–8	CLKHI	**Clock High.** Number of internal time reference periods the serial clock (TWI_CLK) waits before a new clock low period begins (assuming a single master). Represented as an 8-bit binary value.

Master Control Register (TWIMCTL)

The TWI master mode control register (shown in Figure A-145 and described in Table A-131) 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.

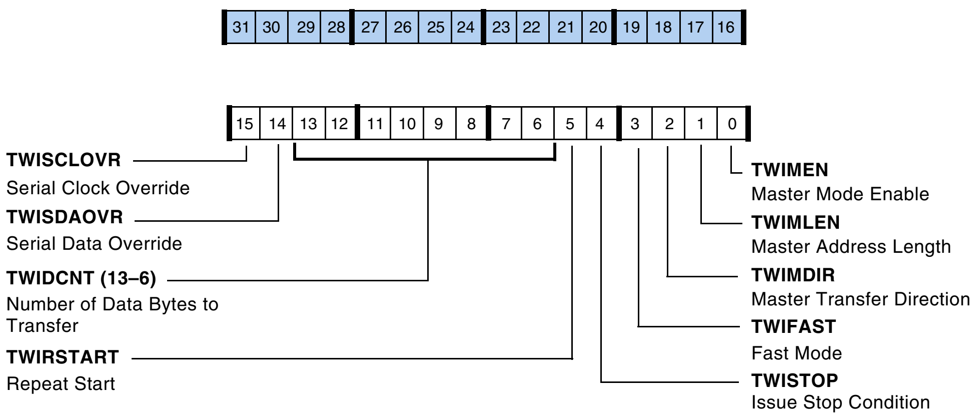

Figure A-145. TWIMCTL Register

Table A-131. TWIMCTL Register Bit Descriptions (RW)

Bit	Name	Description
0	TWIMEN	**Master Mode Enable.** Clears itself at the completion of a transfer. This includes transfers terminated due to errors. 0 = Master mode functionality is disabled. If MEN is cleared during operation, the transfer is aborted and all logic associated with master mode transfers are reset. Serial data and serial clock (TWI_DATA, TWI_CLOCK) are no longer driven. Write 1-to-clear status bits are not effected. 1 = Master mode functionality is enabled. A START condition is generated if the bus is idle.
1	TWIMLEN	**Master Address Length.** 0 = Address is 7-bit 1 = Reserved. Setting this bit to one causes unpredictable behavior.
2	TWIMDIR	**Master Transfer Direction.** 0 = The initiated transfer is master transmit 1 = The initiated transfer is master receive
3	TWIFAST	**Fast Mode.** 0 = Standard mode timing specifications in use (100 kHz) 1 = Fast mode timing specifications in use (400 kHz)
4	TWISTOP	**Issue STOP Condition.** 0 = Normal transfer operation 1 = The transfer concludes as soon as possible avoiding any error conditions (as if data transfer count had been reached) and at that time the interrupt source register is updated along with any associated status bits.
5	TWIRSTART	**Repeat START.** 0 = Transfer concludes with a STOP condition 1 = Rather than issuing a STOP condition at the conclusion of the current transfer (DCNT = 0), issue a repeated START condition, followed by an address byte at the beginning of the next transmission. The current transfer is concluded 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 (MEN) does not clear itself on a repeat start.

Table A-131. TWIMCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
13–6	TWIDCNT	**Data Transfer Count.** Indicates the number of data bytes to transfer. As each data word is transferred, the data transfer count is decremented. When DCNT is zero, a STOP (or restart condition) is issued. Setting DCNT to 0xFF disables the counter. In this transfer mode, data continues to be transferred until it is concluded by setting the STOP bit.
14	TWISDAOVR	**Serial Data (TWI_DATA) Override.** For use when direct control of the Serial Data line is required. Normal master and slave mode operation should not require override operation. 0 = Normal serial data operation under the control of the transmit shift register and acknowledge logic 1 = Serial data output is driven to an active "zero" level, overriding all other logic. This state is held until the bit location is cleared.
15	TWISCLOVR	**Serial Clock (TWI_Clock) Override.** For use when direct control of the serial clock line is required. Normal master and slave mode operation should not require override operation. 0 = Normal serial clock operation under the control of master mode clock generation and slave mode clock stretching logic 1 = Serial clock output is driven to an active "zero" level, overriding all other logic. This state is held until the bit location is cleared.

Master Address Register (TWIMADDR)

During the addressing phase of a transfer, the TWI controller, with its master enabled, transmits the contents of the TWI master mode address register (`TWIMADDR`, shown in Figure A-146). 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 1010000X, then `TWIMADDR` is programmed with 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 `TWIMDIR` bit in the master mode control register.

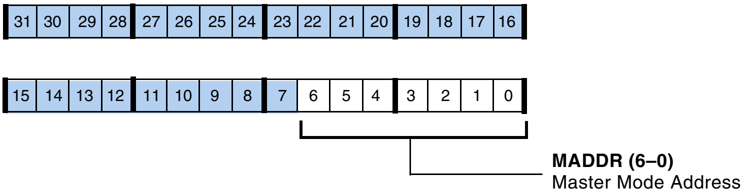

Figure A-146. TWIMADDR Register (RW)

Master Status Register (TWIMSTAT)

The TWI master mode status register (`TWIMSTAT`, shown in Figure A-147 and described in Table A-132) holds information during master mode transfers and at their conclusions. Generally, master mode status bits are not directly associated with the generation of interrupts but offer information on the current transfer. Slave mode operation does not affect master mode status bits.

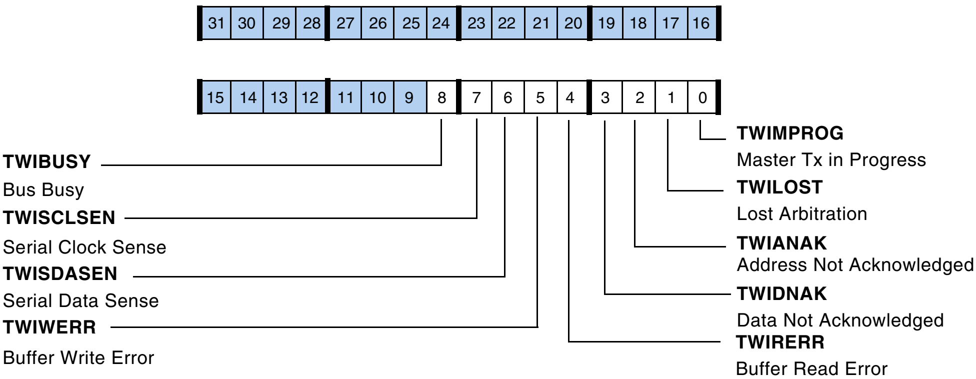

Figure A-147. TWIMSTAT Register

Table A-132. TWIMSTAT Register Bit Descriptions (RO)

Bit	Name	Description
0	TWIMPROG	**Master Transfer In Progress.** 0 = Currently no transfer is taking place. This can occur once a transfer is complete or while an enabled master is waiting for an idle bus. 1 = A master transfer is in progress.
1 (RW1C)	TWILOST	**Lost Arbitration.** 0 = The current transfer has not lost arbitration with another master. 1 = The current transfer was aborted due to the loss of arbitration with another master.
2 (RW1C)	TWIANAK	**Address Not Acknowledged.** 0 = The current master transfer has not detected a NAK during addressing. 1 = The current master transfer was aborted due to the detection of a NAK during the address phase of the transfer.
3 (RW1C)	TWIDNAK	**Data Not Acknowledged.** 0 = The current master transfer has not detected a NAK during data transmission. 1 = The current master transfer was aborted due to the detection of a NAK during data transmission.
4 (RW1C)	TWIRERR	**Buffer Read Error.** 0 = The current master transmit has not detected a buffer read error. 1 = The current master transfer was aborted due to a transmit buffer read error. At the time data was required by the transmit shift register, the buffer was empty.
5 (RW1C)	TWIWERR	**Buffer Write Error.** 0 = The current master receive has not detected a receive buffer write error. 1 = 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.

Table A-132. TWIMSTAT Register Bit Descriptions (RO) (Cont'd)

Bit	Name	Description
6	TWISDASEN	**Serial Data Sense.** For use 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 = An inactive "one" is currently being sensed on serial data line. 1 = An active "zero" is currently being sensed on serial data line. The source of the active driver is not known and can be internal or external.
7	TWISCLSEN	**Serial Clock Sense.** For use 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 = An inactive "one" is currently being sensed on SCLK. 1 = An active "zero" is currently being sensed on SCLK. The source of the active driver is not known and can be internal or external.
8	TWIBUSY	**Bus Busy.** Indicates whether the bus is currently busy or free. This indication applies to all devices. Upon a START condition, setting the register value is delayed due to the input filtering. Upon a STOP condition, clearing the register value occurs after time t_{BUF}. 0 = The bus is free. The clock and data bus signals have been inactive for the appropriate bus free time. 1 = The bus is busy. Clock and/or data activity has been detected.

Slave Mode Control Register (TWISCTL)

The TWI slave mode control register (shown in Figure A-148 and described in Table A-133), 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.

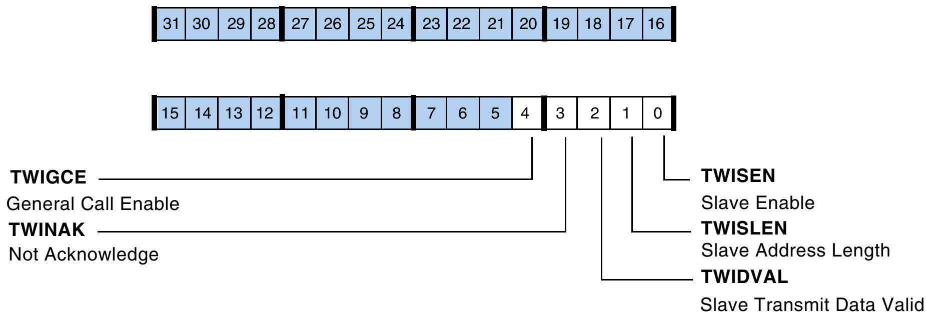

Figure A-148. TWISCTL Register

Table A-133. TWISCTL Register Bit Descriptions (RW)

Bit	Name	Description
0	TWISEN	**Slave Enable.** 0 = The slave is not enabled. No attempt is made to identify a valid address. If cleared during a valid transfer, clock stretching ceases, the serial data line is released and the current byte is not acknowledged. 1 = The slave is enabled. Enabling slave and master modes of operation concurrently is allowed.
1	TWISLEN	**Slave Address Length.** 0 = Address is a 7-bit address 1 = Reserved. Setting this bit to 1 causes unpredictable behavior.
2	TWIDVAL	**Slave Transmit Data Valid.** 0 = Data in the transmit FIFO is for master mode transmits and is not allowed to be used during a slave transmit, and the transmit FIFO is treated as if it is empty. 1 = Data in the transmit FIFO is available for a slave transmission.

Table A-133. TWISCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
3	TWINAK	**Not Acknowledged.** 0 = Slave receive transfer generates an ACK at the conclusion of a data transfer. 1 = Slave receive transfer generates a data NAK at the conclusion of a data transfer. The slave is still considered to be addressed.
4	TWIGCE	**General Call Enable.** General call address detection is available only when slave mode is enabled. 0 = General call address matching is not enabled 1 = General call address matching is enabled. Regardless of the selected address length of slave address, a general call slave receive transfer is accepted. All status and interrupt source bits associated with transfers are updated.

Slave Address Register (TWISADDR)

The TWI slave mode address register (shown in Figure A-149) holds the slave mode address, which is the valid address that the slave-enabled TWI controller responds to. The TWI controller compares this value with the received address during the addressing phase of a transfer.

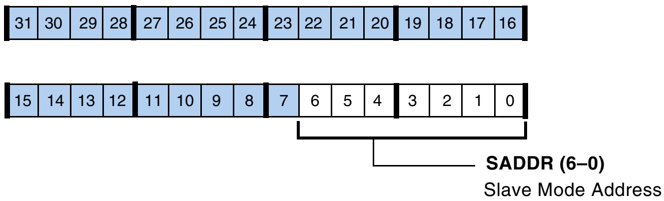

Figure A-149. TWISADDR Register (RW)

Slave Status Register (TWISSTAT)

During and at the conclusion of slave mode transfers, the TWI slave mode status register (shown in Figure A-150) 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 the slave mode status bits.

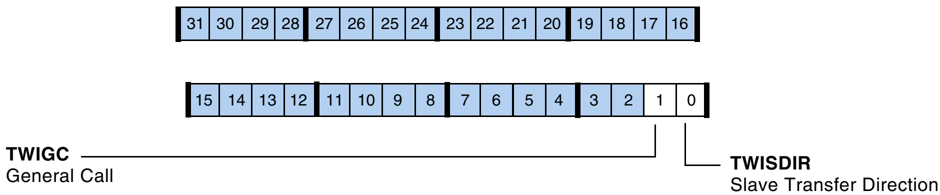

Figure A-150. TWISSTAT Register

Table A-134. TWISSTAT Register Bit Descriptions (RO)

Bit	Name	Description
0	TWISIDR	**Slave Transfer Direction.** This bit self clears if slave mode is disabled (SEN = 0) 0 = At the time of addressing, the transfer direction was determined to be slave receive. 1 = At the time of addressing, the transfer direction was determined to be slave transmit.
1	TWIGC	**General Call.** This bit self clears if slave mode is disabled (SEN = 0) 0 = At the time of addressing, the address was not determined to be a general call. 1 = At the time of addressing, the address was determined to be a general call.

FIFO Control Register (TWIFIFOCTL)

The TWI FIFO control register (shown in Figure A-151 and described in Table A-135) affects only the FIFO and is not tied in any way with master or slave mode operation.

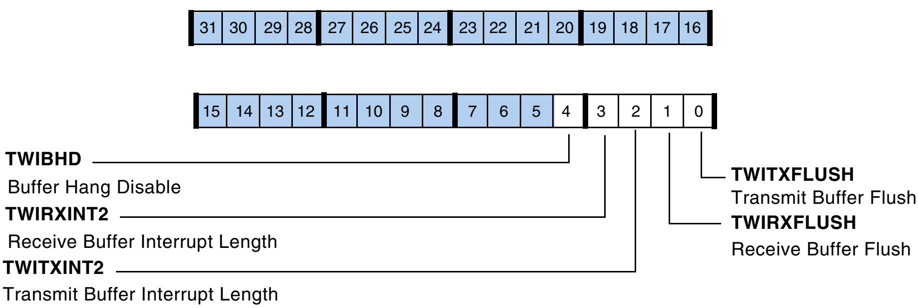

Figure A-151. TWIFIFOCTL Register

Table A-135. TWIFIFOCTL Register Bit Descriptions (RW)

Bit	Name	Description
0	TWITXFLUSH	**Transmit Buffer Flush.** 0 = Normal operation of the transmit buffer and its status bits 1 = Flush the contents of the transmit buffer and update the status 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 as if the transmit buffer is empty.
1	TWIRXFLUSH	**Receive Buffer Flush.** 0 = Normal operation of the receive buffer and its status bits. 1 = Flush the contents of the receive buffer and update the status 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.
2	TWITXINT2	**Transmit Buffer Interrupt Length.** Determines the rate at which transmit buffer interrupts are generated. Interrupts may be generated with each byte transmitted or after two bytes are transmitted. 0 = An interrupt (TWITXINT) is set when TWITXS indicates one or two bytes in the FIFO are empty (01 or 00). 1 = An interrupt (TWITXINT) is set when TWITXS indicates two bytes in the FIFO are empty (00).
3	TWIRXINT2	**Receive Buffer Interrupt Length.** Determines the rate at which receive buffer interrupts are generated. Interrupts may be generated with each byte received or after two bytes are received. 0 = An interrupt (TWIRXINT) is set when TWIRXS indicates one or two bytes in the FIFO are full (01 or 11). 1 = An interrupt (TWIRXINT) is set when TWIRXS indicates two bytes in the FIFO are full (11).
4	TWIBHD	**Receive Buffer Hang Disable.** 0 = Core read of FIFO happens only when Rx FIFO has a valid byte. Write of FIFO happens only when Tx FIFO has at least one empty space. 1 = Core read/write happens irrespective of FIFO status

FIFO Status Register (TWIFIFOSTAT)

The fields in the TWI FIFO status register (shown in Figure A-152 and described in Table A-136) 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.

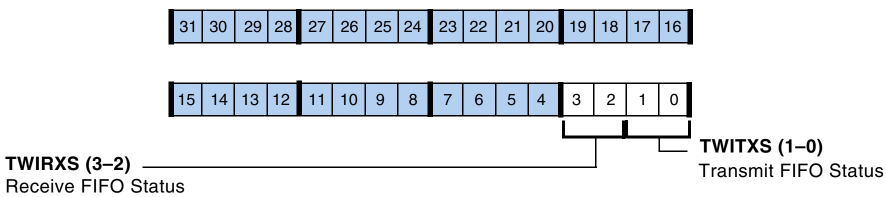

Figure A-152. TWIFIFOSTAT Register

Table A-136. TWIFIFOSTAT Register Bit Descriptions (RO)

Bit	Name	Description
1–0	TWITXS	**Transfer FIFO Status.** These read-only bits indicate 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. 00 = FIFO is empty. Either a single- or double-byte peripheral write of the FIFO goes through immediately. 01 = FIFO contains one byte of data. A single byte peripheral write of the FIFO goes through immediately. A double-byte peripheral write waits until the FIFO is empty 11 = FIFO is full and contains two bytes of data. 10 = Reserved

Table A-136. TWIFIFOSTAT Register Bit Descriptions (RO)

Bit	Name	Description
3–2	TWIRXS	**Receive FIFO Status.** These read-only bits indicate 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. 00 = FIFO is empty. 01 = FIFO contains one byte of data. A single-byte peripheral read of the FIFO goes through immediately. A double-byte peripheral read waits until the FIFO is full. 11 = FIFO is full and contains two bytes of data. Either a single- or double-byte peripheral read of the FIFO is allowed. 10 = Reserved

Interrupt Latch Register (TWIIRPTL)

The TWI interrupt source register (shown in Figure A-153 and described in Table A-137) 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. All bits are sticky and RW1C.

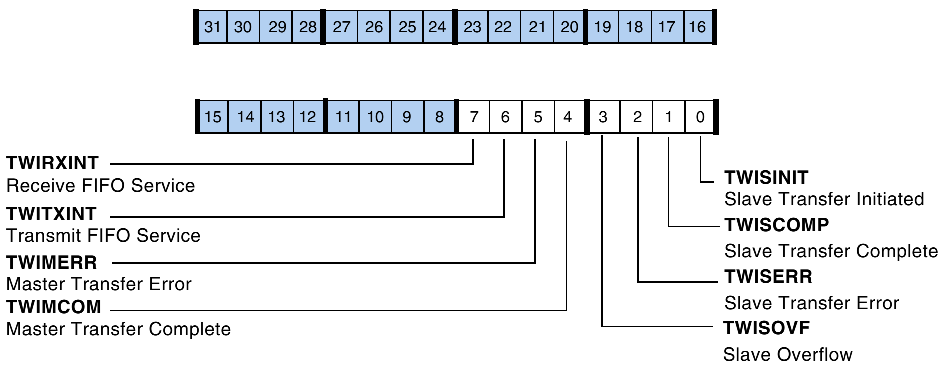

Figure A-153. TWIIRPTL Register

Table A-137. TWIIRPTL Register Bit Descriptions (RW1C)

Bit	Name	Description
0	TWISINIT	**Slave Transfer Initiated.** 0 = A transfer is not in progress. An address match has not occurred since the last time this bit was cleared. 1 = The slave has detected an address match and a transfer has been initiated.
1	TWISCOMP	**Slave Transfer Complete.** 0 = The completion of a transfer not detected 1 = The transfer is complete and either a stop, or a restart was detected.
2	TWISERR	**Slave Transfer Error.** 0 = No errors detected 1 = An error has occurred. A restart or stop condition has occurred during the data receive phase of a transfer.
3	TWISOVF	**Slave Overflow.** 0 = No overflow detected 1 = The slave transfer complete (TWISCOMP) 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.
4	TWIMCOM	**Master Transfer Complete.** 0 = The completion of a transfer not detected 1 = The initiated master transfer is complete. In the absence of a repeat start, the bus is released.
5	TWIMERR	**Master Transfer Error.** 0 = No errors detected 1 = A master error occurred. The conditions surrounding the error are indicated by the master status register (TWIMSTAT).

Table A-137. TWIIRPTL Register Bit Descriptions (RW1C) (Cont'd)

Bit	Name	Description
6	TWITXINT	**Transmit FIFO Service.** If XMTINTLEN2 in the TWIFIFOCTL register is 0, this bit is set each time the TWITXS field in the TWIFIFOSTAT register is updated to either 01 or 00. If XMTINTLEN is 1, this bit is set each time TWITXS is updated to 00. 1 = The transmit FIFO buffer has one or two 8-bit locations available to be written. 0 = FIFO does not require servicing or TWITXS field has not changed since this bit was last cleared.
7	TWIRXINT	**Receive FIFO Service.** If RCVINTLEN2 in the TWIFIFOCTL register is 0, this bit is set each time the TWIRXS field in the TWIFIFOSTAT register is updated to either 01 or 11. If RCVINTLEN2 is 1, this bit is set each time TWIRXS is updated to or 11. 0 = No errors have been detected. 1 = The FIFO does not require servicing or the TWIRXS field has not changed since this bit was last cleared.

Interrupt Enable Register (TWIIMASK)

The TWI interrupt mask register (shown in Figure A-154 and described in Table A-138) enables interrupt sources to assert the interrupt output. Each enable bit corresponds with one interrupt latch bit in the TWI interrupt latch register (`TWIIRPTL`). Reading and writing the `TWIIMASK` register does not affect the contents of the `TWIIRPTL` register.

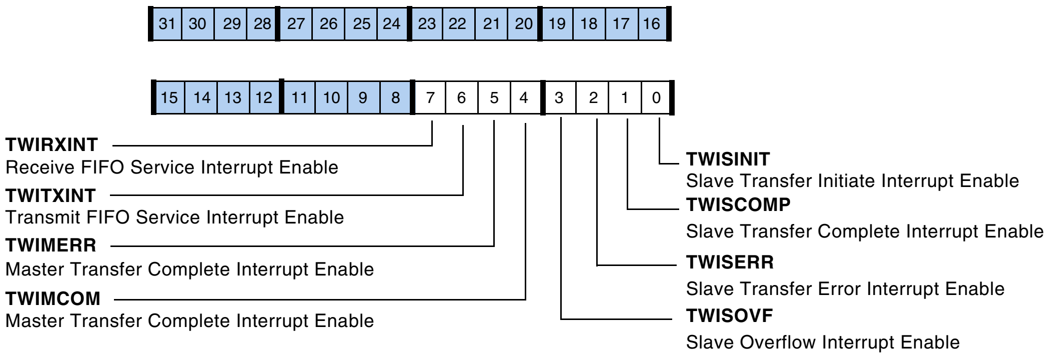

Figure A-154. TWIIMASK Register

Table A-138. TWIIMASK Register Bit Descriptions (RW)

Bit	Name	Description
0	TWISINIT	**Slave Transfer Initiate Interrupt Enable.** 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
1	TWISCOMP	**Slave Transfer Complete Interrupt.** 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
2	TWISERR	**Slave Transfer Error Interrupt.** 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
3	TWISOVF	**Slave Overflow Interrupt Enable.** 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
4	TWIMCOM	**Master Transfer Complete Interrupt Enable.** 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.

Table A-138. TWIIMASK Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
5	TWIMERR	**Master Transfer Error Interrupt Enable.** 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
6	TWITXINT	**Transmit FIFO Service Interrupt Enable.** 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
7	TWIRXINT	**Receive FIFO Service Interrupt Enable.** 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.

Peripheral Timer Registers

The timer peripheral module provides general-purpose timer functionality. It consists of three identical timer units. Each timer has memory-mapped registers. They are described in the following sections.

Read-Modify-Write Timer Control Register

For the timer global control register, the traditional read-modify-write operations to disable a timer have changed. The action is to directly write which simplifies timer enable/disable and can be accomplished with fewer instructions. Example:

Instead of:

```
ustat3=dm(TMCTL);          /* Timer Control Register */
bit set ustat3 TIM1DIS;    /* disables timer 1 */
dm(TMCTL)=ustat3;
```

Use:

```
ustat3 = TIM1DIS;
dm(TMCTL)=ustat3;
```

Writes to the enable and disable bit-pair for a timer works as follows.

`TIMxDIS` = 0, `TIMxEN` = 0 – No action
`TIMxDIS` = 0, `TIMxEN` = 1 – Enable the timer
`TIMxDIS` = 1, `TIMxEN` = x – Disable the timer

For reads, the interpretation is as follows.

`TIM1DIS` = 0, `TIMxEN` = 0 – Timer is disabled
`TIM1DIS` = 1, `TIMxEN` = 1 – Timer is enabled

Any other read combination is not possible. Reading the `TMxCTL` register returns the enable status on both the enable and disable bits.

Timer Control Registers (TMxCTL)

All timer clocks are gated off when the specific timer's configuration register is set to zero at system reset or subsequently reset by user programs. These registers are shown in Figure A-155.

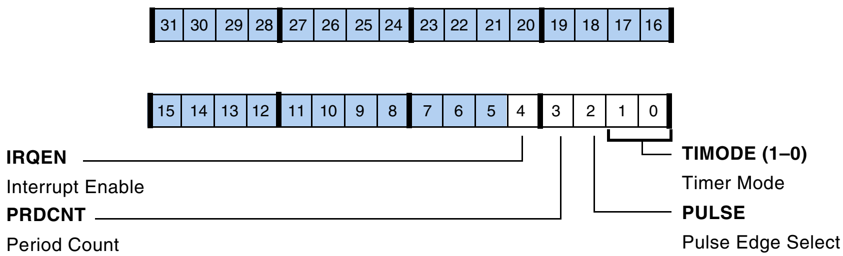

Figure A-155. TMxCTL Register

Table A-139. TMxCTL Register Bit Descriptions (RW)

Bit	Name	Description
1–0	TIMODE	**Timer Mode.** 00 = Reset 01 = PWM_OUT mode (TIMODEPWM) 10 = WDTH_CAP mode (TIMODEW) 11 = EXT_CLK mode (TIMODEEXT)
2	PULSE	**Pulse Edge Select.** 1 = Positive active pulse 0 = Negative active pulse
3	PRDCNT	**Period Count.** 1 = Count to end of period 0 = Count to end of width
4	IRQEN	**Interrupt Enable.** 1 = Enable 0 = Disable

Timer Status Register (TMSTAT)

The global status register TMSTAT is shown in Figure A-156. Status bits are sticky and require a RW1C operation. During a status register read access, all reserved or unused bits return a zero. Each timer generates a unique processor interrupt request signal, TIMxIRQ.

A common status register latches these interrupts. Interrupt bits are sticky and must be cleared to assure that the interrupt is not reissued.

Each timer is provided with its own sticky status register TIMxEN bit. To enable or disable an individual timer, the TIMxEN bit is set or cleared. For example, writing a one to bit 8 sets the TIM0EN bit; writing a one to bit 9 clears it. Writing a one to both bit 8 and bit 9 clears TIM0EN. Reading the status register returns the TIM0EN state on both bit 8 and bit 9. The remaining TIMxEN bits operate similarly.

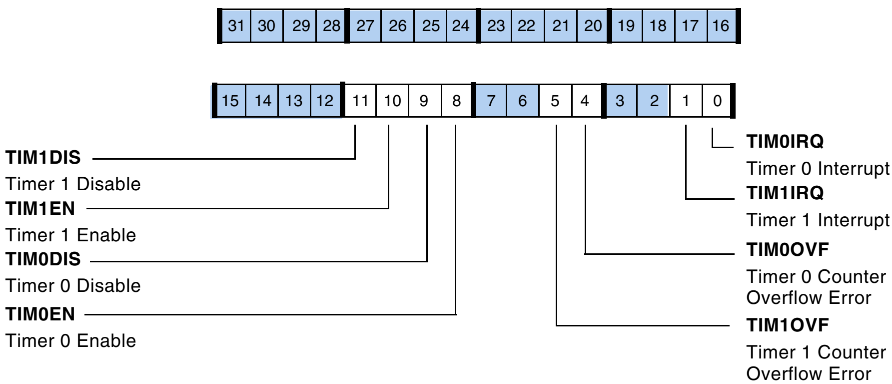

Figure A-156. TMSTAT Register

Table A-140. TMSTAT Register Bit Descriptions (RW)

Bit	Name	Description
0 (RW1C)	TIM0IRQ Timer 0 Interrupt Latch	Also an output
1 (RW1C)	TIM1IRQ Timer 1 Interrupt Latch	Also an output
3–2	Reserved	
4 (RO)	TIM0OVF Timer 0 Overflow/Error	Also an output
5 (RO)	TIM1OVF Timer 1 Overflow/Error	Also an output
7–6	Reserved	
8	TIM0EN Timer 0 Enable	Enable timer 0
9 (RW1C)	TIM0DIS Timer 0 Disable	Disable timer 0
10	TIM1EN Timer 1 Enable	Enable timer 1
10	TIM1EN Timer 1 Disable	Enable timer 1

B REGISTER LISTING

This section lists all available memory mapped IOP registers including the address and reset values.

Power Management and Miscellaneous Registers

Register Mnemonic	Address	Description	Reset
Power Management Register			
PMCTL	0x2000	Power Management Control	Hardware dependent
PMCTL1	0x2001	Power Management Control 1	0
Miscellaneous Registers			
SYSCTL	0x30024	System Control	0x0
RUNRSTCTL	0x2100	Running Reset Control	0x0
REVPID	0x30026	Silicon revision and processor Identification register	Hardware dependent
ROMID	0x20FF	ROM Identification	Hardware dependent

External Port Registers

Register Mnemonic	Address	Description	Reset
External Port Register			
EPCTL	0x1801	External Port Global Control	0xF0
Asynchronous Memory Interface Registers			
AMICTL0	0x1804	AMI Control Register for Bank 1	0x0
AMICTL1	0x1805	AMI Control Register for Bank 2	0x0

Register Mnemonic	Address	Description	Reset
AMICTL2	0x1806	AMI Control Register for Bank 3	0x0
AMICTL3	0x1807	AMI Control Register for Bank 4	0x0
AMISTAT	0x180A	AMI Status	0x1
External Port Direct Memory Access (DMA) Registers			
DMAC0	0x180B	External Port DMA CH 0 Control	0x0
EIEP0	0x1820	External Port CH 0 DMA External Index Address	0x0
EMEP0	0x1821	External Port CH 0 DMA External Modifier	0x0
ECEP0	0x1822	External Port CH 0 DMA External Count	0x0
IIEP0	0x1823	External Port CH 0 DMA Internal Index Address	0x0
IMEP0	0x1824	External Port CH 0 DMA Internal Modifier	0x0
ICEP0	0x1825	External Port CH 0 DMA Internal Count	0x0
CPEP0	0x1826	External Port CH 0 DMA Chain Pointer	0x0
EBEP0	0x1827	External Port CH 0 DMA External Base Address	0x0
TPEP0	0x1828	External Port CH 0 DMA TAP Pointer	0x0
ELEP0	0x1829	External Port CH 0 DMA External Length	0x0
TCEP0	0x182B	External Port CH 0 DMA Delay Line TAP Count	0x0
DFEP0	0x182C	External Port CH 0 DMA Data FIFO	0x0
DMAC1	0x180C	External Port DMA CH 1 Control	0x0
EIEP1	0x1830	External Port CH 1 DMA External Index Address	0x0
EMEP1	0x1831	External Port CH 1 DMA External Modifier	0x0
ECEP1	0x1832	External Port CH 1 DMA External Count	0x0

Register Mnemonic	Address	Description	Reset
IIEP1	0x1833	External Port CH 1 DMA Internal Index Address	0x0
IMEP1	0x1834	External Port CH 1 DMA Internal Modifier	0x0
ICEP1	0x1835	External Port CH 1 DMA Internal Count	0x0
CPEP1	0x1836	External Port CH 1 DMA Chain Pointer	0x0
EBEP1	0x1837	External Port CH 1 DMA External Base Address	0x0
TPEP1	0x1838	External Port CH 1 DMA TAP Pointer	0x0
ELEP1	0x1839	External Port CH 1 DMA External Length	0x0
TCEP1	0x183B	External Port CH 1 DMA Delay Line TAP Count	0x0
DFEP1	0x183C	External Port CH 1 DMA Data FIFO	0x0
DDR2 Registers			
DDR2CTL0	0x1812	DDR2 Control 0	0x1800 0018
DDR2CTL1	0x1813	DDR2 Control 1	0x9452 3466
DDR2CTL2	0x1814	DDR2 Control 2	0x422
DDR2CTL3	0x1815	DDR2 Control 3	0x4780
DDR2CTL4	0x1816	DDR2 Control 4	0x8000
DDR2CTL5	0x1817	DDR2 Control 5	0xC000
DDR2RRC	0x181D	DDR2 Refresh Rate	0x2800614
DDR2STAT0	0x181E	DDR2 Status 0	0x9
DDR2STAT1	0x1840	DDR2 Status 1	0x0
DDR2PADCTL0	0x1841	DDR2 Pad Control 0	0x200 0000
DDR2PADCTL1	0x1842	DDR2 Pad Control 1	0x8020
DLL0CTL1	0x1851	DLL0 Control Register 1	0x0
DLL0STAT0	0x1853	DLL0 Status Register 1	0x0
DLL1CTL1	0x1856	DLL1 Control Register 1	0x0

Register Mnemonic	Address	Description	Reset
DLL1STAT0	0x1858	DLL1 Status Register 1	0x0
Shared Memory DDR2 Registers			
BMAX	0x180D	Bus Maximum Timeout Count	0x0
BCOUNT	0x180E	Bus Current Timeout Count	0x0
SYSTAT	0x180F	Shared Memory Status	0x0
SDRAM Registers			
SDCTL	0x1800	SDRAM Control	0x0102000A
SDRRC	0x1802	SDRAM Refresh Count	0x3081A
SDSTAT0	0x1803	SDRAM Status	0x8
SDSTAT1	0x1843	SDRAM Status	0x0

Serial Port Registers

Register Mnemonic	Address	Description	Reset
Global Serial Port Register			
SPERRSTAT	0x2300	Global SPORT Error Status	0x0
Serial Port Error Control Registers			
SPERRCTL0	0xC18	SPORT0 Error Control	0x0
SPERRCTL1	0xC19	SPORT1 Error Control	0x0
SPERRCTL2	0x418	SPORT2 Error Control	0x0
SPERRCTL3	0x419	SPORT3 Error Control	0x0
SPERRCTL4	0x818	SPORT4 Error Control	0x0
SPERRCTL5	0x819	SPORT5 Error Control	0x0
SPERRCTL6	0x4818	SPORT6 Error Control	0x0
SPERRCTL7	0x4819	SPORT7 Error Control	0x0
SPCTL0	0xC00	SPORT 0 Control Register	0x0000 0000
SPCTLN0	0xC1A	SPORT 0 Control Register 2	0x0000 0000

Register Mnemonic	Address	Description	Reset
DIV0	0xC02	SPORT 0 Divisor for TX/RX SCLK0 and FS0	0x0
SPMCTL0	0xC04	SPORT 0 TDM Control Register	0x0
SP0CS0	0xC05	SPORT 0 TDM Select, CH31–0	0x0
SP0CS1	0xC06	SPORT 0 TDM TX Select, CH63–32	0x0
SP0CS2	0xC07	SPORT 0 TDM TX Select, CH95–64	0x0
SP0CS3	0xC08	SPORT 0 TDM TX Select, CH127–96	0x0
SP0CCS0	0xC0D	SPORT 0 TDM TX Compand Select, CH31–0	0x0
SP0CCS1	0xC0E	SPORT 0 TDM TX Compand Select, CH63–32	0x0
SP0CCS2	0xC0F	SPORT 0 TDM TX Compand Select, CH95–64	0x0
SP0CCS3	0xC10	SPORT 0 TDM TX Compand Select, CH127–96	0x0
TXSP0A	0xC60	SPORT 0A Transmit Data	0x0
RXSP0A	0xC61	SPORT 0A Receive Data	0x0
TXSP0B	0xC62	SPORT 0B Transmit Data	0x0
RXSP0B	0xC63	SPORT 0B Receive Data	0x0
SPCTL1	0xC01	SPORT 1 Control Register	0x0000 0000
SPCTLN1	0xC1B	SPORT 1 Control Register 2	0x0000 0000
DIV1	0xC03	SPORT 1 Divisor for TX/RX SCLK1 and FS1	0x0
SPMCTL1	0xC17	SPORT 1 TDM Control Register	0x0
SP1CS0	0xC09	SPORT 1 TDM Select, CH31–0	0x0
SP1CS1	0xC0A	SPORT 1 TDM RX Select, CH63–32	0x0
SP1CS2	0xC0B	SPORT 1 TDM RX Select, CH95–64	0x0
SP1CS3	0xC0C	SPORT 1 TDM RX Select, CH127–96	0x0
SP1CCS0	0xC11	SPORT 1 TDM RX Compand Select, CH31–0	0x0
SP1CCS1	0xC12	SPORT 1 TDM RX Compand Select, CH63–32	0x0
SP1CCS2	0xC13	SPORT 1 TDM RX Compand Select, CH95–64	0x0
SP1CCS3	0xC14	SPORT 1 TDM RX Compand Select, CH127–96	0x0

Register Mnemonic	Address	Description	Reset
IISP0A	0xC40	Internal Memory DMA Address	0x0
IMSP0A	0xC41	Internal Memory DMA Access Modifier	0x0
CSP0A	0xC42	Contains Number of DMA Transfers Remaining	0x0
CPSP0A	0xC43	Points to Next DMA Parameters	0x0
IISP0B	0xC44	Internal Memory DMA Address	0x0
IMSP0B	0xC45	Internal Memory DMA Access Modifier	0x0
CSP0B	0xC46	Contains Number of DMA Transfers Remaining	0x0
CPSP0B	0xC47	Points to Next DMA Parameters	0x0
IISP1A	0xC48	Internal Memory DMA Address	0x0
IMSP1A	0xC49	Internal Memory DMA Access Modifier	0x0
CSP1A	0xC4A	Contains Number of DMA Transfers Remaining	0x0
CPSP1A	0xC4B	Points to Next DMA Parameters	0x0
IISP1B	0xC4C	Internal Memory DMA Address	0x0
IMSP1B	0xC4D	Internal Memory DMA Access Modifier	0x0
CSP1B	0xC4E	Contains Number of DMA Transfers Remaining	0x0
CPSP1B	0xC4F	Points to Next DMA Parameters	0x0
TXSP1A	0xC64	SPORT 1A Transmit Data	0x0
RXSP1A	0xC65	SPORT 1A Receive Data	0x0
TXSP1B	0xC66	SPORT 1B Transmit Data	0x0
RXSP1B	0xC67	SPORT 1B Receive Data	0x0
SPCTL2	0x400	SPORT 2 Control	0x0000 0000
SPCTL3	0x401	SPORT 3 Control	0x0000 0000
SPCTLN2	0x41A	SPORT 2 Control Register 2	0x0000 0000
SPCTLN3	0x41B	SPORT 3 Control Register 2	0x0000 0000
DIV2	0x402	SPORT 2 Divisor for TX/RX SCLK2 and FS2	0x0
DIV3	0x403	SPORT 3 Divisor for TX/RX SCLK3 and FS3	0x0
SPMCTL2	0x404	SPORTs 2 TDM Control	0x0

Register Mnemonic	Address	Description	Reset
SP2CS0	0x405	SPORT 2 TDM TX Select, CH31–0	0x0
SP2CS1	0x406	SPORT 2 TDM TX Select, CH63–32	0x0
SP2CS2	0x407	SPORT 2 TDM TX Select, CH95–64	0x0
SP2CS3	0x408	SPORT 2 TDM TX Select, CH127–96	0x0
SP3CS0	0x409	SPORT 3 TDM RX Select, CH31–0	0x0
SP3CS1	0x40A	SPORT 3 TDM RX Select, CH63–32	0x0
SP3CS2	0x40B	SPORT 3 TDM RX Select, CH95–64	0x0
SP3CS3	0x40C	SPORT 3 TDM RX Select, CH127–96	0x0
SP2CCS0	0x40D	SPORT 2 TDM TX Compand Select, CH31–0	0x0
SP2CCS1	0x40E	SPORT 2 TDM TX Compand Select, CH63–32	0x0
SP2CCS2	0x40F	SPORT 2 TDM TX Compand Select, CH95–64	0x0
SP2CCS3	0x410	SPORT 2 TDM TX Compand Select, CH127–96	0x0
SP3CCS0	0x411	SPORT 3 TDM RX Compand Select, CH31–0	0x0
SP3CCS1	0x412	SPORT 3 TDM RX Compand Select, CH63–32	0x0
SP3CCS2	0x413	SPORT 3 TDM RX Compand Select, CH95–64	0x0
SP3CCS3	0x414	SPORT 3 TDM RX Compand Select, CH127–96	0x0
SPMCTL3	0x417	SPORT 3 TDM Control	0x0
IISP2A	0x440	Internal Memory DMA Address	0x0
IMSP2A	0x441	Internal Memory DMA Access Modifier	0x0
CSP2A	0x442	Contains Number of DMA Transfers Remaining	0x0
CPSP2A	0x443	Points to Next DMA Parameters	0x0
IISP2B	0x444	Internal Memory DMA Address	0x0
IMSP2B	0x445	Internal Memory DMA Access Modifier	0x0
CSP2B	0x446	Contains Number of DMA Transfers Remaining	0x0
CPSP2B	0x447	Points to Next DMA Parameters	0x0
IISP3A	0x448	Internal Memory DMA Address	0x0

Register Mnemonic	Address	Description	Reset
IMSP3A	0x449	Internal Memory DMA Access Modifier	0x0
CSP3A	0x44A	Contains Number of DMA Transfers Remaining	0x0
CPSP3A	0x44B	Points to Next DMA Parameters	0x0
IISP3B	0x44C	Internal Memory DMA Address	0x0
IMSP3B	0x44D	Internal Memory DMA Access Modifier	0x0
CSP3B	0x44E	Contains Number of DMA Transfers Remaining	0x0
CPSP3B	0x44F	Points to Next DMA Parameters	0x0
TXSP2A	0x460	SPORT 2A Transmit Data	0x0
RXSP2A	0x461	SPORT 2A Receive Data	0x0
TXSP2B	0x462	SPORT 2B Transmit Data	0x0
RXSP2B	0x463	SPORT 2B Receive Data	0x0
TXSP3A	0x464	SPORT 3A Transmit Data	0x0
RXSP3A	0x465	SPORT 3A Receive Data	0x0
TXSP3B	0x466	SPORT 3B Transmit Data	0x0
RXSP3B	0x467	SPORT 3B Receive Data	0x0
SPCTL4	0x800	SPORT 4 Control	0x0000 0000
SPCTL5	0x801	SPORT 5 Control	0x0000 0000
SPCTLN4	0x81A	SPORT 4 Control Register 2	0x0000 0000
SPCTLN5	0x819	SPORT 5 Control Register 2	0x0000 0000
DIV4	0x802	SPORT 4 Divisor for TX/RX SCLK4 and FS4	0x0
DIV5	0x803	SPORT 5 Divisor for TX/RX SCLK5 and FS5	0x0
SPMCTL4	0x804	SPORT 4 TDM Control	0x0
SP4CS0	0x805	SPORT 4 TDM TX Select, CH31–0	0x0
SP4CS1	0x806	SPORT 4 TDM TX Select, CH63–32	0x0
SP4CS2	0x807	SPORT 4 TDM TX Select, CH95–64	0x0
SP4CS3	0x808	SPORT 4 TDM TX Select, CH127–96	0x0
SP5CS0	0x809	SPORT 5 TDM RX Select, CH31–0	0x0

Register Mnemonic	Address	Description	Reset
SP5CS1	0x80A	SPORT 5 TDM RX Select, CH63–32	0x0
SP5CS2	0x80B	SPORT 5 TDM RX Select, CH95–64	0x0
SP5CS3	0x80C	SPORT 5 TDM RX Select, CH127–96	0x0
SP4CCS0	0x80D	SPORT 4 TDM TX Compand Select, CH31–0	0x0
SP4CCS1	0x80E	SPORT 4 TDM TX Compand Select, CH63–32	0x0
SP4CCS2	0x80F	SPORT 4 TDM TX Compand Select, CH95–64	0x0
SP4CCS3	0x810	SPORT 4 TDM TX Compand Select, CH127–96	0x0
SP5CCS0	0x811	SPORT 5 TDM RX Compand Select, CH31–0	0x0
SP5CCS1	0x812	SPORT 5 TDM RX Compand Select, CH63–32	0x0
SP5CCS2	0x813	SPORT 5 TDM RX Compand Select, CH95–64	0x0
SP5CCS3	0x814	SPORT 5 TDM RX Compand Select, CH127–96	0x0
SPMCTL5	0x817	SPORT 5 TDM Control	0x0
IISP4A	0x840	Internal Memory DMA Address	0x0
IMSP4A	0x841	Internal Memory DMA Access Modifier	0x0
CSP4A	0x842	Contains Number of DMA Transfers Remaining	0x0
CPSP4A	0x843	Points to Next DMA Parameters	0x0
IISP4B	0x844	Internal Memory DMA Address	0x0
IMSP4B	0x845	Internal Memory DMA Access Modifier	0x0
CSP4B	0x846	Contains Number of DMA Transfers Remaining	0x0
CPSP4B	0x847	Points to Next DMA Parameters	0x0
IISP5A	0x848	Internal Memory DMA Address	0x0
IMSP5A	0x849	Internal Memory DMA Access Modifier	0x0
CSP5A	0x84A	Contains Number of DMA Transfers Remaining	0x0
CPSP5A	0x84B	Points to Next DMA Parameters	0x0
IISP5B	0x84C	Internal Memory DMA Address	0x0
IMSP5B	0x84D	Internal Memory DMA Access Modifier	0x0

Register Mnemonic	Address	Description	Reset
CSP5B	0x84E	Contains Number of DMA Transfers Remaining	0x0
CPSP5B	0x84F	Points to Next DMA Parameters	0x0
TXSP4A	0x860	SPORT 4A Transmit Data	0x0
RXSP4A	0x861	SPORT 4A Receive Data	0x0
TXSP4B	0x862	SPORT 4B Transmit Data	0x0
RXSP4B	0x863	SPORT 4B Receive Data	0x0
TXSP5A	0x864	SPORT 5A Transmit Data	0x0
RXSP5A	0x865	SPORT 5A Receive Data	0x0
TXSP5B	0x866	SPORT 5B Transmit Data	0x0
RXSP5B	0x867	SPORT 5B Receive Data	0x0
SPCTL6	0x4800	SPORT 6 Control	0x0000 0000
SPCTL7	0x4801	SPORT 7 Control	0x0000 0000
SPCTLN6	0x481A	SPORT 6 Control Register 2	0x0000 0000
SPCTLN7	0x481B	SPORT 7 Control Register 2	0x0000 0000
DIV6	0x4802	SPORT 6 Divisor for TX/RX SCLK6 and FS6	0x0
DIV7	0x4803	SPORT 7 Divisor for TX/RX SCLK7 and FS7	0x0
SPMCTL6	0x4804	SPORT 6 TDM Control	0x0
SP6CS0	0x4805	SPORT 6 TDM TX Select, CH31–0	0x0
SP6CS1	0x4806	SPORT 6 TDM TX Select, CH63–32	0x0
SP6CS2	0x4807	SPORT 6 TDM TX Select, CH95–64	0x0
SP6CS3	0x4808	SPORT 6 TDM TX Select, CH127–96	0x0
SP7CS0	0x4809	SPORT 7 TDM RX Select, CH31–0	0x0
SP7CS1	0x480A	SPORT 7 TDM RX Select, CH63–32	0x0
SP7CS2	0x480B	SPORT 7 TDM RX Select, CH95–64	0x0
SP7CS3	0x480C	SPORT 7 TDM RX Select, CH127–96	0x0
SP6CCS0	0x480D	SPORT 6 TDM TX Compand Select, CH31–0	0x0
SP6CCS1	0x480E	SPORT 6 TDM TX Compand Select, CH63–32	0x0

Register Mnemonic	Address	Description	Reset
SP6CCS2	0x480F	SPORT 6 TDM TX Compand Select, CH95–64	0x0
SP6CCS3	0x4810	SPORT 6 TDM TX Compand Select, CH127–96	0x0
SP7CCS0	0x4811	SPORT 7 TDM RX Compand Select, CH31–0	0x0
SP7CCS1	0x4812	SPORT 7 TDM RX Compand Select, CH63–32	0x0
SP7CCS2	0x4813	SPORT 7 TDM RX Compand Select, CH95–64	0x0
SP7CCS3	0x4814	SPORT 7 TDM RX Compand Select, CH127–96	0x0
SPMCTL7	0x4817	SPORT 7 TDM Control	0x0
IISP6A	0x4840	Internal Memory DMA Address	0x0
IMSP6A	0x4841	Internal Memory DMA Access Modifier	0x0
CSP6A	0x4842	Contains Number of DMA Transfers Remaining	0x0
CPSP6A	0x4843	Points to Next DMA Parameters	0x0
IISP6B	0x4844	Internal Memory DMA Address	0x0
IMSP6B	0x4845	Internal Memory DMA Access Modifier	0x0
CSP6B	0x4846	Contains Number of DMA Transfers Remaining	0x0
CPSP6B	0x4847	Points to Next DMA Parameters	0x0
IISP7A	0x4848	Internal Memory DMA Address	0x0
IMSP7A	0x4849	Internal Memory DMA Access Modifier	0x0
CSP7A	0x484A	Contains Number of DMA Transfers Remaining	0x0
CPSP7A	0x484B	Points to Next DMA Parameters	0x0
IISP7B	0x484C	Internal Memory DMA Address	0x0
IMSP7B	0x484D	Internal Memory DMA Access Modifier	0x0
CSP7B	0x484E	Contains Number of DMA Transfers Remaining	0x0
CPSP7B	0x484F	Points to Next DMA Parameters	0x0
TXSP6A	0x4860	SPORT 6A Transmit Data	0x0
RXSP6A	0x4861	SPORT 6A Receive Data	0x0
TXSP6B	0x4862	SPORT 6B Transmit Data	0x0

Register Mnemonic	Address	Description	Reset
RXSP6B	0x4863	SPORT 6B Receive Data	0x0
TXSP7A	0x4864	SPORT 7A Transmit Data	0x0
RXSP7A	0x4865	SPORT 7A Receive Data	0x0
TXSP7B	0x4866	SPORT 7B Transmit Data	0x0
RXSP7B	0x4867	SPORT 7B Receive Data	0x0

Serial Peripheral Interface Registers

Register Mnemonic	Address	Description	Reset
SPI Registers			
SPICTL	0x1000	SPI Control	0x0400
SPIFLG	0x1001	SPI Flag	0x0F80
SPISTAT	0x1002	SPI Status	0x01
TXSPI	0x1003	SPI Transmit Data	0x0
RXSPI	0x1004	SPI Receive Data	0x0
SPIBAUD	0x1005	SPI Baud Setup	0x0
RXSPI_SHADOW	0x1006	SPI Receive Data Shadow	0x0
IISPI	0x1080	Internal Memory DMA Address	0x0
IMSPI	0x1081	Internal Memory DMA Access Modifier	0x0
CSPI	0x1082	Contains Number of DMA Transfers Remaining	0x0
CPSPI	0x1083	Points to Next DMA Parameters	0x0
SPIDMAC	0x1084	SPI DMA Control	0x0
SPIB Registers			
SPICTLB	0x2800	SPIB Control	0x0400
SPIFLGB	0x2801	SPIB Flag	0x0F00
SPISTATB	0x2802	SPIB Status	0x01
TXSPIB	0x2803	SPIB Transmit Data	0x0

Register Mnemonic	Address	Description	Reset
RXSPIB	0x2804	SPIB Receive Data	0x0
SPIBAUDB	0x2805	SPIB Baud Setup	0x0
RXSPIB_SHADOW	0x2806	SPIB Receive Data Shadow	0x0
IISPIB	0x2880	Internal Memory DMA Address	0x0
IMSPIB	0x2881	Internal Memory DMA Access Modifier	0x0
CSPIB	0x2882	Contains Number of DMA Transfers Remaining	0x0
CPSPIB	0x2883	Points to Next DMA Parameters	0x0
SPIDMACB	0x2884	SPIB DMA Control	0x0

Peripheral Timer Registers

Register Mnemonic	Address	Description	Reset
TMSTAT	0x1400	GP Timer 0/1 Status	0x0
TM0STAT	0x1400	GP Timer 0 Status (Alias of TMSTAT)	0x0
TM0CTL	0x1401	GP Timer 0 Control	0x0
TM0CNT	0x1402	GP Timer 0 Count	0x0
TM0PRD	0x1403	GP Timer 0 Period	0x0
TM0W	0x1404	GP Timer 0 Width	0x0
TM1STAT	0x1408	GP Timer 1 Status (Alias of TMSTAT)	0x0
TM1CTL	0x1409	GP Timer 1 Control	0x0
TM1CNT	0x140A	GP Timer 1 Count	0x0
TM1PRD	0x140B	GP Timer 1 Period	0x0
TM1W	0x140C	GP Timer 1 Width	0x0

DAI/DPI Signal Routing Control Registers

Register Mnemonic	Address	Description	Reset
SRU DAI Routing Registers			
SRU_CLK0	0x2430	SRU Clock Control 0	0x2526 30C2
SRU_CLK1	0x2431	SRU Clock Control 1	0x3DEF 7BDE
SRU_CLK2	0x2432	SRU Clock Control 2	0x3DEF 7BDE
SRU_CLK3	0x2433	SRU Clock Control 3	0x3DEF 7BDE
SRU_CLK4	0x2434	SRU Clock Control 4	0x3DEF 7BDE
SRU_CLK5	0x2435	SRU Clock Control 5	0x3DEF 7BDE
SRU_DAT0	0x2440	SRU Data Control 0	0x0814 4040
SRU_DAT1	0x2441	SRU Data Control 1	0x0F38 B289
SRU_DAT2	0x2442	SRU Data Control 2	0x0000 0450
SRU_DAT3	0x2443	SRU Data Control 3	0x0
SRU_DAT4	0x2444	SRU Data Control 4	0x0
SRU_DAT5	0x2445	SRU Data Control 5	0x0
SRU_DAT6	0x2446	SRU Data Control 6	0x00FB EFBE
SRU_FS0	0x2450	SRU FS Control 0	0x2736 B4E3
SRU_FS1	0x2451	SRU FS Control 1	0x3DEF 7BDE
SRU_FS2	0x2452	SRU FS Control 2	0x3DEF 7BDE
SRU_FS3	0x2453	SRU FS Control 3	0x1EF7BDE
SRU_FS4	0x2454	SRU FS Control 4	0x3DE
SRU_PIN0	0x2460	SRU Pin Control 0	0x04C8 0A94
SRU_PIN1	0x2461	SRU Pin Control 1	0x04E8 4B96
SRU_PIN2	0x2462	SRU Pin Control 2	0x0366 8C98
SRU_PIN3	0x2463	SRU Pin Control 3	0x03A7 14A3
SRU_PIN4	0x2464	SRU Pin Control 4	0x0569 4F9E
SRU_EXT_MISCA	0x2470	SRU External Misc. A Control	0x3DEF 7BDE

Register Mnemonic	Address	Description	Reset
SRU_EXT_MISCB	0x2471	SRU External Misc. B Control	0x3DEF 7BDE
SRU_PBEN0	0x2478	SRU Pin Enable 0	0x0E24 82CA
SRU_PBEN1	0x2479	SRU Pin Enable 1	0x1348 D30F
SRU_PBEN2	0x247A	SRU Pin Enable 2	0x1A55 45D6
SRU_PBEN3	0x247B	SRU Pin Enable 3	0x1D71 F79B
SRU_CLK_SHREG	0x24F1	SRU Shift Register Clock Control	0x0000 0272
SRU_DAT_SHREG	0x24F2	SRU Shift Register Data Control	0x0000 0012
SRU2 DPI Routing Registers			
SRU2_INPUT0	0x1C00	SRU2 Input Signal Routing 0	0x0002 1462
SRU2_INPUT1	0x1C01	SRU2 Input Signal Routing 1	0x1AC0 2C00
SRU2_INPUT2	0x1C02	SRU2 Input Signal Routing 2	0x0
SRU2_INPUT3	0x1C03	SRU2 Input Signal Routing 3	0x0
SRU2_INPUT4	0x1C04	SRU2 Input Signal Routing 4	0x0
SRU2_INPUT5	0x1C05	SRU2 Input Signal Routing 5	0x0
SRU2_PIN0	0x1C10	SRU2 Pin Assignment 0	0x1801 7556
SRU2_PIN1	0x1C11	SRU2 Pin Assignment 1	0x004D B699
SRU2_PIN2	0x1C12	SRU2 Pin Assignment 2	0x0045 0000
SRU2_PBEN0	0x1C20	SRU2 Pin Enable 0	0x0D00 C28B
SRU2_PBEN1	0x1C21	SRU2 Pin Enable 1	0x0021 03CE
SRU2_PBEN2	0x1C22	SRU2 Pin Enable 2	0x0018 5964
DAI Interrupt Registers			
DAI_IMASK_FE	0x2480	DAI Falling Edge Interrupt Mask	0x0
DAI_IMASK_RE	0x2481	DAI Rising Edge Interrupt Mask	0x0
DAI_IRPTL_PRI	0x2484	DAI Interrupt Latch Priority	0x0
DAI_IRPTL_H	0x2488	DAI High Priority Interrupt Latch	0x0
DAI_IRPTL_L	0x2489	DAI Low Priority Interrupt Latch	0x0
DAI_IRPTL_HS	0x248C	Shadow DAI High Priority Interrupt Latch	0x0

Register Mnemonic	Address	Description	Reset
DAI_IRPTL_LS	0x248D	Shadow DAI Low Priority Interrupt Latch	0x0
DPI Interrupt Registers			
DPI_IRPTL	0x1C32	DPI Interrupt Latch	0x0
DPI_IRPTL_SH	0x1C33	DPI Shadow Priority Latch	0x0
DPI_IMASK_FE	0x1C34	DPI Falling Edge Interrupt Mask	0x0
DPI_IMASK_RE	0x1C35	DPI Rising Edge Interrupt Mask	0x0
DAI/DPI Pin Buffer Status Registers			
DAI_PIN_STAT	0x24B9	DAI Pin Status	0x000F FFFF
DPI_PIN_STAT	0x1C31	DPI Pin Status	0x0000 3FFF

DAI/DPI Interrupt Control Registers

Register Mnemonic	Address	Description	Reset
DAI Interrupt Registers			
DAI_IMASK_FE	0x2480	DAI Falling Edge Interrupt Mask	0x0
DAI_IMASK_RE	0x2481	DAI Rising Edge Interrupt Mask	0x0
DAI_IRPTL_PRI	0x2484	DAI Interrupt Latch Priority	0x0
DAI_IRPTL_H	0x2488	DAI High Priority Interrupt Latch	0x0
DAI_IRPTL_L	0x2489	DAI Low Priority Interrupt Latch	0x0
DAI_IRPTL_HS	0x248C	Shadow DAI High Priority Interrupt Latch	0x0
DAI_IRPTL_LS	0x248D	Shadow DAI Low Priority Interrupt Latch	0x0
DPI Interrupt Registers			
DPI_IRPTL	0x1C32	DPI Interrupt Latch	0x0
DPI_IRPTL_SH	0x1C33	DPI Shadow Priority Latch	0x0
DPI_IMASK_FE	0x1C34	DPI Falling Edge Interrupt Mask	0x0
DPI_IMASK_RE	0x1C35	DPI Rising Edge Interrupt Mask	0x0

Programmable Interrupt Priority Control Registers

Register Mnemonic	Address	Description	Reset
PICR0	0x2200	Programmable Interrupt Priority Control 0	0x0A41 8820
PICR1	0x2201	Programmable Interrupt Priority Control 1	0x16A4 A0E6
PICR2	0x2202	Programmable Interrupt Priority Control 2	0x2307 B9AC
PICR3	0x2203	Programmable Interrupt Priority Control 3	0x0000 0012

DAI/DPI Pin Buffer Status Registers

Register Mnemonic	Address	Description	Reset
DAI/DPI Pin Buffer Status Registers			
DAI_PIN_STAT	0x24B9	DAI Pin Status	0x000F FFFF
DPI_PIN_STAT	0x1C31	DPI Pin Status	0x0000 3FFF

Input Data Port Registers

Register Mnemonic	Address	Description	Reset
IDP_CTL0	0x24B0	IDP Control 0	0x0
IDP_CTL1	0x24B2	IDP Control 1	0x0000 FFFF
IDP_CTL2	0x24B3	IDP Control 2	0x0
IDP_PP_CTL	0x24B1	Parallel Data Acquisition Port Control Register	0x0
IDP_FIFO	0x24D0	IDP FIFO Packing Mode	0x0
DAI_STAT0	0x24B8	IDP Status	0x0
DAI_STAT1	0x24BA	IDP Status	0x0
Input Data Port DMA Parameter Registers			
IDP_DMA_I0	0x2400	IDP DMA Channel 0 Index	0x0
IDP_DMA_I1	0x2401	IDP DMA Channel 1 Index	0x0

Register Mnemonic	Address	Description	Reset
IDP_DMA_I2	0x2402	IDP DMA Channel 2 Index	0x0
IDP_DMA_I3	0x2403	IDP DMA Channel 3 Index	0x0
IDP_DMA_I4	0x2404	IDP DMA Channel 4 Index	0x0
IDP_DMA_I5	0x2405	IDP DMA Channel 5 Index	0x0
IDP_DMA_I6	0x2406	IDP DMA Channel 6 Index	0x0
IDP_DMA_I7	0x2407	IDP DMA Channel 7 Index	0x0
IDP_DMA_I0A	0x2408	IDP DMA Channel 0 Index A for Ping Pong DMA	0x0
IDP_DMA_I1A	0x2409	IDP DMA Channel 1 Index A for Ping Pong DMA	0x0
IDP_DMA_I2A	0x240A	IDP DMA Channel 2 Index A for Ping Pong DMA	0x0
IDP_DMA_I3A	0x240B	IDP DMA Channel 3 Index A for Ping Pong DMA	0x0
IDP_DMA_I4A	0x240C	IDP DMA Channel 4 Index A for Ping Pong DMA	0x0
IDP_DMA_I5A	0x240D	IDP DMA Channel 5 Index A for Ping Pong DMA	0x0
IDP_DMA_I6A	0x240E	IDP DMA Channel 6 Index A for Ping Pong DMA	0x0
IDP_DMA_I7A	0x240F	IDP DMA Channel 7 Index A for Ping Pong DMA	0x0
IDP_DMA_I0B	0x2418	IDP DMA Channel 0 Index B for Ping Pong DMA	0x0
IDP_DMA_I1B	0x2419	IDP DMA Channel 1 Index B for Ping Pong DMA	0x0
IDP_DMA_I2B	0x241A	IDP DMA Channel 2 Index B for Ping Pong DMA	0x0
IDP_DMA_I3B	0x241B	IDP DMA Channel 3 Index B for Ping Pong DMA	0x0
IDP_DMA_I4B	0x241C	IDP DMA Channel 4 Index B for Ping Pong DMA	0x0

Register Mnemonic	Address	Description	Reset
IDP_DMA_I5B	0x241D	IDP DMA Channel 5 Index B for Ping Pong DMA	0x0
IDP_DMA_I6B	0x241E	IDP DMA Channel 6 Index B for Ping Pong DMA	0x0
IDP_DMA_I7B	0x241F	IDP DMA Channel 7 Index B for Ping Pong DMA	0x0
IDP_DMA_M0	0x2410	IDP DMA Channel 0 Modify	0x0
IDP_DMA_M1	0x2411	IDP DMA Channel 1 Modify	0x0
IDP_DMA_M2	0x2412	IDP DMA Channel 2 Modify	0x0
IDP_DMA_M3	0x2413	IDP DMA Channel 3 Modify	0x0
IDP_DMA_M4	0x2414	IDP DMA Channel 4 Modify	0x0
IDP_DMA_M5	0x2415	IDP DMA Channel 5 Modify	0x0
IDP_DMA_M6	0x2416	IDP DMA Channel 6 Modify	0x0
IDP_DMA_M7	0x2417	IDP DMA Channel 7 Modify	0x0
IDP_DMA_C0	0x2420	IDP DMA Channel 0 Count	0x0
IDP_DMA_C1	0x2421	IDP DMA Channel 1 Count	0x0
IDP_DMA_C2	0x2422	IDP DMA Channel 2 Count	0x0
IDP_DMA_C3	0x2423	IDP DMA Channel 3 Count	0x0
IDP_DMA_C4	0x2424	IDP DMA Channel 4 Count	0x0
IDP_DMA_C5	0x2425	IDP DMA Channel 5 Count	0x0
IDP_DMA_C6	0x2426	IDP DMA Channel 6 Count	0x0
IDP_DMA_C7	0x2427	IDP DMA Channel 7 Count	0x0
IDP_DMA_PC0	0x2428	IDP DMA Channel 0 Ping Pong Count	0x0
IDP_DMA_PC1	0x2429	IDP DMA Channel 1 Ping Pong Count	0x0
IDP_DMA_PC2	0x242A	IDP DMA Channel 2 Ping Pong Count	0x0
IDP_DMA_PC3	0x242B	IDP DMA Channel 3 Ping Pong Count	0x0
IDP_DMA_PC4	0x242C	IDP DMA Channel 4 Ping Pong Count	0x0
IDP_DMA_PC5	0x242D	IDP DMA Channel 5 Ping Pong Count	0x0

Register Mnemonic	Address	Description	Reset
IDP_DMA_PC6	0x242E	IDP DMA Channel 6 Ping Pong Count	0x0
IDP_DMA_PC7	0x242F	IDP DMA Channel 7 Ping Pong Count	0x0

Precision Clock Generator Registers

Register Mnemonic	Address	Description	Reset
PCG_CTLA0	0x24C0	Precision Clock A Control 0	0x0
PCG_CTLA1	0x24C1	Precision Clock A Control 1	0x0
PCG_CTLB0	0x24C2	Precision Clock B Control 0	0x0
PCG_CTLB1	0x24C3	Precision Clock B Control 1	0x0
PCG_CTLC0	0x24C6	Precision Clock C Control 0	0x0
PCG_CTLC1	0x24C7	Precision Clock C Control 1	0x0
PCG_CTLD0	0x24C8	Precision Clock D Control 0	0x0
PCG_CTLD1	0x24C9	Precision Clock D Control 1	0x0
PCG_PW1	0x24C4	Precision Clock Pulse Width Control 1	0x0
PCG_PW2	0x24CA	Precision Clock Pulse Width Control 2	0x0
PCG_SYNC1	0x24C5	Precision Clock Frame Sync Synchronization 1	0x0
PCG_SYNC2	0x24CB	Precision Clock Frame Sync Synchronization 2	0x0

Pulse Width Modulation Registers

Register Mnemonic	Address	Description	Reset
PWMGCTL	0x3800	PWM Global Control	0x0
PWMGSTAT	0x3801	PWM Global Status	0x0
PWMCTL0	0x3000	PWM Control 0	0x0
PWMSTAT0	0x3001	PWM Status 0	0x0009
PWMPERIOD0	0x3002	PWM Period 0	0x0
PWMDT0	0x3003	PWM Dead Time 0	0x0

Register Mnemonic	Address	Description	Reset
PWMA0	0x3005	PWM Channel A Duty Control 0	0x0
PWMB0	0x3006	PWM Channel B Duty Control 0	0x0
PWMSEG0	0x3008	PWM Output Enable 0	0x0
PWMAL0	0x300A	PWM Channel AL Duty Control 0	0x0
PWMBL0	0x300B	PWM Channel BL Duty Control 0	0x0
PWMDBG0	0x300E	PWM Debug Status 0	0x0
PWMPOL0	0x300F	PWM Output Polarity Select 0	0x00FF
PWMCTL1	0x3010	PWM Control 1	0x0
PWMSTAT1	0x3011	PWM Status 1	0x0009
PWMPERIOD1	0x3012	PWM Period 1	0x0
PWMDT1	0x3013	PWM Dead Time 1	0x0
PWMA1	0x3015	PWM Channel A Duty Control 1	0x0
PWMB1	0x3016	PWM Channel B Duty Control 1	0x0
PWMSEG1	0x3018	PWM Output Enable 1	0x0
PWMAL1	0x301A	PWM Channel AL Duty Control 1	0x0
PWMBL1	0x301B	PWM Channel BL Duty Control 1	0x0
PWMDBG1	0x301E	PWM Debug Status 1	0x0
PWMPOL1	0x301F	PWM Output Polarity Select 1	0x00FF
PWMCTL2	0x3400	PWM Control 2	0x0
PWMSTAT2	0x3401	PWM Status 2	0x0009
PWMPERIOD2	0x3402	PWM Period 2	0x0
PWMDT2	0x3403	PWM Dead Time 2	0x0
PWMA2	0x3405	PWM Channel A Duty Control 2	0x0
PWMB2	0x3406	PWM Channel B Duty Control 2	0x0
PWMSEG2	0x3408	PWM Output Enable 2	0x0
PWMAL2	0x340A	PWM Channel AL Duty Control 2	0x0
PWMBL2	0x340B	PWM Channel BL Duty Control 2	0x0

Register Mnemonic	Address	Description	Reset
PWMDBG2	0x340E	PWM Debug Status 2	0x0
PWMPOL2	0x340F	PWM Output Polarity Select 2	0x00FF
PWMCTL3	0x3410	PWM Control 3	0x0
PWMSTAT3	0x3411	PWM Status 3	0x0009
PWMPERIOD3	0x3412	PWM Period 3	0x0
PWMDT3	0x3413	PWM Dead Time 3	0x0
PWMA3	0x3415	PWM Channel A Duty Control 3	0x0
PWMB3	0x3416	PWM Channel B Duty Control 3	0x0
PWMSEG3	0x3418	PWM Output Enable 3	0x0
PWMAL3	0x341A	PWM Channel AL Duty Control 3	0x0
PWMBL3	0x341B	PWM Channel BL Duty Control 3	0x0
PWMDBG3	0x341E	PWM Debug Status 3	0x0
PWMPOL3	0x341F	PWM Output Polarity Select 3	0x00FF

Memory-to-Memory DMA Registers

Register Mnemonic	Address	Description	Reset
MTMCTL	0x2C01	Memory-to-Memory DMA Control	0x0
IIMTMW	0x2C10	MTM DMA Destination Index	0x0
IIMTMR	0x2C11	MTM DMA Source Index	0x0
IMMTMW	0x2C0E	MTM DMA Destination Modify	0x0
IMMTMR	0x2C0F	MTM DMA Source Modify	0x0
CMTMW	0x2C16	MTM DMA Destination Count	0x0
CMTMR	0x2C17	MTM DMA Source Count	0x0

Hardware Accelerator Registers (FFT/FIR/IIR)

Register Mnemonic	Address	Description	Reset
FFT Accelerator Registers			
FFTCTL1	0x5300	FFT Global Control	0x0
FFTCTL2	0x530C	Channel Control	0x0
FFTMACSTAT	0x5302	MAC Status	0x0
FFTDADDR	0x5303	Debug Address	0x0
FFTDDATA	0x5304	Debug Data	0x0
IIFFT	0x5310	Input Index	0x0
IMFFT	0x5311	Input Modifier	0x0
ICFFT	0x5312	Input Count	0x0
ILFFT	0x5313	Input Length	0x0
IBFFT	0x5314	Input Base	0x0
CPIFFT	0x5315	Input Chain Pointer	0x0
OIFFT	0x5318	Output Index	0x0
OMFFT	0x5319	Output Modifier	0x0
OCFFT	0x531A	Output Count	0x0
OLFFT	0x531B	Output Length	0x0
OBFFT	0x531C	Output Base	0x0
CPOFFT	0x531D	Output Chain Pointer	0x0
FFTDMASTAT	0x5320	DMA Status	0x0
FFTSHDMASTAT	0x5321	DMA Shadow Status	0x0
FIR Accelerator Registers			
FIRCTL1	0x5000	FIR Global Control	0x0
FIRDMASTAT	0x5001	DMA Status Reg	0x0
FIRMACSTAT	0x5002	MAC Status	0x0
FIRDEBUGCTL	0x5004	Debug Control	0x0

Register Mnemonic	Address	Description	Reset
FIRDBGADDR	0x5005	Debug Address	0x0
FIRDBGWRDATA	0x5006	Debug Data Write	0x0
FIRDBGRDDATA	0x5007	Debug Data Read	0x0
FIRCTL2	0x5010	Channel Control	0x0
IIFIR	0x5011	Input Data Index	0x0
IMFIR	0x5012	Input Data Modifier	0x0
ICFIR	0x5013	Input Data Count	0x0
IBFIR	0x5014	Input Data Base	0x0
OIFIR	0x5015	Output Data Index	0x0
OMFIR	0x5016	Output Data Modifier	0x0
OCFIR	0x5017	Output Data Count	0x0
OBFIR	0x5018	Output Data Base	0x0
CIFIR	0x5019	Coefficient Index	0x0
CMFIR	0x501A	Coefficient Modifier	0x0
CCFIR	0x501B	Coefficient Count	0x0
CPFIR	0x501C	Chain Pointer	0x0
IIR Accelerator Registers			
IIRCTL1	0x5200	IIR Global Control	0x0
IIRDMASTAT	0x5201	DMA Status Reg	0x0
IIRMACSTAT	0x5202	MAC Status	0x0
IIRDEBUGCTL	0x5203	Debug Control	0x0
IIRDBGADDR	0x5204	Debug Address	0x0
IIRDBGWRDATA_L	0x5205	Debug Data Write LS 32 Bits	0x0
IIRDBGWRDATA_H	0x5206	Debug Data Write MS 8 Bits	0x0
IIRDBGRDDATA_L	0x5207	Debug Data Read LS 32 Bits	0x0
IIRDBGRDDATA_H	0x5208	Debug Data Read MS 8 Bits	0x0
IIRCTL2	0x5210	Channel Control	0x0

Register Mnemonic	Address	Description	Reset
IIIIR	0x5211	Input Data Index	0x0
IMIIR	0x5212	Input Data Modifier	0x0
ICIIR	0x5213	Input Data Count	0x0
IBIIR	0x5214	Input Data Base	0x0
OIIIR	0x5215	Output Data Index	0x0
OMIIR	0x5216	Output Data Modifier	0x0
OCIIR	0x5217	Output Data Count	0x0
OBIIR	0x5218	Output Data Base	0x0
CIIIR	0x5219	Coefficient Index	0x0
CMIIR	0x521A	Coefficient Modifier	0x0
CCIIR	0x521B	Coefficient Count	0x0
CPIIR	0x521C	Chain Pointer	0x0

S/PDIF Interface Registers

Register Mnemonic	Address	Description	Reset
S/PDIF Transmit Registers			
DITCTL	0x24A0	Digital Interface Transmit Control	0x0
S/PDIF Channel Status Registers			
DITCHANA0	0x24A1	Transmit CH0 Subframe A	0x0
DITCHANA1	0x24D4	Transmit CH1 Subframe A	0x0
DITCHANA2	0x24D5	Transmit CH2 Subframe A	0x0
DITCHANA3	0x24D6	Transmit CH3 Subframe A	0x0
DITCHANA4	0x24D7	Transmit CH4 Subframe A	0x0
DITCHANA5	0x24D8	Transmit CH5 Subframe A	0x0
DITCHANB0	0x24A2	Transmit CH0 Subframe B	0x0
DITCHANB1	0x24DA	Transmit CH1 Subframe B	0x0

Register Mnemonic	Address	Description	Reset
DITCHANB2	0x24DB	Transmit CH2 Subframe B	0x0
DITCHANB3	0x24DC	Transmit CH3 Subframe B	0x0
DITCHANB4	0x24DD	Transmit CH4 Subframe B	0x0
DITCHANB5	0x24DE	Transmit CH5 Subframe B	0x0
S/PDIF User Bit Status Registers			
DITUSRBITA0	0x24E0	Transmit User Bit CH0 Subframe A	0x0
DITUSRBITA1	0x24E1	Transmit User Bit CH1 Subframe A	0x0
DITUSRBITA2	0x24E2	Transmit User Bit CH2 Subframe A	0x0
DITUSRBITA3	0x24E3	Transmit User Bit CH3 Subframe A	0x0
DITUSRBITA4	0x24E4	Transmit User Bit CH4 Subframe A	0x0
DITUSRBITA5	0x24E5	Transmit User Bit CH5 Subframe A	0x0
DITUSRBITB0	0x24E8	Transmit User Bit CH0 Subframe B	0x0
DITUSRBITB1	0x24E9	Transmit User Bit CH1 Subframe B	0x0
DITUSRBITB2	0x24EA	Transmit User Bit CH2 Subframe B	0x0
DITUSRBITB3	0x24EB	Transmit User Bit CH3 Subframe B	0x0
DITUSRBITB4	0x24EC	Transmit User Bit CH4 Subframe B	0x0
DITUSRBITB5	0x24ED	Transmit User Bit CH5 Subframe B	0x0
DITUSRUPD	0x24EF	Transmit User Bit Update	0x0
S/PDIF Receiver Registers			
DIRCTL	0x24A8	Receiver Control	0x0
DIRSTAT	0x24A9	Receiver Status	0x20
DIRCHANL	0x24AA	Receiver Left Channel Status	0x0
DIRCHANR	0x24AB	Receiver Right Channel Status	0x0

Sample Rate Converter Registers

Register Mnemonic	Address	Description	Reset
SRCCTL0	0x2490	SRC0 Control	0x0
SRCCTL1	0x2491	SRC1 Control	0x0
SRCMUTE	0x2492	SRC Mute	0x0
SRCRAT0	0x2498	SRC0 Output to Input Ratio	0x8000 8000
SRCRAT1	0x2499	SRC1 Output to Input Ratio	0x8000 8000

UART Registers

Register Mnemonic	Address	Description	Reset
UART0THR	0x3C00	UART0 Transmit Hold	0x0
UART0RBR	0x3C00	UART0 Receive Buffer	0x0
UART0DLL	0x3C00	UART0 Divisor Latch Low	0x0
UART0IER	0x3C01	UART0 Interrupt Enable	0x0
UART0DLH	0x3C01	UART0 Divisor Latch High	0x0
UART0IIR	0x3C02	UART0 Interrupt ID	0x1
UART0LCR	0x3C03	UART0 Line Control	0x0
UART0MODE	0x3C04	UART0 Mode	0x1
UART0LSR	0x3C05	UART0 Line Status	0x60
UART0SCR	0x3C07	UART0 Scratch	Undefined
UART0IIRSH	0x3C09	UART0 Interrupt ID Shadow	0x1
UART0LSRSH	0x3C0A	UART0 Line Status Shadow	0x60
RXI_UAC0	0x3E00	UART Receive Index Register	0x0
RXM_UAC0	0x3E01	UART Receive Modifier Register	0x0
RXC_UAC0	0x3E02	UART Receive Count Register	0x0
RXCP_UAC0	0x3E03	UART Receive Chain Pointer Register	0x0
TXI_UAC0	0x3F00	UART Transmit Index Register	0x0

Register Mnemonic	Address	Description	Reset
TXM_UAC0	0x3F01	UART Transmit Modifier Register	0x0
TXC_UAC0	0x3F02	UART Receive Count Register	0x0
TXCP_UAC0	0x3F03	UART Transmit Chain Pointer Register	0x0
UART0TXCTL	0x3F04	UART0 DMA TX Control	0x0
UART0RXCTL	0x3E04	UART0 DMA RX Control	0x0
UART0TXSTAT	0x3F05	UART0 DMA TX Status	0x0
UART0RXSTAT	0x3E05	UART0 DMA RX Status	0x0

Two-Wire Interface Registers

Register Mnemonic	Address	Description	Reset
TWIDIV	0x4400	SCL Clock Divider	0x0
TWIMITR	0x4404	Master Internal Time Reference	0x0
TWISCTL	0x4408	Slave Mode Control	0x0
TWISSTAT	0x440C	Slave Mode Status	0x0
TWISADDR	0x4410	Slave Mode Address	0x0
TWIMCTL	0x4414	Master Mode Control	0x0
TWIMSTAT	0x4418	Master Mode Status	0x0
TWIMADDR	0x441C	Master Mode Address	0x0
TWIIRPTL	0x4420	Interrupt Latch	0x0
TWIIMASK	0x4424	Interrupt Mask	0x0
TWIFIFOCTL	0x4428	FIFO Control	0x0
TWIFIFOSTAT	0x442C	FIFO Status	0x0
TXTWI 8	0x4480	8-bit FIFO Transmit	0x0
TXTWI 16	0x4484	16-bit FIFO Transmit	0x0
RXTWI8	0x4488	8-Bit FIFO Receive	0x0
RXTWI16	0x448C	16-Bit FIFO Receive	0x0

Link Port Registers

Register Mnemonic	Address	Description	Reset
LCTL0	0x4C00	Link Port 0 Control	0x0
LSTAT0	0x4C01	Link Port 0 Status	0x0
TXLB0	0x4C08	Link Port 0 TX Buffer	Undefined
RXLB0	0x4C09	Link Port 0 RX Buffer	Undefined
TXLB0_IN_SHADOW	0x4C0D	Link Port 0 Shadow TX Buffer	0x0
TXLB0_OUT_SHADOW	0x4C0C	Link Port 0 Shadow Output Pack	0x0
RXLB0_IN_SHADOW	0x4C0B	Link Port 0 Shadow Input RX Buffer	0x0
RXLB0_OUT_SHADOW	0x4C0A	Link Port 0 Shadow Output Pack	0x0
LSTAT0_SHADOW	0x4C03	Link Port 0 Shadow Status	0x0
IILB0	0x4C18	Link Port 0 Internal Index	0x0
IMLB0	0x4C19	Link Port 0 Internal Modifier	0x0
CLB0	0x4C1A	Link Port 0 Internal Count	0x0
CPLB0	0x4C1B	Link Port 0 Chain Pointer	0x0
LCTL1	0x4C20	Link Port 1 Control	0x0
LSTAT1	0x4C21	Link Port 1 Status	0x0
TXLB1	0x4C28	Link Port 1 TX Buffer	Undefined
RXLB1	0x4C29	Link Port 1 RX Buffer	Undefined
TXLB1_IN_ SHADOW	0x4C2D	Link Port 1 Shadow Input TX Buffer	0x0
TXLB1_OUT_SHADOW	0x4C2C	Link Port 1 Shadow Output Pack	0x0
RXLB1_IN_SHADOW	0x4C2B	Link Port 1 Shadow Input RX Buffer	0x0
RXLB1_OUT_SHADOW	0x4C2A	Link Port 1 Shadow Output Pack	0x0
LSTAT1_SHADOW	0x4C23	Link Port 1 Shadow Status	0x0
IILB1	0x4C38	Link Port 1 Internal Index	0x0
IMLB1	0x4C39	Link Port 1 Internal Modifier	0x0

Register Mnemonic	Address	Description	Reset
CLB1	0x4C3A	Link Port 1 Internal Count	0x0
CPLB1	0x4C3B	Link Port 1 Chain Pointer	0x0

Shift Register Register

Register Mnemonic	Address	Description	Reset
SR_CTL	0x24F0	Shift Register Control	0x0

Watchdog Timer Registers

Register Mnemonic	Address	Description	Reset
WDTCTL	0x5400	Watchdog Timer Control	0x0
WDTCURCNT	0x5401	Watchdog Timer Current Count	0x0
WDTTRIP	0x5402	Watchdog Timer Trip	0x0
WDTCNT	0x5403	Watchdog Timer Count	0x0
WDTSTATUS	0x5404	Watchdog Timer Status	0x0
WDTUNLOCK	0x5405	Watchdog Timer Unlock	0x0
WDTCLKSEL	0x5408	Watchdog Timer Clock Select	0x0

Real-Time Clock Registers

Register Mnemonic	Address	Description	Reset
RTC_CLOCK	0x4CA0	Real-Time Clock Clock (1Hz Register)	Undefined
RTC_ALARM	0x4CA1	Real-Time Clock Alarm (1Hz Register)	Undefined
RTC_CTL	0x4CA2	Real-Time Clock Control	0x0
RTC_STAT	0x4CA3	Real-Time Clock Status	0x0
RTC_STPWTCH	0x4CA4	Real-Time Clock Stopwatch (1Hz Register)	Undefined

Register Mnemonic	Address	Description	Reset
RTC_INIT	0x4CA6	Real-Time Clock Initialization (1Hz Register)	0x0
RTC_INITSTAT	0x4CA7	Real-Time Clock Initialization Status	Undefined

Media Local Bus Registers

Register Mnemonic	Address	Description	Reset
MLB_DCCR	0x4100	MLB Device Control	0x0
MLB_SSCR	0x4101	MLB System Status Configuration	0x0
MLB_SDCR	0x4102	System Data Configuration	0x0
MLB_SMCR	0x4103	System Interrupt Mask	0x00000060
MLB_VCCR	0x4107	Version Control Config (Contains the IP Version)	0x202
MLB_SBCR	0x4108	Synchronous Base Address	0x0
MLB_ABCR	0x4109	Asynchronous Base Address	0x0
MLB_CBCR	0x410A	Control Base Address	0x0
MLB_CICR	0x410C	Channel Interrupt Status	0x0
MLB_CECR0	0x4110	Channel 0 Control	0x0
MLB_CSCR0	0x4111	Channel 0 Status	0x8000 0000
MLB_CCBCR0	0x4112	Channel 0 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR0	0x4113	Channel 0 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR0	0x41A0	Channel 0 Local Channel Buffer Control	0x0040 0000
MLB_CECR1	0x4114	Channel 1 Control	0x0
MLB_CSCR1	0x4115	Channel 1 Status	0x8000 0000
MLB_CCBCR1	0x4116	Channel 1 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR1	0x4117	Channel 1 Next Buffer (TX Buffer in I/O Mode)	0x0

Register Mnemonic	Address	Description	Reset
MLB_LCBCR1	0x41A1	Channel 1 Local Channel Buffer Control	0x0040 0001
MLB_CECR2	0x4118	Channel 2 Control	0x0
MLB_CSCR2	0x4119	Channel 2 Status	0x8000 0000
MLB_CCBCR2	0x411A	Channel 2 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR2	0x411B	Channel 2 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR2	0x41A2	Channel 2 Local Channel Buffer Control	0x0040 0002
MLB_CECR3	0x411C	Channel 3 Control	0x0
MLB_CSCR3	0x411D	Channel 3 Status	0x8000 0000
MLB_CCBCR3	0x411E	Channel 3 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR3	0x411F	Channel 3 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR3	0x41A3	Channel 3 Local Channel Buffer Control	0x0040 0003
MLB_CECR4	0x4120	Channel 4 Control	0x0
MLB_CSCR4	0x4121	Channel 4 Status	0x8000 0000
MLB_CCBCR4	0x4122	Channel 4 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR4	0x4123	Channel 4 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR4	0x41A4	Channel 4 Local Channel Buffer Control	0x0040 0004
MLB_CECR5	0x4124	Channel 5 Control	0x0
MLB_CSCR5	0x4125	Channel 5 Status	0x8000 0000
MLB_CCBCR5	0x4126	Channel 5 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR5	0x4127	Channel 5 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR5	0x41A5	Channel 5 Local Channel Buffer Control	0x0040 0005
MLB_CECR6	0x4128	Channel 6 Control	0x0

Register Mnemonic	Address	Description	Reset
MLB_CSCR6	0x4129	Channel 6 Status	0x8000 0000
MLB_CCBCR6	0x412A	Channel 6 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR6	0x412B	Channel 6 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR6	0x41A6	Channel 6 Local Channel Buffer Control	0x0040 0006
MLB_CECR7	0x412C	Channel 7 Control	0x0
MLB_CSCR7	0x412D	Channel 7 Status	0x8000 0000
MLB_CCBCR7	0x412E	Channel 7 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR7	0x412F	Channel 7 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR7	0x41A7	Channel 7 Local Channel Buffer Control	0x0040 0007
MLB_CECR8	0x4130	Channel 8 Control	0x0
MLB_CSCR8	0x4131	Channel 8 Status	0x8000 0000
MLB_CCBCR8	0x4132	Channel 8 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR8	0x4133	Channel 8 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR8	0x41A8	Channel 8 Local Channel Buffer Control	0x0040 0008
MLB_CECR9	0x4134	Channel 9 Control	0x0
MLB_CSCR9	0x4135	Channel 9 Status	0x8000 0000
MLB_CCBCR9	0x4136	Channel 9 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR9	0x4137	Channel 9 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR9	0x41A9	Channel 9 Local Channel Buffer Control	0x0040 0009
MLB_CECR10	0x4138	Channel 10 Control	0x0
MLB_CSCR10	0x4139	Channel 10 Status	0x8000 0000

Register Mnemonic	Address	Description	Reset
MLB_CCBCR10	0x413A	Channel 10 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR10	0x413B	Channel 10 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR10	0x41AA	Channel 10 Local Channel Buffer Control	0x0040 000A
MLB_CECR11	0x413C	Channel 11 Control	0x0
MLB_CSCR11	0x413D	Channel 11 Status	0x8000 0000
MLB_CCBCR11	0x413E	Channel 11 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR11	0x413F	Channel 11 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR11	0x41AB	Channel 11 Local Channel Buffer Control	0x0040 000B
MLB_CECR12	0x4140	Channel 12 Control	0x0
MLB_CSCR12	0x4141	Channel 12 Status	0x8000 0000
MLB_CCBCR12	0x4142	Channel 12 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR12	0x4143	Channel 12 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR12	0x41AC	Channel 12 Local Channel Buffer Control	0x0040 000C
MLB_CECR13	0x4144	Channel 13 Control	0x0
MLB_CSCR13	0x4145	Channel 13 Status	0x8000 0000
MLB_CCBCR13	0x4146	Channel 13 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR13	0x4147	Channel 13 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR13	0x41AD	Channel 13 Local Channel Buffer Control	0x0040 000D
MLB_CECR14	0x4148	Channel 14 Control	0x0
MLB_CSCR14	0x4149	Channel 14 Status	0x8000 0000
MLB_CCBCR14	0x414A	Channel 14 Current Buffer (RX Buffer in I/O Mode)	0x0

Register Mnemonic	Address	Description	Reset
MLB_CNBCR14	0x414B	Channel 14 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR14	0x41AE	Channel 14 Local Channel Buffer Control	0x0040 000E
MLB_CECR15	0x414C	Channel 15 Control	0x0
MLB_CSCR15	0x414D	Channel 15 Status	0x8000 0000
MLB_CCBCR15	0x414E	Channel 15 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR15	0x414F	Channel 15 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR15	0x41AF	Channel 15 Local Channel Buffer Control	0x0040 000F
MLB_CECR16	0x4150	Channel 16 Control	0x0
MLB_CSCR16	0x4151	Channel 16 Status	0x8000 0000
MLB_CCBCR16	0x4152	Channel 16 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR16	0x4153	Channel 16 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR16	0x41B0	Channel 16 Local Channel Buffer Control	0x0040 0010
MLB_CECR17	0x4154	Channel 17 Control	0x0
MLB_CSCR17	0x4155	Channel 17 Status	0x8000 0000
MLB_CCBCR17	0x4156	Channel 17 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR17	0x4157	Channel 17 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR17	0x41B1	Channel 17 Local Channel Buffer Control	0x0040 0011
MLB_CECR18	0x4158	Channel 18 Control	0x0
MLB_CSCR18	0x4159	Channel 18 Status	0x8000 0000
MLB_CCBCR18	0x415A	Channel 18 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR18	0x415B	Channel 18 Next Buffer (TX Buffer in I/O Mode)	0x0

Register Mnemonic	Address	Description	Reset
MLB_LCBCR18	0x41B2	Channel 18 Local Channel Buffer Control	0x0040 0012
MLB_CECR19	0x415C	Channel 19 Control	0x0
MLB_CSCR19	0x415D	Channel 19 Status	0x8000 0000
MLB_CCBCR19	0x415E	Channel 19 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR19	0x415F	Channel 19 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR19	0x41B3	Channel 19 Local Channel Buffer Control	0x0040 0013
MLB_CECR20	0x4160	Channel 20 Control	0x0
MLB_CSCR20	0x4161	Channel 20 Status	0x8000 0000
MLB_CCBCR20	0x4162	Channel 20 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR20	0x4163	Channel 20 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR20	0x41B4	Channel 20 Local Channel Buffer Control	0x0040 0014
MLB_CECR21	0x4164	Channel 21 Control	0x0
MLB_CSCR21	0x4165	Channel 21 Status	0x8000 0000
MLB_CCBCR21	0x4166	Channel 21 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR21	0x4167	Channel 21 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR21	0x41B5	Channel 21 Local Channel Buffer Control	0x0040 0015
MLB_CECR22	0x4168	Channel 22 Control	0x0
MLB_CSCR22	0x4169	Channel 22 Status	0x8000 0000
MLB_CCBCR22	0x416A	Channel 22 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR22	0x416B	Channel 22 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR22	0x41B6	Channel 22 Local Channel Buffer Control	0x0040 0016
MLB_CECR23	0x416C	Channel 23 Control	0x0

Register Mnemonic	Address	Description	Reset
MLB_CSCR23	0x416D	Channel 23 Status	0x8000 0000
MLB_CCBCR23	0x416E	Channel 23 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR23	0x416F	Channel 23 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR23	0x41B7	Channel 23 Local Channel Buffer Control	0x0040 0017
MLB_CECR24	0x4170	Channel 24 Control	0x0
MLB_CSCR24	0x4171	Channel 24 Status	0x8000 0000
MLB_CCBCR24	0x4172	Channel 24 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR24	0x4173	Channel 24 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR24	0x41B8	Channel 24 Local Channel Buffer Control	0x0040 0018
MLB_CECR25	0x4174	Channel 25 Control	0x0
MLB_CSCR25	0x4175	Channel 25 Status	0x8000 0000
MLB_CCBCR25	0x4176	Channel 25 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR25	0x4177	Channel 25 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR25	0x41B9	Channel 25 Local Channel Buffer Control	0x0040 0019
MLB_CECR26	0x4178	Channel 26 Control	0x0
MLB_CSCR26	0x4179	Channel 26 Status	0x8000 0000
MLB_CCBCR26	0x417A	Channel 26 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR26	0x417B	Channel 26 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR26	0x41BA	Channel 26 Local Channel Buffer Control	0x0040 001A
MLB_CECR27	0x417C	Channel 27 Control	0x0
MLB_CSCR27	0x417D	Channel 27 Status	0x8000 0000

Register Mnemonic	Address	Description	Reset
MLB_CCBCR27	0x417E	Channel 27 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR27	0x417F	Channel 27 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR27	0x41BB	Channel 27 Local Channel Buffer Control	0x0040 001B
MLB_CECR28	0x4180	Channel 28 Control	0x0
MLB_CSCR28	0x4181	Channel 28 Status	0x8000 0000
MLB_CCBCR28	0x4182	Channel 28 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR28	0x4183	Channel 28 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR28	0x41BC	Channel 28 Local Channel Buffer Control	0x0040 001C
MLB_CECR29	0x4184	Channel 29 Control	0x0
MLB_CSCR29	0x4185	Channel 29 Status	0x8000 0000
MLB_CCBCR29	0x4186	Channel 29 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR29	0x4187	Channel 29 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR29	0x41BD	Channel 29 Local Channel Buffer Control	0x0040 001D
MLB_CECR30	0x4188	Channel 30 Control	0x0
MLB_CSCR30	0x4189	Channel 30 Status	0x8000 0000
MLB_CCBCR30	0x418A	Channel 30 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR30	0x418B	Channel 30 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR30	0x41BE	Channel 30 Local Channel Buffer Control	0x0040 001E

C AUDIO FRAME FORMATS

This appendix introduces all the serial timing protocols used for audio inter-chip communications. These formats are listed and their availability in the various peripherals noted in Table C-1.

Table C-1. Audio Format Availability

Frame Format	SPORTs	IDP/SIP	ASRC Input	ASRC Output	S/PDIF Tx	S/PDIF Rx	PCG
Serial	Yes						Yes
I^2S	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Left-justified	Yes	Yes	Yes	Yes	Yes		Yes
Right-justified, 24-bit		Yes	Yes	Yes	Yes		Yes
Right-justified, 20-bit		Yes	Yes	Yes	Yes		Yes
Right-justified, 18-bit		Yes	Yes	Yes	Yes		Yes
Right-justified, 16-bit		Yes	Yes	Yes	Yes		Yes
TDM, 128 channel	Yes		Yes	Yes			Yes

Overview

The following protocols are available in the SHARC processor and are briefly described in this appendix. For complete information on the industry standard protocols, see the specification listings in each section.

- Standard Serial Mode
- Left-justified Mode (Sony format)
- I^2S Mode (Sony/Philips format)
- Time Division Multiplex (TDM) Mode
- MOST Mode
- Right-justified Mode
- S/PDIF (consumer mode)
- EBU/AES3 (professional mode)

Standard Serial Mode

Most processors allow word lengths of 4 to 32 bits to be transmitted or received through their serial ports. For convenience, most AFE (analog front-end) devices operate with 16-bit word lengths for both data and status transfer between the AFE and processor. The serial ports (SPORTs) of most DSPs are designed for full-duplex operation. They differ from the typical serial interface of micro controllers in that they use a frame sync pulse to indicate the start of the data frame. In the case of full duplex asynchronous transfers, two separate FS pulses are used for transmit and receive. The typical micro controller serial interfaces use the serial clock (`SCLK`) as an indicator of serial data, meaning that the `SCLK` is only active when data is valid. The DSP serial interface can operate with a continuous

SCLK, in which the frame synchronization (FS) pulse indicates the start of valid data.

Serial mode allows a flexible timing which can be used in unframed mode or framed mode. In framed mode the user can select between timing for early and late frame sync. Moreover the word order can be selected as LSB or MSB first.

I^{2}S Mode

The Inter-IC-Sound (I^{2}S) bus protocol is a popular 3 wire serial bus standard that was developed to standardize communication across a wide range of peripheral devices. Today the I^{2}S protocol has become the standard method of communicating with consumer and professional audio products.

The I^{2}S protocol provides transmission of 2 channel (stereo) Pulse Code Modulation digital data, where each audio sample is sent MSB first. The following list shows applications that use this format.

- Audio D/A and A/D converters
- PC multimedia audio controllers
- Digital audio transmitters and receivers that support serial digital audio transmission standards, such as AES/EBU, S/PDIF, IEC958, CP-340, and CP-1201
- Digital audio signal processors
- Dedicated digital filter chips
- Sample rate converters

Timing diagrams for I^{2}S, right-justified and left-justified formats can be found in the product-specific data sheet.

The I^2S bus transmits audio data from 8–32 bits and control signals over separate lines. The data line carries two multiplexed data channels—the left channel and the right channel. In I^2S mode, if both channels on a SPORT are set up to transmit, then the SPORT transmits left and right I^2S channels simultaneously. If both channels on a SPORT are set up to receive, the SPORT receives left and right I^2S channels simultaneously. Data is transmitted in MSB-first format.

I^2S consists, as stated above, of a bit clock, a word select and the data line. The bit clock pulses once for each discrete bit of data on the data lines. The bit clock operates at a frequency which is a multiple of the sample rate. The bit clock frequency multiplier depends on number of bits per channel, times the number of channels. For example, CD Audio with a sample frequency of 44.1 kHz and 32 bits of precision per (2) stereo channels has a bit clock frequency of 2.8224 MHz. The word select clock lets the device know whether channel 1 or channel 2 is currently being sent, since I^2S allows two channels to be sent on the same data line.

Transitions on the word select clock also serve as a start-of-word indicator. The word clock line pulses once per sample, so while the bit clock runs at some multiple of the sample frequency, the word clock always matches the sample frequency. For a two channel (stereo) system, the word clock is a square wave, with an equal number of bit clock pulses clocking the data to each channel. In a mono system, the word clock pulses one bit clock length to signal the start of the next word, but is no longer be square. Instead, bit clocking transitions occur with the word clock either high or low.

Note the major difference between I^2S and left/right justified modes is a left MSB data shift by one `SCLK` cycle in relation to the frame.

Standard I^2S data is sent from MSB to LSB, starting at the left edge of the word select clock, with one bit clock delay. This allows both the transmitting and receiving devices to ignore the audio precision of the remote device. If the transmitter is sending 32 bits per channel to a device with only 24 bits of internal precision, the receiver ignores the extra bits of precision by not storing the bits past the 24th bit. Likewise, if the transmitter is sending 16 bits per channel to a receiving device with 24 bits of precision, the receiver zero-fills the missing bits. This feature makes it possible to mix and match components of varying precision without re configuration.

Left-Justified Mode

Left-justified mode (also known as SONY Format) is a mode where in each frame sync cycle two samples of data are transmitted/received—one sample on the high segment of the frame sync, the other on the low segment of the frame sync. Prior to development of the I^2S standard, many manufacturers used a variety of non-standard stereo modes. Some companies continue to use this mode, which is supported by many of today's audio front-end devices.

Programs have control over various attributes of this mode. One attribute is the number of bits (8- to 32-bit word lengths). However, each sample of the pair that occurs on each frame sync must be the same length.

Right-Justified Mode

Right-justified mode is a mode where in each frame sync cycle two samples of data are transmitted/received—one sample on the high segment of the frame sync, the other on the low segment of the frame sync. Prior to development of the I^2S standard, many manufacturers used a variety of non-standard stereo modes. Some companies continue to use this mode, which is supported by many of today's audio front-end devices.

Programs have control over various attributes of this mode. One attribute is the number of bits (8- to 32-bit word lengths). However, each sample of the pair that occurs on each frame sync must be the same length.

TDM Mode

Many applications require multiple I/O channels to implement the desired system functions (such as telephone line and acoustic interfaces). Because most DSPs provide one, or at most two SPORTs, and one of these may be required for interfacing to the host or supervisory processor, it may be impractical, if not impossible, to dedicate a separate SPORT interface to each AFE connection.

The solution is to devise a way to connect a series of serial devices to one SPORT. Different converter manufacturers have approached this task in different ways. In essence, though, there are only two choices; either a time division multiplexing (TDM) approach, where each device is active on the SPORT in a particular time slot, or a cascading approach, where all devices are daisy chained together and data is transferred by shifting it through the chain and then following with a latching signal or a serial protocol. Figure C-1 illustrates a pulsed frame clock for the TDM operation.

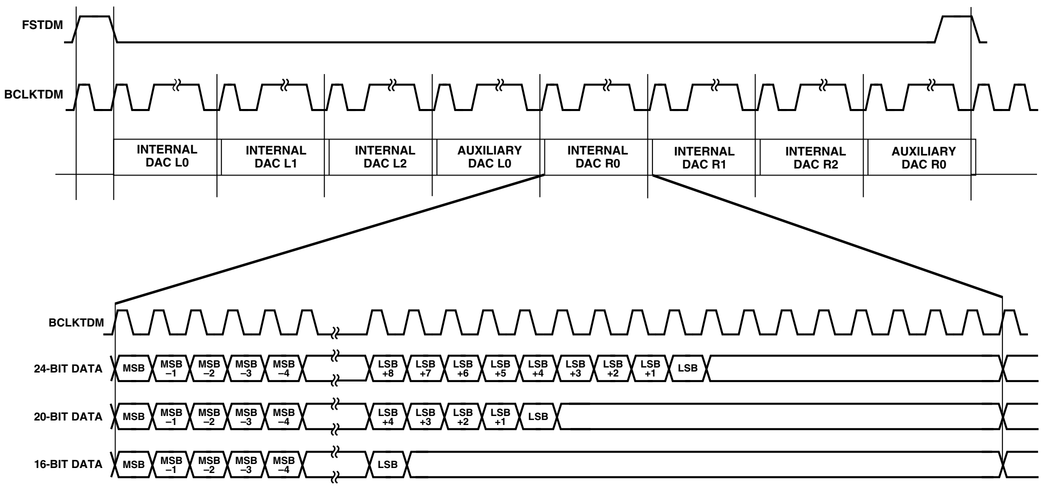

Figure C-1. TDM Mode Timing

Packed I^2S Mode

This mode allows applications to send more than the standard 32 bits per channel normally available through standard I^2S mode. Packed mode is implemented using standard TDM mode. Packed mode supports up to 128 channels (as does TDM mode) as well as the maximum of (128 x 32) bits per left or right channel. As shown in Figure C-2, packed I^2S waveforms are the same as the wave forms used in TDM mode, except that the frame sync is toggled for every frame, and therefore emulates I^2S mode. In other words, packed I^2S mode is a hybrid between TDM and I^2S mode.

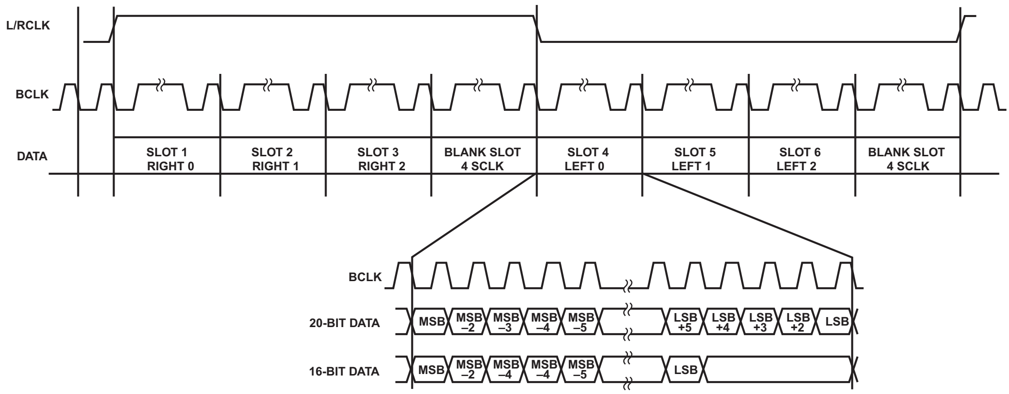

Figure C-2. Packed I2S Mode

MOST Mode

A special packed TDM mode is available that allows four channels to be fit into a space of 64-bit clock cycles. This mode is called packed TDM4 mode, or MOST™ mode. MOST (Media Oriented Systems Transport) is a networking standard intended for interconnecting multimedia components in automobiles and other vehicles. Many integrated circuits intended to interface with a MOST bus use a packed TDM4 data format.

Figure C-3 shows a word length of 16 bits for packed TDM4 mode. This figure is shown with a negative `BCLK` polarity, a negative `LRCLK` polarity, and an MSB delay of 1. The MSB position of the serial data must be delayed by one bit clock from the start of the frame (I^2S position).

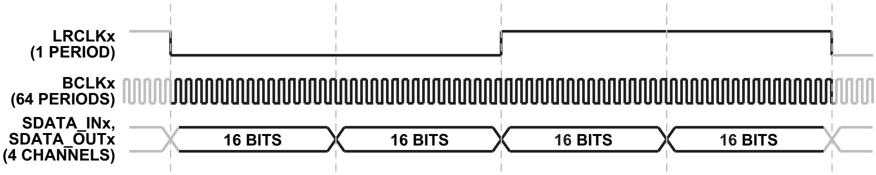

Figure C-3. Packed TDM4 Mode Timing

AES/EBU/SPDIF Formats

For this section, it is important to be familiar with serial digital application interface standards IEC-60958, EIAJ CP-340, AES3 and AES11.

S/PDIF data is transmitted as a stream of 32-bit data words. A data frame consists of 384 words in total, with 192 data words transmitted for the A stereo channel, and 192 data words transmitted for the B stereo channel.

The difference between the AES/EBU and S/PDIF protocol is the channel status bit. If the channel status bit is not set, then:

- 0 = Consumer/professional
- 1 = Normal/compressed data
- 2 = Copy prohibit/copy permit
- 3 = 2 channels/4 channels
- 4 = n/a
- 5 = No pre-emphasis/pre-emphasis

There is one channel status bit in each sub-frame, (comprising of 192 bits per audio block). This translates to 192/8 = 24 bytes available (per audio block). The meaning of the channel status bits are as follows.

- The biphase encoded AES3 stream is composed of subframes (Figure C-5 on page C-11). Subframes consist of a preamble, four auxiliary bits, a 20-bit audio word, a validity bit, a user bit, a channel status bit, and a parity bit.

- The preamble indicates the start of the subframe. The four auxiliary bits normally are the least significant bits of the 24-bit audio word when pasted to the 20-bit audio word. In some cases, the auxiliary bits are used to convey some kind of other data indicated by the channel status bits.

- The validity bit (if cleared, =0) indicates the audio sample word is suitable for direct analog conversion. User data bits may be used in any way desired by the program. The channel status bit conveys information about the status of the channel. Examples of status are length of audio sample words, number of audio channels, sampling frequency, sample address code, alphanumeric source, and destination codes and emphasis. The parity bit is set or cleared to provide an even number of ones and of zeros for time slots 4-31.

- Each frame in the AES3 stream is made up of two subframes. The first subframe is channel A, and the second subframe is channel B. A block is comprised of 192 frames. The channel status is organized into two 192 bit blocks, one for channel A and one for channel B. Normally, the channel status of channel A is equal to channel B. It is extremely rare that they are ever different. Three different preambles are used to indicate the start of a block and the start of channel A or B.

1. Preamble Z indicates the start of a block and the start of subframe channel A

2. Preamble X indicates the start of a channel A subframe when not at the start of a block.

3. Preamble Y indicates the start of a channel B subframe.

The user bits from the channel A and B subframes are simply strung together. For more information, please refer to the AES3 standard.

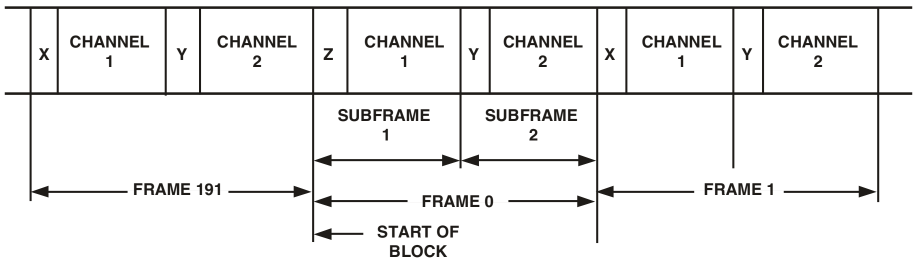

Figure C-4. S/PDIF Block Structure

The data carried by the S/PDIF interface is transmitted serially. In order to identify the assorted bits of information the data stream is divided into frames, each of which are 64 time slots (or 128 unit intervals[1]) in length (Figure C-4). Since the time slots correspond with the data bits, the frame is often described as being 64 bits in length.

A frame is uniquely composed of two subframes. The first subframe normally starts with preamble X. However, the preamble changes to preamble Z once every 192 frames. This defines the block of frames structure used to organize the channel status information. The second subframe always starts with preamble Y.

[1] The unit interval is the minimum time interval between condition changes of a data transmission signal.

Subframe Format

Each frame consists of two subframes. Figure C-5 shows an illustration of a subframe, which consists of 32 time slots numbered 0 to 31. A subframe is 64 unit intervals in length. The first four time slots of each subframe carry the preamble information. The preamble marks the subframe start and identifies the subframe type. The next 24 time slots carry the audio sample data, which is transmitted in a 24-bit word with the least significant bit (LSB) first. When a 20-bit coding range is sufficient, time slots 8 to 27 carry the audio sample word with the LSB in time slot 8. Time slots 4 to 7 may be used for other applications. Under these circumstances, the bits in time slots 4 to 7 are designated auxiliary sample bits. If the source provides fewer bits than the interface allows (either 20 or 24), the unused LSBs are set to logic 0.

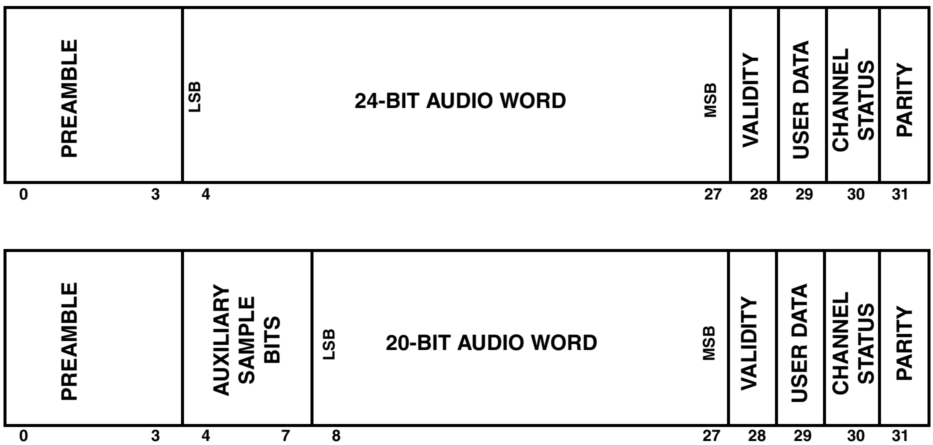

Figure C-5. Subframe Format

This functionality is important when using the S/PDIF receiver in common applications where there are multiple types of data to handle. If there are PCM audio data streams as well as encoded data streams, for example a CD audio stream and a DVD audio stream with encoded data, there is a danger of incorrectly passing the encoded data directly to the DAC. This results in the 'playing' of encoded data as audio, causing loud odd noises to be played. The non-audio flag provides an easy method to mark the this type of data.

After the audio sample word, there are four final time slots which carry:

1. **Validity bit (time slot 28).** The validity bit is logic 0 if the audio sample word is suitable for conversion to an analog audio signal, and logic 1 if it is not. This bit is set if the CHST_BUF_ENABLE bit and the VALIDITY_A (VALIDITY_B for channel 2) bit is set in the SPDIF_TX_CTL register. This bit is also set if the corresponding bit given with the sample is set.

2. **User data bit (time slot 29).** This bit carries user-specified information that may be used in any way. This bit is set if the corresponding bit given with the left/right sample is set.

3. **Channel status bit (time slot 30).** The channel status for each audio signal carries information associated with that audio signal, making it possible for different channel status data to be carried in the two subframes of the digital audio signal. Examples of information to be carried in the channel status are: length of audio sample words, number of audio channels, sampling frequency, sample address code, alphanumeric source and destination codes, and emphasis.

   Channel status information is organized in 192-bit blocks, subdivided into 24 bytes. The first bit of each block is carried in the frame with preamble Z.

4. **Parity bit (time slot 31).** The parity bit indicates that time slots 4 to 31 inclusive will carry an even number of ones and an even number of zeros (even parity). The parity bit is automatically generated for each subframe and inserted into the encoded data.

The two subframes in a frame can be used to transmit two channels of data (channel 1 in subframe 1, channel 2 in subframe 2) with a sample rate equal to the frame rate. Alternatively, the two subframes can carry successive samples of the same channel of data, but at a sample rate that is twice the frame rate. This is called single-channel, double-frequency (SCDF). For more information, see “Data Output Mode” on page 14-12.

Channel Coding

To minimize the direct-current (dc) component on the transmission line, to facilitate clock recovery from the data stream, and to make the interface insensitive to the polarity of connections, time slots 4 to 31 are encoded in bi-phase mark.

Each bit to be transmitted is represented by a symbol comprising two consecutive binary states. The first state of a symbol is always different from the second state of the previous symbol. The second state of the symbol is identical to the first if the bit to be transmitted is logic 0. However, it is different if the bit is logic 1.

Figure C-6 shows that the ones in the original data end up with mid cell transitions in the bi-phase mark encoded data, while zeros in the original data do not. Note that the bi-phase mark encoded data always has a transition between bit boundaries.

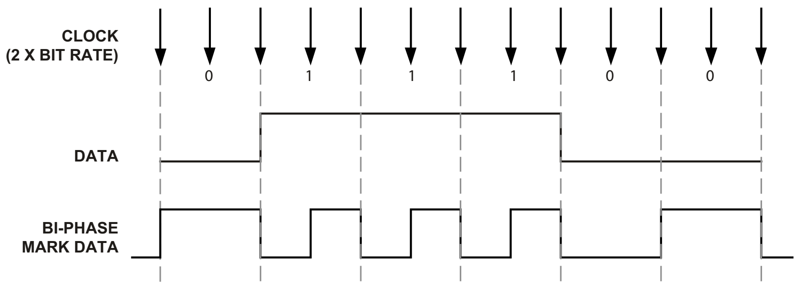

Figure C-6. Bi-phase Mark Encoding

Preambles

Preambles are specific patterns that provide synchronization and identify the subframes and blocks. To achieve synchronization within one sampling period and to make this process completely reliable, these patterns violate the bi-phase mark code rules, thereby avoiding the possibility of data imitating the preambles.

A set of three preambles, shown in Table C-2, are used. These preambles are transmitted in the time allocated to four time slots at the start of each subframe (time slots 0 to 3) and are represented by eight successive states. The first state of the preamble is always different from the second state of the previous symbol (representing the parity bit).

Table C-2. Preambles

Preamble	Preceding state 0	Preceding state 1	Description
X	11100010	00011101	Subframe 1
Y	11100100	00011011	Subframe 2
Z	11101000	00010111	Subframe 1 and block start

Like bi-phase code, the preambles are dc free and provide clock recovery. They differ in at least two states from any valid bi-phase sequence.

I INDEX

Numerics

A

B

D

R

T