ADSP-2126x SHARC® Processor Peripherals Manual

Revision 3.0, December 2005

Part Number
82-002002-01

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

ANALOG DEVICES

Copyright Information

Printed in the USA.

Disclaimer

Trademark and Service Mark Notice

CONTENTS

PREFACE

SERIAL PERIPHERAL INTERFACE PORT

INPUT DATA PORT

DIGITAL AUDIO INTERFACE

REGISTERS REFERENCE

INDEX

PREFACE

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

Purpose of This Manual

The *ADSP-2126x SHARC Processor Peripherals Manual* contains information about the DSP architecture and DSP assembly language for SHARC processors. These are 32-bit, fixed- and floating-point digital signal processors from Analog Devices for use in computing, communications, and consumer applications.

The manual provides information on how assembly instructions execute on the SHARC processor's architecture along with reference information about DSP operations.

Intended Audience

The primary audience for this manual is a programmer who is familiar with Analog Devices processors. This manual assumes that 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 the appropriate hardware reference manuals and data sheets) that describe your target architecture.

Manual Contents

The manual consists of:

- Chapter 1, "Introduction"
  Provides an architectural overview of the ADSP-2126x processor.
- Chapter 2, "I/O Processor"
  Describes ADSP-2126x input/output processor architecture.
- Chapter 3, "Parallel Port"
  Describes the processor's on-chip DMA controller as a mechanism for transferring data without core interruption.
- Chapter 4, "Serial Ports"
  Describes the six dual 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 5, "Serial Peripheral Interface Port"
  Describes the operation of the 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 6, "Input Data Port"
  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 7, "Digital Audio Interface"
  Provides information about the digital audio interface (DAI) which allows you to attach an arbitrary number and variety of peripherals to the ADSP-2126x while retaining high levels of compatibility.
- Chapter 8, "Precision Clock Generator"
  Details the precision clock generators (PCG) each of which generates a pair of signals derived from a clock input signal.

- Chapter 9, "System Design"
  Describes system features of the ADSP-2126x processor. These include power, reset, clock, JTAG, and booting, as well as pin descriptions and other system level information.
- Appendix A, "Registers Reference"
  Provides 'at-a-glance' register figures and bit descriptions.

This hardware reference is a companion document to the *ADSP-2126x SHARC Processor Core Manual*.

What's New in This Manual

Revision 3.0 of the *ADSP-2126x SHARC Processor Peripherals Manual* differs in a number of ways from the revision 2.0 book. In revision 3.0 all errata reports against the previous revision have been corrected.

Technical or Customer Support

You can reach Analog Devices, Inc. Customer Support in the following ways:

- Visit the Embedded Processing and DSP products Web site at
  `http://www.analog.com/processors/technicalSupport`
- E-mail tools questions to
  `processor.tools.support@analog.com`
- E-mail processor questions to
  `processor.support@analog.com (World wide support)`
  `processor.europe@analog.com (Europe support)`
  `processor.china@analog.com (China support)`
- Phone questions to **1-800-ANALOGD**

- Contact your Analog Devices, Inc. local sales office or authorized distributor
- Send questions by mail to:

```
Analog Devices, Inc.
One Technology Way
P.O. Box 9106
Norwood, MA 02062-9106
USA
```

Supported Processors

The following is the list of Analog Devices, Inc. processors supported in VisualDSP++®.

TigerSHARC® (ADSP-TSxxx) Processors

The name *TigerSHARC* refers to a family of floating-point and fixed-point (8-bit, 16-bit, and 32-bit) processors. VisualDSP++ currently supports the following TigerSHARC families: ADSP-TS101 and ADSP-TS20x.

SHARC (ADSP-21xxx) Processors

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

Blackfin® (ADSP-BFxxx) Processors

The name *Blackfin* refers to a family of 16-bit, embedded processors. VisualDSP++ currently supports the following Blackfin families: ADSP-BF53x and ADSP-BF56x.

Product Information

You can obtain product information from the Analog Devices Web site, from the product CD-ROM, or from the printed publications (manuals).

Analog Devices is online at www.analog.com. Our Web site provides information about a broad range of products—analog integrated circuits, amplifiers, converters, and digital signal processors.

MyAnalog.com

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

Registration

Visit www.myanalog.com to sign up. Click **Register** to use MyAnalog.com. Registration takes about five minutes and serves as a means to select the information you want to receive.

If you are already a registered user, just log on. Your user name is your e-mail address.

Processor Product Information

For information on embedded processors and DSPs, visit our Web site at www.analog.com/processors, which provides access to technical publications, data sheets, application notes, product overviews, and product announcements.

You may also obtain additional information about Analog Devices and its products in any of the following ways.

- E-mail questions or requests for information to
  `processor.support@analog.com (World wide support)`
  `processor.europe@analog.com (Europe support)`
  `processor.china@analog.com (China support)`
- Fax questions or requests for information to
  1-781-461-3010 (North America)
  +49-89-76903-157 (Europe)
- Access the FTP Web site at
  `ftp ftp.analog.com` (or `ftp 137.71.25.69`)
  `ftp://ftp.analog.com`

Related Documents

The following publications that describe the ADSP-2126x processor (and related processors) can be ordered from any Analog Devices sales office:

- *ADSP-21261 SHARC Processor Data Sheet*
- *ADSP-21262 SHARC Processor Data Sheet*
- *ADSP-21266 SHARC Processor Data Sheet*
- *ADSP-21267 SHARC Processor Data Sheet*
- *ADSP-2126x SHARC Processor Core Manual*
- *ADSP-21160 SHARC DSP Instruction Set Reference*

For information on product related development software and Analog Devices processors, see these publications:

- *VisualDSP++ User's Guide for SHARC Processors*
- *VisualDSP++ C/C++ Compiler and Library Manual for SHARC Processors*
- *VisualDSP++ Assembler and Preprocessor Manual for SHARC Processors*
- *VisualDSP++ Linker and Utilities Manual for SHARC Processors*
- *VisualDSP++ Kernel (VDK) User's Guide*

Visit the Technical Library Web site to access all processor and tools manuals and data sheets:

`http://www.analog.com/processors/technical_library`

Online Technical Documentation

Online documentation comprises the VisualDSP++ Help system, software tools manuals, hardware tools manuals, processor manuals, the Dinkum Abridged C++ library, and Flexible License Manager (FlexLM) network license manager software documentation. You can easily search across the entire VisualDSP++ documentation set for any topic of interest. For easy printing, supplementary `.PDF` files of most manuals are also provided.

Each documentation file type is described as follows.

File	Description
.CHM	Help system files and manuals in Help format
.HTM or .HTML	Dinkum Abridged C++ library and FlexLM network license manager software documentation. Viewing and printing the HTML files requires a browser, such as Internet Explorer 4.0 (or higher).
.PDF	VisualDSP++ and processor manuals in Portable Documentation Format (PDF). Viewing and printing the .PDF files requires a PDF reader, such as Adobe Acrobat Reader (4.0 or higher).

If documentation is not installed on your system as part of the software installation, you can add it from the VisualDSP++ CD-ROM at any time by running the Tools installation. Access the online documentation from the VisualDSP++ environment, Windows® Explorer, or the Analog Devices Web site.

Accessing Documentation From VisualDSP++

From the VisualDSP++ environment:

- Access VisualDSP++ online Help from the Help menu's **Contents**, **Search**, and **Index** commands.
- Open online Help from context-sensitive user interface items (toolbar buttons, menu commands, and windows).

Accessing Documentation From Windows

In addition to any shortcuts you may have constructed, there are many ways to open VisualDSP++ online Help or the supplementary documentation from Windows.

Help system files (.CHM) are located in the Help folder, and .PDF files are located in the Docs folder of your VisualDSP++ installation CD-ROM. The Docs folder also contains the Dinkum Abridged C++ library and the FlexLM network license manager software documentation.

Using Windows Explorer

- Double-click the vdsp-help.chm file, which is the master Help system, to access all the other .CHM files.
- Double-click any file that is part of the VisualDSP++ documentation set.

Using the Windows Start Button

- Access VisualDSP++ online Help by clicking the **Start** button and choosing **Programs**, **Analog Devices**, **VisualDSP++**, and **VisualDSP++ Documentation**.
- Access the .PDF files by clicking the **Start** button and choosing **Programs**, **Analog Devices**, **VisualDSP++**, **Documentation for Printing**, and the name of the book.

Accessing Documentation From the Web

Download manuals at the following Web site:
http://www.analog.com/processors/technical_library

Select a processor family and book title. Download archive (.ZIP) files, one for each manual. Use any archive management software, such as WinZip, to decompress downloaded files.

Printed Manuals

For general questions regarding literature ordering, call the Literature Center at **1-800-ANALOGD** (**1-800-262-5643**) and follow the prompts.

VisualDSP++ Documentation Set

To purchase VisualDSP++ manuals, call **1-603-883-2430**. The manuals may be purchased only as a kit.

If you do not have an account with Analog Devices, you are referred to Analog Devices distributors. For information on our distributors, log onto http://www.analog.com/salesdir.

Hardware Tools Manuals

To purchase EZ-KIT Lite® and In-Circuit Emulator (ICE) manuals, call **1-603-883-2430**. The manuals may be ordered by title or by product number located on the back cover of each manual.

Processor Manuals

Hardware reference and instruction set reference manuals may be ordered through the Literature Center at **1-800-ANALOGD** (**1-800-262-5643**), or downloaded from the Analog Devices Web site. Manuals may be ordered by title or by product number located on the back cover of each manual.

Data Sheets

All data sheets (preliminary and production) may be downloaded from the Analog Devices Web site. Only production (final) data sheets (Rev. 0, A, B, C, and so on) can be obtained from the Literature Center at **1-800-ANALOGD** (**1-800-262-5643**); they also can be downloaded from the Web site.

To have a data sheet faxed to you, call the Analog Devices Faxback System at **1-800-446-6212**. Follow the prompts and a list of data sheet code numbers will be faxed to you. If the data sheet you want is not listed, check for it on the Web site.

Conventions

Text conventions used in this manual are identified and described as follows.

Example	Description
Close command (**File** menu)	Titles in reference sections indicate the location of an item within the VisualDSP++ environment's menu system (for example, the **Close** command appears on the **File** menu).
`{this | that}`	Alternative items in syntax descriptions appear within curly brackets and separated by vertical bars; read the example as `this` or `that`. One or the other is required.
`[this | that]`	Optional items in syntax descriptions appear within brackets and separated by vertical bars; read the example as an optional `this` or `that`.
`[this,…]`	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipse; read the example as an optional comma-separated list of `this`.
`.SECTION`	Commands, directives, keywords, and feature names are in text with `letter gothic` font.
`filename`	Non-keyword placeholders appear in text with italic style format.
	**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.

Additional conventions, which apply only to specific chapters, may appear throughout this document.

1 INTRODUCTION

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 probability of overflow, using a floating-point DSP can ease 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 ADSP-2126x processors are highly integrated, lower cost 32-bit floating-point DSPs which provide many of these design advantages.

For brevity, the ADSP-21262, ADSP-21266 and ADSP-21267 SHARC processors will be referred to as the ADAP-2126x. For instances where functionality applies to one or the other processor specifically, it will be noted in the text.

ADSP-2126x Processor Design Advantages

The ADSP-2126x processor is a high performance 32-bit processor used for medical imaging, communications, military, audio, test equipment, 3D graphics, speech recognition, motor control, imaging, and other applications. By adding a dual-ported 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-21000 Family processor core to form a complete system-on-a-chip.

The SHARC processor architecture balances a high performance processor core with high performance buses (PM, DM, I/O). In the core, every instruction can execute in a single cycle. The buses and instruction cache provide rapid, unimpeded data flow to the core to maintain the execution rate.

Figure 1-1 shows a detailed block diagram of the processor, illustrating the following architectural features:

- Two processing elements (PEx and PEy), each containing 32-bit IEEE floating-point computation units—multiplier, ALU, shifter, and data register file
- Program sequencer with related instruction cache, interval timer, and Data Address Generators (DAG1 and DAG2)
- Dual-ported SRAM
- Input/Output (I/O) processor with integrated DMA controller, SPI-compatible port, and serial ports for point-to-point multiprocessor communications
- JTAG Test Access Port for emulation
- Parallel port for interfacing to off-chip memory and peripherals

Figure 1-1 also shows the three on-chip buses of the ADSP-2126x processor: the Program Memory (PM) bus, Data Memory (DM) bus, and Input/Output (I/O) bus. The PM bus provides access to either instructions or data. During a single cycle, these buses let the processor access two data operands from memory, access an instruction (from the cache), and perform a DMA transfer.

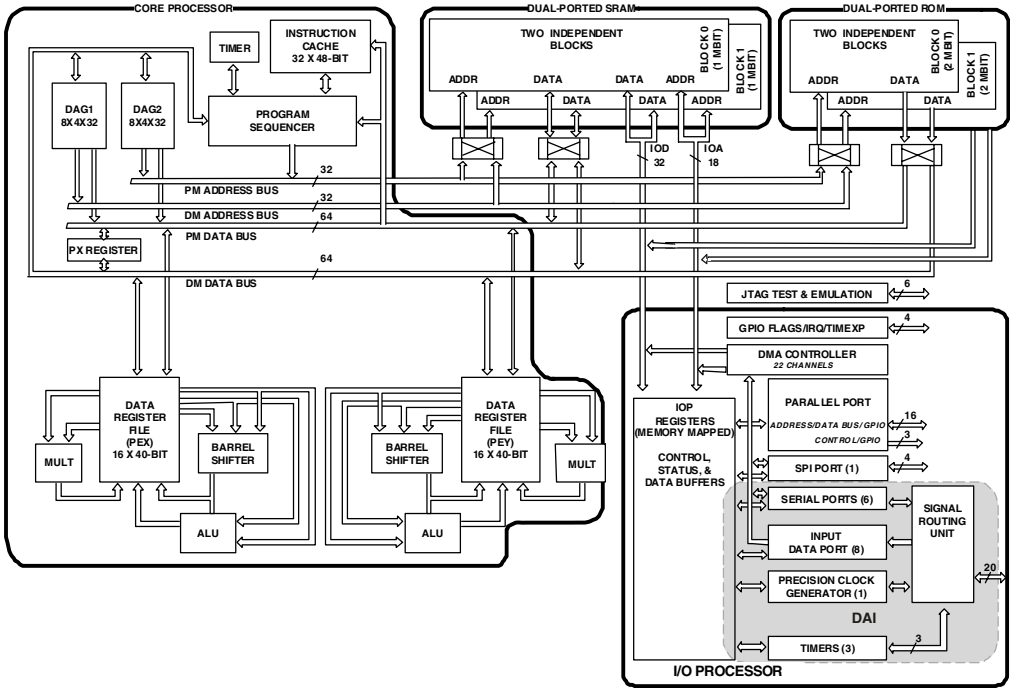


Figure 1-1. ADSP-2126x SHARC Processor Block Diagram

Further, the ADSP-2126x processor addresses the five central requirements for DSPs:

- Fast, flexible arithmetic computation units
- Unconstrained data flow to and from the computation units
- Extended precision and dynamic range in the computation units
- Dual address generators with circular buffering support
- Efficient program sequencing

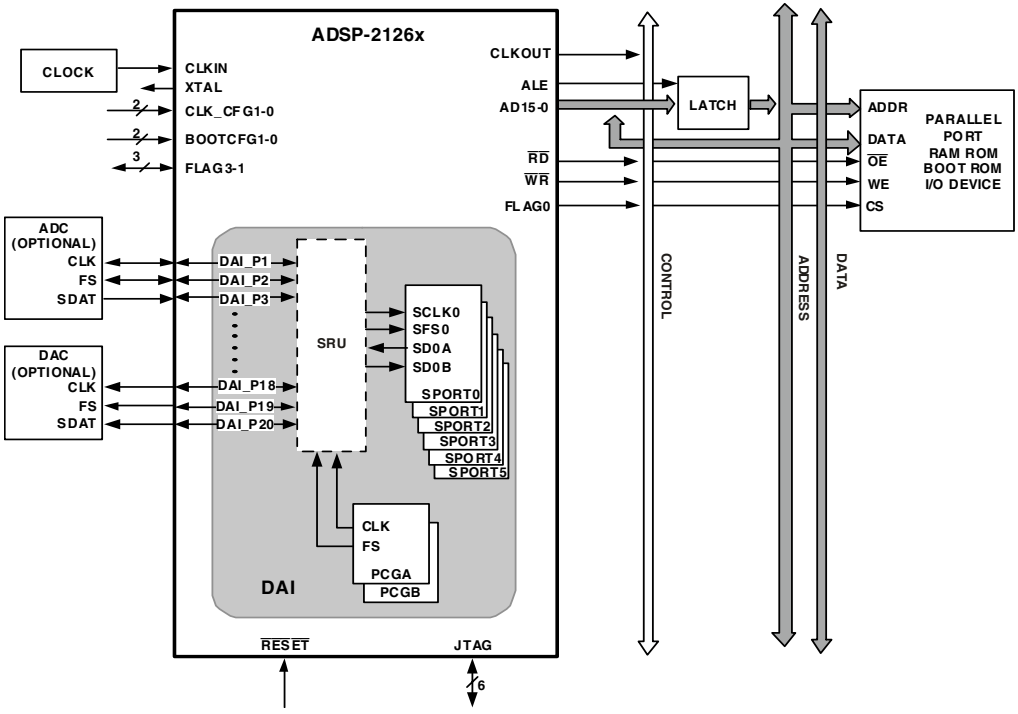


Figure 1-2. Typical Single Processor System

Fast, Flexible Arithmetic. The ADSP-21000 family processors execute all instructions in a single cycle. They provide fast cycle times and a complete set of arithmetic operations. The processor is IEEE floating-point compatible and allows either interrupt on arithmetic exception or latched status exception handling.

Unconstrained Data Flow. The ADSP-2126x processor has a Super Harvard Architecture combined with a ten-port data register file. In every cycle, the processor can write or read two operands to or from the register file, supply two operands to the ALU, supply two operands to the

multiplier, and receive three results from the ALU and multiplier. The processor's 48-bit orthogonal instruction word supports parallel data transfers and arithmetic operations in the same instruction.

40-Bit Extended-Precision. The processor handles 32-bit IEEE floating-point format, 32-bit integer and fractional formats (twos-complement and unsigned), and extended-precision 40-bit floating-point format. The processors carry extended precision throughout their computation units, limiting intermediate data truncation errors (up to 80 bits of precision are maintained during multiply-accumulate operations).

Dual Address Generators. The processor has two Data Address Generators (DAGs) that provide immediate or indirect (pre- and post-modify) addressing. Modulus, bit-reverse, and broadcast operations are supported with no constraints on data buffer placement.

Efficient Program Sequencing. In addition to zero-overhead loops, the processor supports single-cycle setup and exit for loops. Loops are both nestable (six levels in hardware) and interruptable. The processors support both delayed and non-delayed branches.

High Bandwidth I/O. The processors contain up to a dedicated, 4M bits on-chip ROM, a parallel port, an SPI port, serial ports, Digital Audio Interface (DAI), and JTAG. The DAI incorporates a precision clock generator, input data port, and a signal routing unit.

Serial Ports. Provides an inexpensive interface to a wide variety of digital and mixed-signal peripheral devices. The serial ports can operate at up to half the processor core clock (`CCLK`) rate.

Digital Audio Interface (DAI). The DAI includes a precision clock generator, an input data port and a signal routing unit.

Input Data Port (IDP). The IDP provides an additional input path to the processor core configurable as eight channels of serial data or seven channels of serial data and a single channel of up to 20-bit wide parallel data.

Signal Routing Unit (SRU). Provides configuration flexibility by allowing software-programmable connections to be made between the DAI components, serial ports, three pulse-width modulation (PWM) timers, and 20 DAI pins.

Serial Peripheral Interface (SPI). The SPI provides master or slave serial boot through SPI, full-duplex operation, master-slave mode multi-master support, open drain outputs, Programmable baud rates, clock polarities, and phases.

I/O Processor (IOP). The IOP manages the SHARC processor's off-chip data I/O to alleviate the core of this burden. This unit manages the other processor peripherals such as the SPI, DAI, and IDP as well as direct memory accesses (DMA).

Architectural Overview

The ADSP-2126x processor forms 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 ADSP-2126x processor architecture, which appears in Figure 1-1.

Processor Core

The processor core of the ADSP-2126x processor 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. All digital signal processing occurs in the processor core. For complete information, see the *ADSP-2126x SHARC Processor Core Manual*.

Processor Peripherals

The term processor peripherals refers to the multiple on-chip functional blocks used to communicate with off-chip devices. The ADSP-2126x processor peripherals include the JTAG, Parallel, Serial, SPI ports, DAI components (PCG, Timers, and IDP), and any external devices that connect to the processor.

Dual-Ported Internal Memory (SRAM)

The individual ADSP-2126x processor products contain varying amounts of memory. For example, the ADSP-21262 processor provides 2M bits of internal SRAM and 2M bits of internal ROM, each of which is organized as two blocks of 1M bit. Each memory block of SRAM is dual-ported for single cycle, independent accesses by the core processor and I/O processor. The dual-ported memory and separate on-chip buses allow two data transfers from the core and one from I/O, all in a single cycle.

All of the memory can be accessed as 16-, 32-, 48-, or 64-bit words. The amount of memory for each word size changes, based on the part number. On the ADSP-2126x processor, the memory can be configured as a maximum of 64K words of 32-bit data, 128K words of 16-bit data, 42K words of 48-bit instructions (and 40-bit data), or combinations of different word sizes up to 2M bits.

The processor also supports a 16-bit floating-point storage format, which effectively doubles the amount of data that may be stored on chip. Conversion between the 32-bit floating-point and 16-bit floating-point formats completes in a single instruction.

While each memory block can store combinations of code and data, accesses are most efficient when one block stores data, (using the DM bus for transfers), and the other block stores instructions and data, (using the PM bus for transfers). Using the DM bus and PM bus in this way, with one dedicated to each memory block, assures single-cycle execution with two data transfers. In this case, the instruction must be available in the

cache. The processor also maintains single-cycle execution when one of the data operands is transferred to or from off-chip, using the processor parallel port.

I/O Processor

The ADSP-2126x processor Input/Output Processor (IOP) manages the SHARC processor's off-chip data I/O to alleviate the core of this burden. Up to 22 simultaneous DMA transfers (22 DMA channels) are supported for transfers between internal memory and serial ports (12), the input data port (IDP) (8), SPI port (1), and the parallel port. The I/O processor can perform DMA transfers between the peripherals and internal memory at the full core clock speed. The dual-ported architecture of the internal memory allows the IOP and the core to access internal memory simultaneously with no reduction in throughput.

Serial Ports. The ADSP-2126x processor features up to six synchronous serial ports that provide an inexpensive interface to a wide variety of digital and mixed-signal peripheral devices. The serial ports can operate at up to up to half of the processor core clock rate with maximum of 50M bits per second. Each serial port features two data pins that function as a pair based on the same serial clock and frame sync. Accordingly, each serial port has two DMA channels and serial data buffers associated with it to service the dual serial data pins. Programmable data direction provides greater flexibility for serial communications. Serial port data can automatically transfer to and from on-chip memory using DMA. Each of the serial ports offers a TDM multichannel mode (up to 128 channels) and supports μ-law or A-law companding. I^2S support is also provided.

The serial ports can operate with least significant bit first (LSBF) or most significant bit first (MSBF) transmission order, with word lengths from three to 32 bits. The serial ports offer selectable synchronization and transmit modes. Serial port clocks and frame syncs can be internally or externally generated.

Parallel Port. The ADSP-2126x processor parallel port provides the processor interface to asynchronous 8-bit memory. The parallel port supports a 66M bytes per second transfer rate and 256 word page boundaries. The on-chip DMA controller automatically packs external data into the appropriate word width during transfers.

The parallel port supports packing of 32-bit words into 8-bit or 16-bit external memory and programmable external data access duration from 3 to 32 clock cycles.

Serial Peripheral (Compatible) Interface (SPI). The ADSP-2126x processor SPI is an industry standard synchronous serial link that enables the SPI-compatible port to communicate with other SPI-compatible devices. SPI is an interface consisting of two data pins, one device select pin, and one clock pin. It is a full-duplex synchronous serial interface, supporting both master and slave modes. It can operate in a multi master environment by interfacing with up to four other SPI-compatible devices, either acting as a master or slave device.

The SPI-compatible peripheral implementation also supports programmable baud rate and clock phase/polarities, as well as the use of open drain drivers to support the multi master scenario to avoid data contention.

ROM Based Security. For ADSP-2126x processors with application code in the on-chip ROM, an optional ROM security feature is included. This feature provides hardware support for securing user software code by preventing unauthorized reading from the enabled code. The processor does not boot-load any external code, executing exclusively from internal ROM. The processor also is not freely accessible via the JTAG port. Instead, a 64-bit key is assigned to the user. This key must be scanned in through the JTAG or Test Access Port. The device ignores a wrong key. Emulation features and external boot modes are only available after the correct key is scanned.

Digital Audio Interface (DAI)

The Digital Audio Interface (DAI) unit is a new addition to the SHARC processor peripherals. This set of audio peripherals consists of an interrupt controller, an interface data port, and a signal routing unit.

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

Input Data Port (IDP). The input data port provides the DAI with a way to transmit data from within the DAI to the core. The IDP provides a means for up to eight additional DMA paths from the DAI into on-chip memory. All eight channels support 24-bit wide data and share a 16-deep FIFO.

Signal Routing Unit (SRU). Conceptually similar to a "patch-bay" or multiplexer, the SRU provides a group of registers that define the interconnection of the serial ports, the interface data port, the DAI pins, and the precision clock generators.

Development Tools

The ADSP-2126x processor is supported by VisualDSP++, an easy to use Integrated Development & Debugging Environment (IDDE). VisualDSP++ allows you to manage projects from start to finish from within a single, integrated interface. Because the project development and debug environments are integrated, you can move easily between editing, building, and debugging activities.

Differences From Previous SHARCs

This section identifies differences between the ADSP-2126x processor and previous SHARCs: ADSP-21161, ADSP-21160, ADSP-21060, ADSP-21061, ADSP-21062, and ADSP-21065L. Like the ADSP-2116x family, the ADSP-2126x processor family is based on the original ADSP-2106x SHARC family. The ADSP-2126x processor preserves much of the ADSP-2106x architecture and is code compatible to the ADSP-21160, while extending performance and functionality. For background information on SHARC processors and the ADSP-2106x Family processors, see the *ADSP-2106x SHARC User's Manual* or the *ADSP-21065L SHARC DSP Technical Reference*.

Processor Core Enhancements

Computational bandwidth on the ADSP-2126x processor is significantly greater than that on the ADSP-2106x processors. The increase comes from raising the operational frequency and adding another processing element: ALU, shifter, multiplier, and register file. The new processing element lets the processor process multiple data streams in parallel (SIMD mode). The processor operates at 200 MHz using a three stage pipeline.

Like the ADSP-21160 processor, the program sequencer on the ADSP-2126x processor differs from the ADSP-2106x processor family, having several enhancements: new interrupt vector table definitions, SIMD mode stack and conditional execution model, and instruction decodes associated with new instructions. Interrupt vectors have been added that detect illegal memory accesses. Also, mode stack and mode mask support have been added to improve context switch time.

As with the ADSP-21160 processor, the DAGs on the ADSP-2126x processor differ from the ADSP-2106x processors in that DAG2 (for the PM bus) has the same addressing capability as DAG1 (for the DM bus). The DAG registers move 64 bits per cycle. Additionally, the DAGs support the new memory map and long word transfer capability. Circular buffering on

the ADSP-2126x processor can be quickly disabled on interrupts and restored on the return. Data “broadcast”, from one memory location to both data register files, is determined by appropriate index register usage.

Processor Internal Bus Enhancements

The PM, DM, and I/O data buses have increased from 32 bits on the ADSP-2106x DSPs to 64 bits. Additional multiplexing and control logic enable 16-, 32-, or 64-bit wide moves between both register files and memory. The processor is capable of broadcasting a single memory location to each of the register files in parallel. Also, the processor permits register contents to be exchanged between the two processing elements' register files in a single cycle.

Memory Organization Enhancements

The ADSP-2126x processor memory map differs from that of the ADSP-2106x DSPs. The system memory map supports double-word transfers each cycle, reflects extended internal memory capacity for derivative designs, and works with an updated control register for SIMD support. The ADSP-2126x processor family provides enough on-chip memory for several audio decoders.

Parallel Port Enhancements

The parallel port differs from that of the ADSP-2106x DSPs. A new packing mode permits DMA for instructions and data to and from 8-bit external memory. The parallel port supports SRAM, EPROM, and flash memory. There are two modes supported for transfers. In one mode, 8-bit data and 8-bit address can be transferred. In another mode, data and address lines are multiplexed to transfer 16 bits of address/data.

I/O Architecture Enhancements

The I/O processor on the provides much greater throughput than that on the ADSP-2106x DSPs.

The ADSP-2126x processor DMA controller supports up to 22 channels compared to 14 channels on the ADSP-21161 processor. DMA transfers occur at clock speed in parallel with full speed processor execution.

Instruction Set Enhancements

The ADSP-2126x processor provides source code compatibility with the previous SHARC processor family members, to the application assembly source code level. All instructions, control registers, and system resources available in the ADSP-2106x core programming model are also available in the ADSP-2126x processor. Instructions, control registers, or other facilities, required to support the new feature set of the ADSP-2116x core include:

- Code compatibility to the ADSP-21160 SIMD core
- Supersets of the ADSP-2106x programming model
- Reserved facilities in the ADSP-2106x programming model
- Symbol name changes from the ADSP-2106x programming models

These name changes can be managed through reassembly by using the development tools to apply the ADSP-2126x processor symbol definitions header file and linker description file. While these changes have no direct impact on existing core applications, system and I/O processor initialization code and control code do require modifications.

Although the porting of source code written for the ADSP-2106x family to the ADSP-2126x processor has been simplified, code changes will be required to take full advantage of the new ADSP-2126x processor features. For more information, see the *ADSP-21160 SHARC DSP Instruction Set Reference*.

2 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-2126x processor contains 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 listed below and shown in Figure 2-3 on page 2-22:

- Internal memory ↔ external memory devices
- Internal memory ↔ serial port I/O
- Internal memory ↔ SPI I/O
- Internal memory ← Digital Audio Interface (DAI)

By managing DMA, the I/O processor frees the processor core, allowing it to perform other processor operations while off-chip data I/O occurs as a background task. The dual-ported internal memory allows the core and IOP to simultaneously access the same block of internal memory. 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 DMAs of processor memory through the parallel, SPI, input data port (IDP) and serial ports.

Each DMA is referred to as a *channel*, and each channel is configured independently.

There are 22 channels of DMA available on the ADSP-2126x processor—one channel for the SPI interface, one channel for the parallel port interface, 12 channels via the serial ports, and eight channels for the input data port (IDP). Another DMA feature is interrupt generation upon completion of a DMA transfer or upon completion of a chain of DMAs.

General Procedure for Configuring DMA

To configure the ADSP-2126x processor to use DMA, use the following general procedure.

1. Determine which DMA options you want to use:
    - IOP/Core interaction method – Interrupt driven or status driven (polling)
    - DMA transfer method – Chained or Non chained
    - Channel priority scheme – fixed or rotating
2. Determine how you want the DMA to operate:
    - Determine and set up the data's source and/or destination addresses (INDEX)
    - Set up the word COUNT (data buffer size)
    - Configure the MODIFY values (step size)
3. Configure the peripheral(s):
    - Serial ports (SPORTs)
    - Parallel port (PP)
    - Input data port (IDP)

4. Enable DMA

   - Set the applicable bits in the appropriate registers:
     –parallel port–PPDEN in PPCTL
     –serial port–SDEN_x (SCHEN_x for chaining) in SPCTLx
     –SPI–SPIDEN (SPICHEN for chaining) in SPIDMAC
     –IDP–IDP_DMA_EN in the IDP_CTL

IOP/Core Interaction Options

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.

Interrupt Driven I/O

Interrupts on the ADSP-2126x processor are generated at the end of a DMA transfer. This happens when the count register for a particular channel decrements to zero. The interrupt vector locations for each of the channels are listed in Table 2-1. The interrupt register diagram and bit descriptions are given in the *ADSP-2126x SHARC Processor Core Manual* and "DAI Interrupt Controller Registers" on page A-110.

Programs can check the appropriate status register (for example PPCTL for the parallel port) 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. The only exception to this is the IDP_DMAx_STAT bits of the DAI_STAT register can become active even if DMA, through some IDP channel, is not intended.

The following are some other I/O processor interrupt attributes.

- When an unchained (single block) DMA process reaches completion (as the count decrements to zero) on any DMA channel, the I/O processor latches that DMA channel's interrupt. It does this by setting the DMA channel's interrupt latch bit in the IRPTL, LIRPTL, DAI_IRPTL_H, or DAI_IRPTL_L registers.
- For chained DMA, the I/O processor generates interrupts in one of two ways: If PCI = 1, an interrupt occurs for each DMA in the chain; if PCI = 0, an interrupt occurs at the end of a complete chain. (For more information on DMA chaining, see "DMA Controller Operation" on page 2-8).
- When a DMA channel's buffer is not being used for a DMA process, the I/O processor can generate an interrupt on single word writes or reads of the buffer. This interrupt service differs slightly for each port. For more information on single word interrupt-driven transfers, see "Parallel Port Control Register (PPCTL)" on page A-55, and SPCTL register in Table 4-6 on page 4-51.

During interrupt-driven DMA, programs use the interrupt mask bits in the IMASK, LIRPTL, DAI_IRPTL_PRI, DAI_IRPTL_RE, and DAI_IRPTL_FE registers to selectively mask DMA channel interrupts that the I/O processor latches into the IRPTL, LIRPTL, DAI_IRPTL_H, and DAI_IRPTL_L registers.

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 more word to be transferred or received and an interrupt is then generated.

A channel interrupt mask in the IMASK, LIRPTL, DAI_IRPTL_PRI, DAI_IRPTL_RE, and DAI_IRPTL_FE registers determines whether a latched interrupt is to be serviced or not. When an interrupt is masked, it is latched but not serviced.

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.

The I/O processor can also generate interrupts for I/O port operations that do not use DMA. In this case, the I/O processor generates an interrupt when data becomes available at the receive buffer or when the transmit buffer is not full (when there is room for the core to write to the buffer). Generating interrupts in this manner lets programs implement interrupt-driven I/O under control of the processor core. Care is needed because multiple interrupts can occur if several I/O ports transmit or receive data in the same cycle.

Table 2-1. DMA Interrupt Vector Locations

Associated Register(s)	Bits	Vector Address	Interrupt Name	DMA Channel	Data Buffer
IRPTL/IMASK	14	0x38	SP1I	0	RXSP1A, TXSP1A
LIRPTL	0	0x44	SP0I	2	RXSP0A, TXSP0A
IRPTL/IMASK	15	0x3C	SP3I	4	RXSP3A, TXSP3A
LIRPTL	1	0x48	SP2I	6	RXSP2A, TXSP2A
IRPTL/IMASK	16	0x40	SP5I	8	RXSP5A, TXSP5A
LIRPTL	2	0x4C	SP4I	10	RXSP4A, TXSP4A
IRPTL/IMASK	14	0x38	SP1I	1	RXSP1B, TXSP1B
LIRPTL	0	0x44	SP0I	3	RXSP0B, TXSP0B
IRPTL/IMASK	15	0x3C	SP3I	5	RXSP3B, TXSP3B
LIRPTL	1	0x48	SP2I	7	RXSP2B, TXSP2B

Table 2-1. DMA Interrupt Vector Locations (Cont'd)

Associated Register(s)	Bits	Vector Address	Interrupt Name	DMA Channel	Data Buffer
IRPTL/IMASK	16	0x40	SP5I	9	RXSP5B, TXSP5B
LIRPTL	2	0x4C	SP4I	11	RXSP4B, TXSP4B
IRPTL/IMASK (high priority option) LIRPTL (low priority option)	12 9	0x30 0x74	SPIHI SPILI	20	RXSPI, TXSPI
LIRPTL	3	0x50	PPI	21	RXPP, TXPP
IRPTL/IMASK (high priority option) LIRPTL (low priority option)	11 6	0x2C 0x5C	DAIHI DAILI	12	IDP_FIF0
IRPTL/IMASK (high priority option) LIRPTL (low priority option)	11 6	0x2C 0x5C	DAIHI DAILI	13	IDP_FIF0
IRPTL/IMASK (high priority option) LIRPTL (low priority option)	11 6	0x2C 0x5C	DAIHI DAILI	14	IDP_FIF0
IRPTL/IMASK (high priority option) LIRPTL (low priority option)	11 6	0x2C 0x5C	DAIHI DAILI	15	IDP_FIF0
IRPTL/IMASK (high priority option) LIRPTL (low priority option)	11 6	0x2C 0x5C	DAIHI DAILI	16	IDP_FIF0
IRPTL/IMASK (high priority option) LIRPTL (low priority option)	11 6	0x2C 0x5C	DAIHI DAILI	17	IDP_FIF0

Table 2-1. DMA Interrupt Vector Locations (Cont'd)

Associated Register(s)	Bits	Vector Address	Interrupt Name	DMA Channel	Data Buffer
IRPTL/IMASK (high priority option) LIRPTL (low priority option)	11 6	0x2C 0x5C	DAIHI DAILI	18	IDP_FIF0
IRPTL/IMASK (high priority option) LIRPTL (low priority option)	11 6	0x2C 0x5C	DAIHI DAILI	19	IDP_FIF0

The SPI has two interrupts—a lower priority option (SPILI) and a higher priority option (SPIHI). This allows two interrupts to have priorities that are higher and lower than serial ports.

The DAI also has two interrupts—the lower priority option (DAILI) and higher priority option (DAIHI). This allows two interrupts to have priorities that are higher and lower than serial ports.

Polling/Status Driven I/O

The second method of controlling I/O is through status polling. The I/O processor monitors the status of data transfers on DMA channels and indicates interrupt status in the IRPTL, LIRPTL, DAI_IRPTL_H, and DAI_IRPTL_L registers. Note that because polling uses processor resources it is not as efficient as an interrupt-driven system. Also note that polling the DMA status registers reduces I/O bandwidth. The following provide more information on the registers that control and monitor I/O processes.

- All the bits in IRPTL and LIRPTL registers are shown in the *ADSP-2126x SHARC Processor Core Manual*.
- Figure A-59 on page A-112 lists all the bits in DAI_IRPTL_H.
- Figure A-60 on page A-113 lists all the bits in DAI_IRPTL_L.

The DMA controller in the ADSP-2126x processor maintains the status information of the channels in each of the peripherals registers, SPMCTLxy, PPCTL, DAI_STAT, and SPIDMAC. More information on these registers can be found at the following locations.

- Bit definitions for the SPIDMAC register are illustrated in "SPI DMA Configuration Register (SPIDMAC)" on page A-50.
- Bit definitions for the SPMCTLxy register are illustrated in "SPORT Multichannel Control Registers (SPMCTLxy)" on page A-28.
- Bit definitions for the PPCTL register are illustrated in "Parallel Port Control Register (PPCTL)" on page A-55.
- Bit definitions for the DAI_STAT register are illustrated in Figure A-56 on page A-105.

There is a one cycle latency between a change in DMA channel status and the status update in the corresponding register.

If chaining is enabled on a DMA channel, programs should not use polling to determine channel status as it can provide inaccurate information. In this case, the DMA appears inactive if it is sampled while the next transfer control block (TCB) is loading.

DMA Controller Operation

There are two methods you can use to start DMA sequences: chaining and non-chaining.

Non-chained DMA. To start a new DMA sequence after the current one is finished, a program must first clear the DMA enable bit, write new parameters to the index, modify, and count registers, then set the DMA enable bit to re-enable 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 pointed to by that channel's chain pointer (`CP`) register. Using chaining, programs can set up consecutive DMA operations and each operation can have different attributes.

Chaining is only supported on the SPI and SPORT DMA channels. The parallel port, and IDP port do not support chaining.

In general, a DMA sequence starts when one of the following occurs:

- Chaining is disabled, and the DMA enable bit transitions from low to high.
- Chaining is enabled, DMA is enabled, and the chain pointer register address field is written with a nonzero value. In this case, TCB chain loading of the channel parameter registers occurs first.
- Chaining is enabled, the chain pointer register address field is nonzero, and the current DMA sequence finishes. Again, TCB chain loading occurs.

A DMA sequence ends when one of the following occurs:

- The count register decrements to zero, and the `CP` register is zero.
- Chaining is disabled and the channel's DMA enable bit transitions from high to low. If the DMA enable bit goes low (=0) and chaining is enabled, the channel enters chain insertion mode and the DMA sequence continues. For more information, see "Inserting a TCB in an Active Chain" on page 2-15.

Once a program starts a DMA process, the process is influenced by two external controls—DMA channel priority and DMA chaining. For more information, see "Managing DMA Channel Priority" on page 2-17 or "Chaining DMA Processes" below.

Chaining DMA Processes

The location of the DMA parameters for the next sequence comes from the chain pointer (CP) register. In chained DMA operations, the ADSP-2126x processor automatically initializes and then begins another DMA transfer when the current DMA transfer is complete. In addition to the standard DMA parameter registers, each DMA channel (SP and SPI) also has a CP register that points to the next set of DMA parameters stored in the processor's internal memory. In the SPI this is the CPSPI and in the SPORT it is CPSPxy. Each new set of parameters is stored in a four-word, user initialized buffer in internal memory known as a transfer control block (TCB). In TCB chain loading, the ADSP-2126x processor's IOP automatically reads the TCB from internal memory and then loads the values into the channel parameter registers to set up the next DMA sequence.

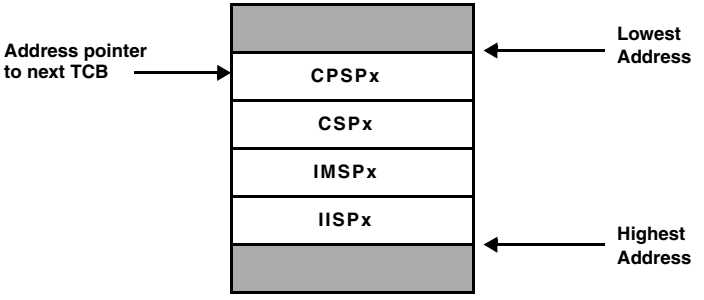
The structure of a TCB is conceptually the same as that of a traditional linked-list. Each TCB has several data values and a pointer to the next TCB. Further, the chain pointer of a TCB may point to itself to constantly reiterate the same DMA.

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. This bit must be set to one to enable chaining. When chaining is enabled, DMA transfers are initiated by writing a memory address to the CP register. This is also an easy way to start a single DMA sequence, with no subsequent chained DMAs.

The CP register can be loaded at any time during the DMA sequence. This allows a DMA channel to have chaining disabled (CP register address field = 0x0000) until some event occurs that loads the CP register with a nonzero value. Writing all zeros to the address field of the chain pointer register (CP) also disables chaining.

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

The parallel port and IDP port do not support DMA chaining.



Chaining is not available on the IDP or parallel ports.
An "x" denotes the DMA channel used.

Figure 2-1. TCB Chaining

The chain pointer register is 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 ADSP-2126x processor, this offset value is 0x0008 0000.

Bit 19 of the chain pointer register is the Program Controlled Interrupts (PCI) bit. This bit controls whether an interrupt is latched after each DMA completes or whether the interrupt is latched after the entire DMA sequence completes. If set, the PCI bit enables a DMA channel interrupt to occur after every DMA in the chain. If cleared, an interrupt occurs at the completion of the entire DMA sequence.

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.

Because the PCI bit is not part of the memory address in the chain pointer register, programs must use care when writing and reading addresses to and from the register. To prevent errors, programs should mask out the PCI bit (bit 19) when copying the address in a chain pointer to another address register.

The DMA registers are shown in Figure 2-2.

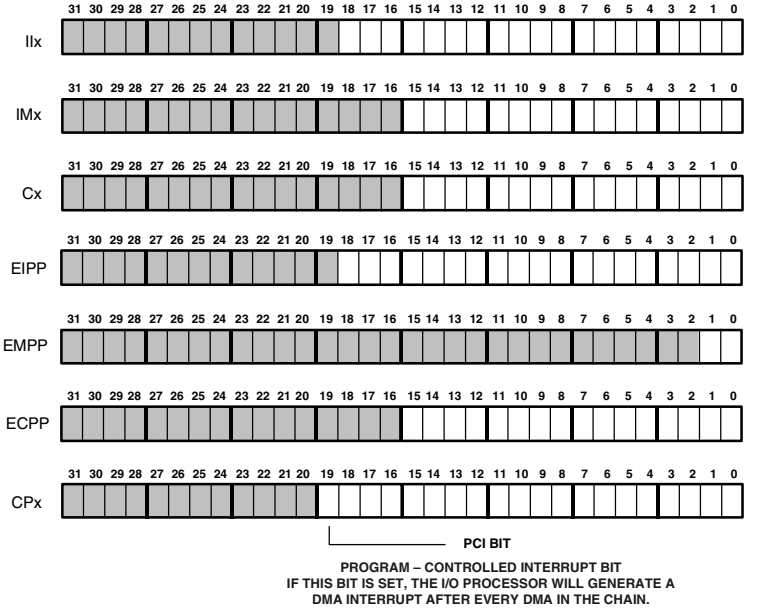


Figure 2-2. DMA Parameter Registers

Transfer Control Block Chain Loading (TCB)

During TCB chain loading, the I/O processor loads the DMA channel parameter registers with values retrieved from internal memory. 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 CP register should not point to the first location of the array. Instead the CP register should point to array[3].

Table 2-2 shows the TCB-to-register loading sequence for the serial port and SPI port DMA channels. The I/O processor reads each word of the TCB and loads it into the corresponding register. Programs must set up the TCB in memory in the order shown in Table 2-2, placing the index parameter at the address pointed to by the CP 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 CP value of zero.

Table 2-2. TCB Chain Loading Sequence[1]

Address[2]	Serial Ports	SPI Port
CPSPx + 0x0008 0000	IISPx	IISPI
CPSPx – 1 + 0x0008 0000	IMSPx	IMSPI
CPSPx – 2 + 0x0008 0000	CSPx	CSPI
CPSPx – 3 + 0x0008 0000	CPSPx	CPSPI

1 Chaining is not available using the IDP or parallel ports.
2 An "x" denotes the DMA channel used. While the TCB is eight locations in length, SPI and serial ports only use the first four locations.

A TCB chain load request is prioritized like all other DMA operations. 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 registers for the highest priority DMA channel first. A channel that is in the process of chain loading cannot be interrupted by a higher priority channel. For a list of DMA channels in priority order, see Table 2-5 on page 2-28.

Setting Up and Starting the Chain

To set up and initiate a chain of DMA operations, use these steps:

1. Set up all TCBs in internal memory.
2. Write to the appropriate DMA control register, setting the DMA enable bit to one and the chaining enable bit to one.
3. Write the address containing the index register value of the first TCB to the chain pointer register, which starts the chain.

The I/O processor responds by autoinitializing the first DMA parameter registers with the values from the first TCB, and then starts the first data transfer.

Setting Up and Starting Chained DMA over the SPI

Configuring and starting chained DMA transfers over the SPI port is the same as for the serial port, 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`), and the chain pointer register (`CPSPI`) points to a TCB that describes the second DMA in the sequence.

Writing an address to the `CPSPI` register does not begin a chained DMA sequence unless `IISPI`, `IMSPI`, and `CSPI` are initialized, SPI DMA is enabled, the SPI port is enabled, and SPI DMA chaining is enabled.

The sequence for setting up and starting a chained DMA is outlined in the following steps and can also be seen in Listing 5-3 on page 5-60.

1. Configure the TCB associated with each DMA in the chain except for the first DMA in the chain.

2. Write the first three parameters for the initial DMA to the IISPI, IMSPI, and CSPI registers directly.
3. Select a baud rate using the SPIBAUD register.
4. Select which flag to use as the SPI slave select signal in the SPIFLG register.
5. Configure and enable the SPI port with the SPICTL register.
6. Configure the DMA settings for the entire sequence, enabling DMA and DMA chaining in the SPIDMAC register.
7. Begin the DMA by writing the address of a TCB (describing the second DMA in the chain) to the CPSPI register.

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. For example:

```
R0 = next_TCB+3; /* addr of next chain */
R1 = 0x7FFFF;    /* mask 19 bits */
R0 = R0 or R1;
CPx = R0;
```

Inserting a TCB in an Active Chain

This is supported by serial ports only. The SPI interface does not support inserting a TCB in an active chain.

It is possible to insert a single DMA operation or another DMA chain within an active 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 lets a program insert a new DMA or DMA chain within the current chain without effecting the current DMA transfer. Use the following sequence to insert a DMA subchain for the serial port 0A channel while another chain is active:

1. Enter chain insertion mode by setting SCHEN_A = 1 and SDEN_A = 0 in the channel's DMA control register, SPCTL0. The DMA interrupt indicates when the current DMA sequence has completed.
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 SCHEN_A = 1 and SDEN_A = 1.

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.

Setting Up DMA Channel Allocation and Priorities

The ADSP-2126x processor has 22 DMA channels including 12 channels accessible via the serial ports, one SPI channel, one parallel port channel, and eight input data port channels. Each channel has a set of parameter registers which are used to set up DMA transfers. Table 2-3 shows the

DMA channel allocation and parameter register assignments for the ADSP-2126x processor. DMA channel 0 has the highest priority and DMA channel 21 has the lowest priority.

Managing DMA Channel Priority

The default channel priority is: DMA channel 0 as highest priority and DMA channel 22 as lowest priority. Table 2-5 on page 2-28 lists the DMA channels in priority order. When a channel becomes the highest priority requester, the I/O processor services the channel's request. In the next clock cycle, the I/O processor starts the DMA transfer.

The I/O data (IOD) bus is 32 bits wide and is the only 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 parallel port may fill its RXPP buffer just as a SPORT shifts a word into its RXn 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.

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. Figure 2-4 on page 2-27 shows the paths for internal DMA requests within the I/O processor.

If a DMA channel is disabled (PPDEN, SPIDEN, SDEN, or IDP_DMA_EN bits =0), the I/O processor does not issue internal DMA grants to that channel (whether or not the channel has data to transfer).

The default DMA channel priority is *fixed prioritization* by DMA channel group (serial ports, parallel port, IDP, or SPI port). Table 2-5 on page 2-28 lists the DMA channels in descending order of priority.

For information on programming serial port priority modes, see Table 4-7 on page 4-65.

The I/O processor determines which DMA channel has the highest priority internal DMA request during every cycle between each data transfer.

Processor core accesses of I/O processor registers and TCB chain loading (both of which occur after the IOD transfer) are subject to the same prioritization scheme as the DMA channels. Applying this scheme uniformly prevents I/O bus contention, because these accesses are also performed over the internal I/O bus. For more information, see "Chaining DMA Processes" on page 2-10.

DMA Bus Arbitration

DMA channel arbitration is the method that the IOP uses to determine how groups rotate priority with other channels. This feature is enabled by setting the `DCPR` bit in the IOP's `SYSCTL` register.

DMA-capable peripherals execute DMA data transfers to and from internal memory over the IOD bus. When more than one of these peripherals requests access to the IOD bus in a clock cycle, the bus arbiter, which is attached to the IOD bus, determines which master should have access to the bus and grants the bus to that master.

IOP channel arbitration can be set to use either a *fixed* (`SYSCTL[7]` = 0) or *rotating* (`SYSCTL[7]` = 1) algorithm.

In the fixed priority scheme, the lower indexed peripheral has the highest priority.

In the rotating priority scheme, the default priorities at reset are the same as that of the fixed priority. However, the peripheral priority is determined by group, not individually. Peripheral groups are shown in Table 2-3.

Initially, Group A has the highest priority and Group F the lowest. As one group completes its DMA operation, it is assigned the lowest priority (moves to the back of the line) and the next group is given the highest priority.

When none of the peripherals request bus access, the highest priority peripheral, for example, peripheral#0, is granted the bus. However, this does not change the currently assigned priorities to various peripherals.

Within a peripheral group the priority is highest for the higher indexed peripheral (see Table 2-3). For example in SP01 (group A), SP1 has the highest priority.

Table 2-3. DMA Channel Allocation and Parameter Register Assignments

DMA Channel Number	Data Buffer	Group	IOP Address of Data Buffers	Description
0 (highest priority)	RXSP1A, TXSP1A	A	0xC65, 0xC64	Serial Port 1A Data
1	RXSP1B, TXSP1B	A	0xC67, 0xC66	Serial Port 1B Data
2	RXSP0A, TXSP0A	A	0xC61, 0xC60	Serial Port 0A Data
3	RXSP0B, TXSP0B	A	0xC63, 0xC62	Serial Port 0 B Data
4	RXSP3A, TXSP3A	B	0x465, 0x464	Serial Port 3A Data
5	RXSP3B, TXSP3B	B	0x467, 0x466	Serial Port 3B Data
6	RXSP2A, TXSP2A	B	0x461, 0x460	Serial Port 2A Data
7	RXSP2B, TXSP2B	B	0x463, 0x462	Serial Port 2B Data

Table 2-3. DMA Channel Allocation and Parameter Register Assignments (Cont'd)

DMA Channel Number	Data Buffer	Group	IOP Address of Data Buffers	Description
8	RXSP5A, TXSP5A	C	0x865 or 0x864	Serial Port 5A Data
9	RXSP5B, TXSP5B	C	0x867 or 0x866	Serial Port 5B Data
10	RXSP4A, TXSP4A	C	0x861 or 0x860	Serial Port 4A Data
11	RXSP4B, TXSP4B	C	0x863 or 0x862	Serial Port 4B Data
12	IDP_FIF0	D	0x24D0	DAI IDP Channel 0
13	IDP_FIF0	D	0x24D0	DAI IDP Channel 1
14	IDP_FIF0	D	0x24D0	DAI IDP Channel 2
15	IDP_FIF0	D	0x24D0	DAI IDP Channel 3
16	IDP_FIF0	D	0x24D0	DAI IPD Channel 4
17	IDP_FIF0	D	0x24D0	DAI IDP Channel 5
18	IDP_FIF0	D	0x24D0	DAI IDP Channel 6
19	IDP_FIF0	D	0x24D0	DAI IDP Channel 7
20	RXSPI, TXSPI	E	0x1004, 0x1003	SPI Data
21 (lowest priority)	RXPP, TXPP	F	0x1809, 0x1808	Parallel Port Data

Setting Up DMA Parameter Registers

Once you have determined and configured the DMA options, you can configure the DMA parameter registers. The parameter registers control the source and destination of the data, the size of the data buffer, and the step size used. These topics are described in detail in the following sections.

DMA Transfer Direction

DMA transfers between internal memory and external memory devices use the processor's parallel port. For these types of transfers, a program provides the DMA controller with the internal memory buffer size, address, and address modifier, as well as the external memory buffer size, address and 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 2-4 on page 2-25 shows the parameter registers for each DMA channel.

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 I/O port 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 2-4 on page 2-27 shows more detail on DMA channel data paths. Figure 2-3 shows the processor's I/O processor, related ports, and buses.

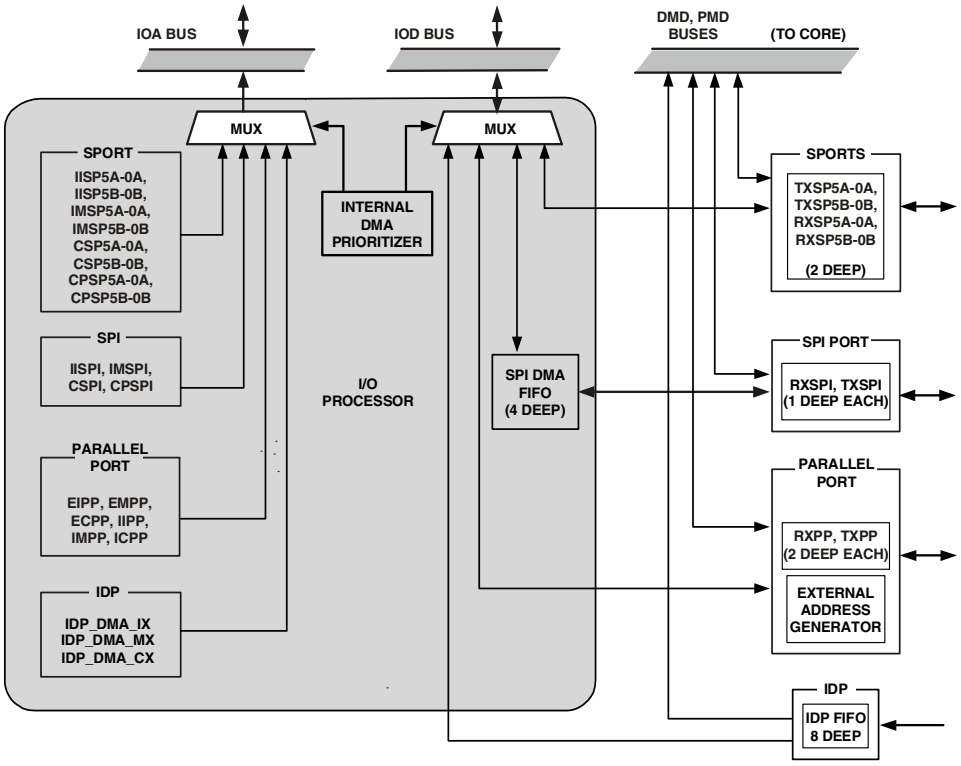


Figure 2-3. I/O Processor Block Diagram

Data Buffer Registers

The data buffer registers in Figure 2-3 on page 2-22 shows the data buffer registers for each port. These registers include:

- **Serial Port Receive Buffer** (`RXSPx`). These receive buffers for the serial ports have two position FIFOs for receiving data when connected to another serial device.
- **Serial Port Transmit Buffer** (`TXSPx`). These transmit buffers for the serial ports have two position FIFOs for transmitting data when connected to another serial device.
- **SPI Receive Buffer** (`RXSPI`). This receive buffer for the SPI port has a single position buffer for receiving data when connected to another serial device.
- **SPI Transmit Buffer** (`TXSPI`). This transmit buffer for the SPI port has a single position buffer for transmitting data when connected to another serial device.
- **Parallel Port Transmit Buffer** (`TXPP`). This transmit buffer for the parallel port has two-position FIFOs for transmitting data when connected to another device.
- **Parallel Port Receive Buffer** (`RXPP`). This receive buffer for the parallel port has two position FIFOs for receiving data when connected to another parallel device.
- **Input Data Port Buffers** (`IDP_FIFO`). This receive buffer for the input data port has eight position buffers for receiving data when connected to another device.

Port, Buffer, and DMA Control Registers

The Port, Buffer, and DMA Control Registers in Figure 2-3 shows the control registers for the ports and DMA channels. These registers include:

- **Parallel Port Control register** (PPCTL). This register enables the parallel port system, DMA, and external data width. It also configures wait states, bus hold cycles and identifies the status of the parallel port FIFO, internal, and external interfaces.
- **Input Data Port Control register** (IDP_CTL). This is the control register for input data ports.
- **Serial Port Control registers** (SPCTLx, SPMCTLxy). These control registers select the receive or transmit format, monitor FIFO status, enable chaining, and start DMA for each serial port.
- **SPI Port Control register** (SPICTL). This control register configures and enables the SPI interface, selects the device as master or slave, and determines the data transfer and word size. The SPIDMAC register also controls SPI DMA and FIFO status.

Table 2-4 shows the parameter registers for each DMA channel. These registers function similarly to data address generator registers and include:

- **Internal Index registers** (IISPx, IISPI, IIPP, IDP_DMA_Ix). Index registers provide an internal memory address, acting as a pointer to the next internal memory DMA read or write location.
- **Internal Modify registers** (IMSPx, IMPP, IMSPI, IDP_DMA_Mx). Modify registers provide the signed increment by which the DMA controller post-modifies the corresponding internal memory index register after the DMA read or write.
- **Count registers** (CSPx, ICPP, CSPI, IDP_DMA_Cx). Count registers indicate the number of words remaining to be transferred to or from internal memory on the corresponding DMA channel.

- **Chain Pointer registers** (CPSPx, CPSPI). Chain pointer registers hold the starting address of the 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.
- **External Index registers** (EIPP). Index registers provide an external memory address, acting as a pointer to the next external memory DMA read or write location.
- **External Modify registers** (EMPP). Modify registers provide the increment by which the DMA controller post-modifies the corresponding external memory index register after the DMA read or write.
- **External Count registers** (ECPP). External count registers indicate the number of words remaining to be transferred to or from external memory on the corresponding DMA channel.

Table 2-4. ADSP-2126x Processor DMA Parameter Registers

Register	Function	Width	Description
IIy	Internal Index Register	19 bits	Address of buffer in internal memory
IMxy	Internal Modify Register	16 bits[1]	Stride for internal buffer
Cxy	Internal Count Register	16 bits	Length of internal buffer
CPxy	Chain Pointer Register	20 bits	Chain pointer for DMA chaining
EIPP	External Index Register	19 bits	Address of buffer in external memory
EMPP	External Modify Register	2 bits	Stride for external buffer
ECPP	External Count Register	16 bits	Length of external buffer

1 IDP_DMA_Mx are 6 bits wide only.

Addressing

Figure 2-4 shows a block diagram of the I/O processor's address generator (DMA controller). Table 2-4 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.

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-2126x processor, this offset value is 0x0008 0000.

DMA addresses must always be normal word (32-bit) memory, and internal memory data transfer sizes are 32 bits. External data transfer sizes may be 16 or 8 bits. The I/O processor can transfer short word data (16-bit) using the packing capability of the serial port and SPI port DMA channels.

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

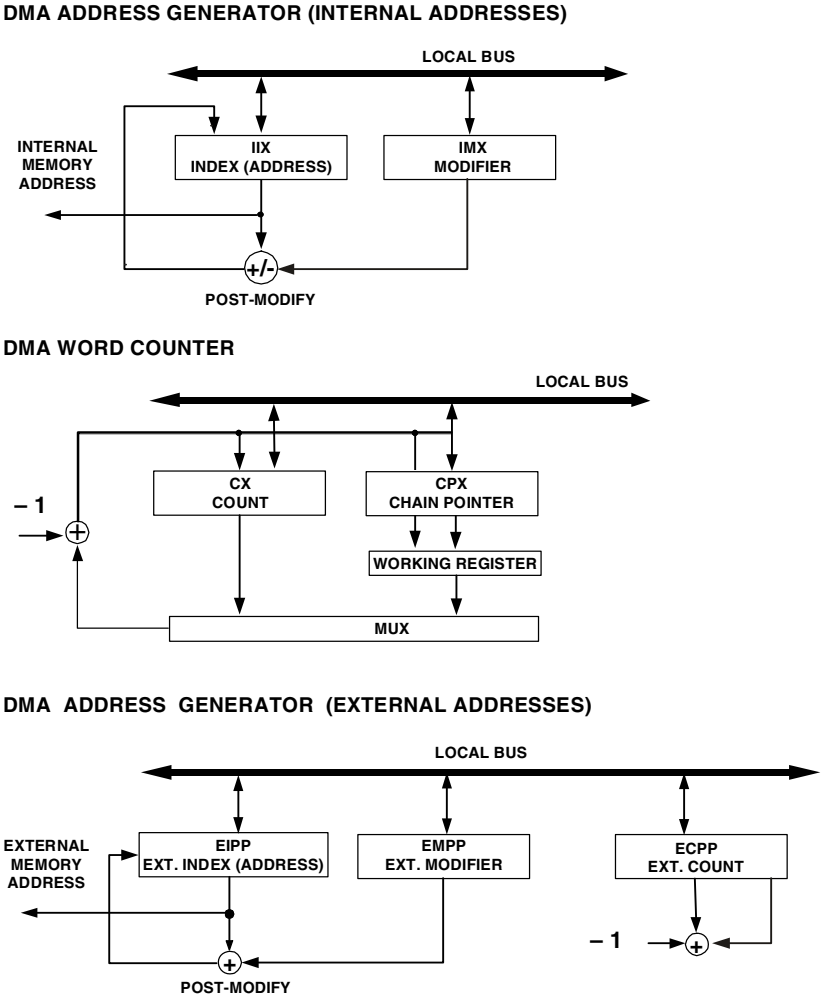


Figure 2-4. DMA Address Generator

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.

If the I/O processor modifies the index register past the maximum 18-bit value to indicate an address out of internal memory, the index wraps around to zero. With the offset for the ADSP-2126x processor, the wraparound address is 0x0008 0000.

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 216 transfers. This count occurs because the I/O processor starts the first transfer before testing the count value. The only way to disable a DMA channel is to clear its DMA enable bit.

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.

The processor's 22 DMA channels are numbered as shown in Table 2-5. This table also shows the control, parameter, and data buffer registers that correspond to each channel.

Note: In SP01, SP1 has a higher priority. Similarly, for SP23 and SP45, the odd numbered SPs have a higher priority (SP3, SP5).

Table 2-5. DMA Channel Registers: Controls, Parameters, and Buffers

DMA Channel Number	Control Registers	Parameter Registers	Buffer Registers	Description
0	SPCTL1	IISP1A, IMSP1A, CSP1A, CPSP1A	RXSP1A, TXSP1A	Serial Port 1A Data
1	SPCTL1	IISP1B, IMSP1B, CSP1B, CPSP1B	RXSP1B, TXSP1B	Serial Port 1B Data

Table 2-5. DMA Channel Registers: Controls, Parameters, and Buffers (Cont'd)

DMA Channel Number	Control Registers	Parameter Registers	Buffer Registers	Description
2	SPCTL0	IISP0A, IMSP0A, CSP0A, CPSP0A	RXSP0A, TXSP0A	Serial Port 0A Data
3	SPCTL0	IISP0B, IMSP0B, CSP0B, CPSP0B	RXSP0B, TXSP0B	Serial Port 0B Data
4	SPCTL3	IISP3A, IMSP3A, CSP3A, CPSP3A	RXSP3A, TXSP3A	Serial Port 3A Data
5	SPCTL3	IISP3B, IMSP3B, CSP3B, CPSP3B	RXSP3B, TXSP3B	Serial Port 3B Data
6	SPCTL2	IISP2A, IMSP2A, CSP2A, CPSP2A	RXSP2A, TXSP2A	Serial Port 2A Data
7	SPCTL2	IISP2B, IMSP2B, CSP2B, CPSP2B	RXSP2B, TXSP2B	Serial Port 2B Data
8	SPCTL5	IISP5A, IMSP5A, CSP5A, CPSP5A	RXSP5A, TXSP5A	Serial Port 5A Data
9	SPCTL5	IISP5B, IMSP5B, CSP5B, CPSP5B	RXSP5B, TXSP5B	Serial Port 5B Data
10	SPCTL4	IISP4A, IMSP4A, CSP4A, CPSP4A	RXSP4A, TXSP4A	Serial Port 4A Data
11	SPCTL4	IISP4B, IMSP4B, CSP4B, CPSP4B	RXSP4B, TXSP4B	Serial Port 4B Data
12	IDP_CTL	IDP_DMA_I0, IDP_DMA_M0, IDP_DMA_C0	IDP_FIFO	DAI IDP Channel 0
13	IDP_CTL	IDP_DMA_I1, IDP_DMA_M1, IDP_DMA_C1	IDP_FIFO	DAI IDP Channel 1
14	IDP_CTL	IDP_DMA_I2, IDP_DMA_M2, IDP_DMA_C2	IDP_FIFO	DAI IDP Channel 2

Table 2-5. DMA Channel Registers: Controls, Parameters, and Buffers (Cont'd)

DMA Channel Number	Control Registers	Parameter Registers	Buffer Registers	Description
15	IDP_CTL	IDP_DMA_I3, IDP_DMA_M3, IDP_DMA_C3	IDP_FIFO	DAI IDP Channel 3
16	IDP_CTL	IDP_DMA_I4, IDP_DMA_M4, IDP_DMA_C4	IDP_FIFO	DAI IPD Channel 4
17	IDP_CTL	IDP_DMA_I5, IDP_DMA_M5, IDP_DMA_C5	IDP_FIFO	DAI IDP Channel 5
18	IDP_CTL	IDP_DMA_I6, IDP_DMA_M6, IDP_DMA_C6	IDP_FIFO	DAI IDP Channel 6
19	IDP_CTL	IDP_DMA_I7, IDP_DMA_M7, IDP_DMA_C7	IDP_FIFO	DAI IDP Channel 7
20	SPICTL	IISPI, IMSPI, CSPI, CPSPI	RXSPI, TXSPI	SPI Data
21	PPCTL	EIPP, EMPP, ECPP, IIPP, IMPP, ICPP	RXPP, TXPP	Parallel Port Data

All of the I/O processor's registers are memory-mapped, ranging from address 0x0000 0000 to 0x0003 FFFF. For more information on these registers, see "I/O Processor Registers" on page A-2.

Setting Up DMA

Because the I/O processor registers are memory-mapped, the processor has access to program DMA operations. A processor 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 SPI port, parallel port, serial ports and input data ports each have a DMA enable bit (SPIDEN, PPDEN, SDEN or IDP_DMA_EN) in their channel control register. 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 internal memory only.

3 PARALLEL PORT

The ADSP-2126x processor has a parallel port that allows bidirectional transfers between it and external parallel devices. Using the parallel port bus and control lines, the processor can interface to 8-bit or 16-bit wide external memory devices. The parallel port provides a DMA interface between internal and external memory and has the ability to support core driven data transfer modes. Regardless of whether 8-bit or 16-bit external memory devices are used, the internal data word size is always 32 bits (normal word addressing), and the parallel port employs packing to place the data appropriately.

The processor provides two data packing modes, 8/32 and 16/32. For reads, data packing involves shifting multiple successive 8- or 16-bit elements from the parallel port to the ADSP-2126x processor's Receive register to form each 32-bit word, transferring multiple successive 8-bit or 16-bit elements. For writes, packing involves shifting each 32-bit word out into 8- or 16-bit elements that are placed into the memory device.

This chapter describes the parallel port operation, registers, interrupt function, and transfer protocol. Figure 3-1 shows a block diagram of the parallel port.

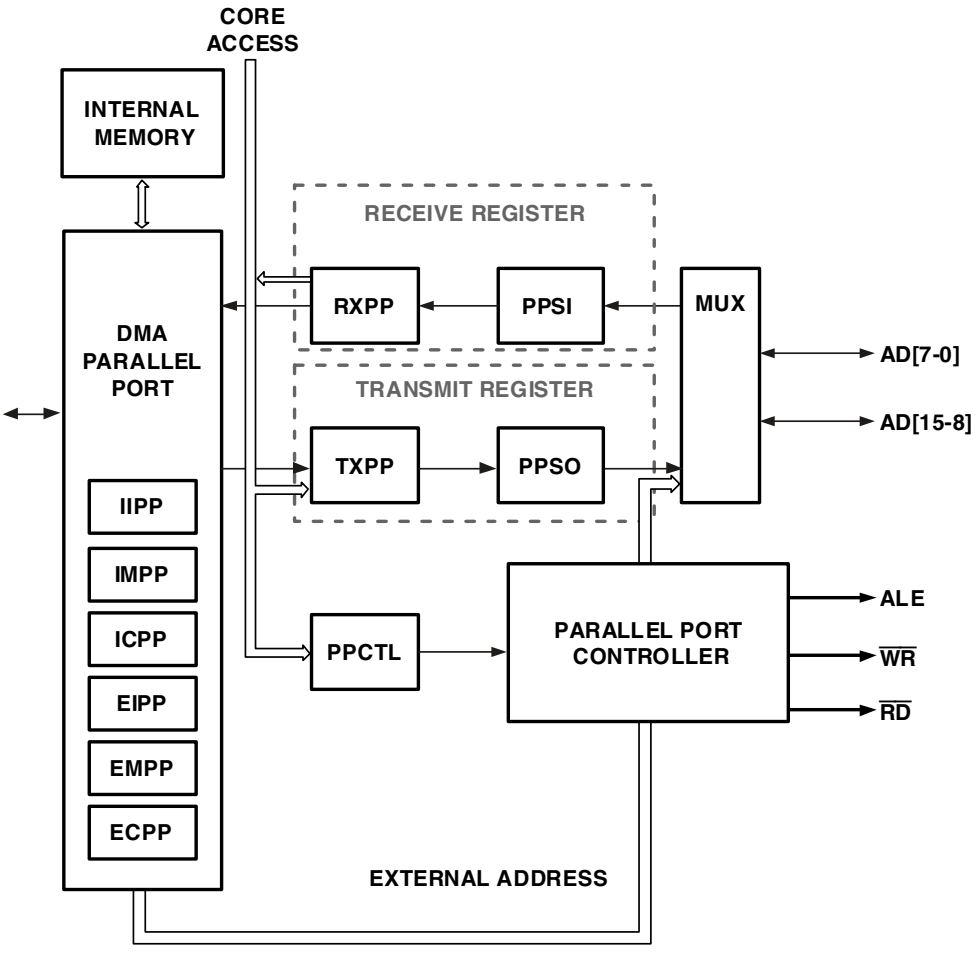


Figure 3-1. Parallel Port Block Diagram

Parallel Port Pins

This section describes the pins that the parallel port uses for its operation.

For a complete list of pin descriptions and package pinouts, see the product specific data sheet for your device.

Address/Data (AD15–0) pins. The ADSP-2126x processor provides time multiplexed address/data pins that are used for providing both address and data information. The state of the address/data pins is determined by the 8- or 16-bit operating mode and the state of the ALE, $\overline{\text{RD}}$, and $\overline{\text{WR}}$ pins.

Read strobe ($\overline{\text{RD}}$) pin. This output pin is asserted low to indicate a read operation. Data is latched into the processor using the rising edge of this signal.

Write strobe ($\overline{\text{WR}}$) pin. This output pin is asserted low to indicate a write operation. The rising edge of this signal can be used by memory devices to latch the data from the processor.

Address Latch Enable (ALE) pin. The address latch enable pin is used to strobe an external latch connected to the address/data pins (AD15-0). The external latch holds the most significant bits (MSBs) of the external memory address. An ALE cycle is inserted for the first access after the parallel port is enabled and anytime the upper 16 bits of the address change from a previous cycle.

In 8-bit mode, a maximum of 24 bits of external address are facilitated through latching the upper 16 bits, EA23-8, from AD15-0 into the external latch during the ALE phase of the cycle. The remaining 8 bits of address EA7-0 are provided through AD15-8 during the second half of the cycle when the $\overline{\text{RD}}$ or $\overline{\text{WR}}$ signal is asserted.

In 16-bit mode, a maximum of 16 bits of external address are facilitated through latching the upper 16 bits of AD15-0 from AD15-0 into the external latch during the ALE phase of the cycle. The AD15-0 represent the external 16 bits of data during the second half of the cycle when the $\overline{\text{RD}}$ or $\overline{\text{WR}}$ signal is asserted.

The ALE pin is active high by default, but can be set active low via the PPALEPL bit (bit 13) in the Parallel Port Control (PPCTL) register.

Since ALE polarity is active high by default, systems using parallel port boot mode must use address latching hardware that can process this active high signal.

Alternate Pin Functions

The following sections describe how to make the parallel port pins function as flag pins and how to make the parallel data acquisition port pins function as address pins. For additional information on pin multiplexing, see "Pin Multiplexing" on page 9-5.

Parallel Ports as FLAG Pins

Setting (= 1) bit 20 in the SYSCTL register, (PPFLGS) causes the 16 address pins to function as FLAG0-FLAG15. To use the parallel port for data access, this bit should be cleared (= 0). For more information, see "System Design" on page 9-1.

The ADSP-2126x processor supports up to 16 general purpose FLAG pins. These FLAG signals are multiplexed with other signals, and may be used in several different ways. If the parallel port is disabled, then the 16 address and data pins become FLAG0-FLAG15. If the parallel port is in use, then these same 16 FLAG signals can be routed through the SRU, to 16 DAI pins. Finally, FLAG0-FLAG3 are available on four separate pins. These pins are shared with $\overline{\text{IRQ0-2}}$ and TIMEXP.

Configuring the parallel port pins to function as FLAG0-15 also causes these four dedicated pins to change to their alternate role, IRQ0-2 and TIMEXP.

Parallel Data Acquisition Port as Address Pins

PDAP use of AD[15:0] pins. When bit 26 of the IDP_PP_CTL register is set, the Parallel Data Acquisition Port (PDAP) reads from the parallel port's AD0-15 pins. When this bit is cleared, the PDAP reads data using DAI pins DAIP20-5. To use the parallel port, this bit must be cleared (= 0). For more information, see "Parallel Data Acquisition Port (PDAP)" on page 6-6.

Parallel Port Operation

This section describes how the parallel port transfers data. The SYSCTL and PPCTL registers control the parallel port operating mode. The bits in the SYSCTL register are listed in the *ADSP-2126x SHARC Processor Core Manual*. Table 3-3 on page 3-17 lists all the bits in the PPCTL register.

Basic Parallel Port External Transaction

A parallel port external transaction consists of a combination of an ALE cycle and a data cycle, which is either a read or write cycle. The following section describes parallel port operation as it relates to processor timing. Refer to the data sheet for your processor for detailed timing specifications.

An ALE cycle is an address latch cycle. In this cycle the $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals are inactive and ALE is strobed. The upper 16 bits of the address are driven onto the AD15-0 lines, and shortly thereafter the ALE pin is strobed, with AD15-0 remaining valid slightly after de-assertion to ensure a sufficient hold time for the external latch. The ALE pin always remains high for 2 x CCLK, irrespective of the data cycle duration values that are set in the

PPCTL register. The parallel port runs at 1/3 the CCLK rate, and so the ALE cycle is 3 x CCLK. An ALE cycle is inserted whenever the upper 16 bits of address differs from a previous access, as well as after the parallel port is enabled.

In a read cycle, the $\overline{\text{WR}}$ and ALE signals are inactive and $\overline{\text{RD}}$ is strobed. If the upper 16 bits of the external address have changed, this cycle is always preceded by an ALE cycle. In 8-bit mode, the lower 8 bits of the address, EA7-0, are driven on the AD15-8 pins, and data is sampled from the AD7-0 pins on the rising edge of $\overline{\text{RD}}$. In 16-bit mode, address bits are not driven in the read cycle, the external address is provided entirely by the external latch, and data is sampled from the AD15-0 pins at the rising edge of $\overline{\text{RD}}$. Read cycles can be lengthened by configuring the parallel port data cycle duration bits in the PPCTL register.

In a write cycle, $\overline{\text{RD}}$ and ALE are inactive and $\overline{\text{WR}}$ is strobed. If the upper 16 bits of the external address have changed, this cycle is always preceded by an ALE cycle. In 8-bit mode, the lower 8 bits of the address are driven on the AD15-8 pins and data is driven on the AD7-0 pins. In 16-bit mode, address bits are not driven in the write cycle, the external address is provided entirely by the external latch, 16-bit data is driven onto the AD15-0 pins, and data is written to the external device on the rising edge of the $\overline{\text{WR}}$ signal. Address and data are driven before the falling edge of $\overline{\text{WR}}$ and deasserted after the rising edge to ensure enough setup and hold time with respect to the $\overline{\text{WR}}$ signal. Write cycles can be lengthened by configuring the parallel port data cycle duration bits in the PPCTL register.

Reading From an External Device or Memory

The parallel port has a two stage data FIFO for receiving data (RXPP). In the first stage, a 32-bit register (PPSI) provides an interface to the external data pins and packs the 8- or 16-bit data into 32 bits. Once the 32-bit data is received in PPSI, the data is transferred into the second 32-bit reg-

ister (RXPP). Once the receive FIFO is full, the chip cannot initiate any more external data transfers. The RXPP register acts as the interface to the core or I/O processor (for DMA).

The PPTRAN bit must be zero in order to be read.

The order of 8 to 32-bit data packing is shown in Table 3-1. The first byte received is [7:0], second [15:8] and so on. The 16- to 32-bit packing scheme is shown in the third column of the table.

Table 3-1 does not show ALE cycles; it shows only the order of the data reads and writes.

Table 3-1. Packing Sequence for 32-Bit Data

Transfer	AD7–0, 8-bit to 32-bit (8-bit bus, LSW first)	AD15–0, 16-bit to 32-bit (16-bit bus, LSW first)
First	Word 1; bits 7–0	Word 1; bits 15–0
Second	Word 1; bits 15–8	Word 1; bits 31–16
Third	Word 1; bits 23–16	
Fourth	Word 1; bits 31–24	

Writing to an External Device or Memory

The parallel port has a two stage data FIFO for transmitting data (TXPP). The first stage (TXPP) is a 32-bit register that receives data from the internal memory via the DMA controller or a core write. The data in TXPP is moved to the second 32-bit register, PPSO. The PPSO register provides an interface to the external pins. Once a full word is transferred out of PPSO, TXPP data is moved to PPSO, if TXPP is not empty.

The PPTRAN bit of the PPCTL register must be set to one in order to enable writes to it.

The order of 32- to 8-bit data unpacking is shown in Table 3-2. The first byte transferred from PPSO is [7:0], the second [15:8] and so on. The 32-bit to 16-bit unpacking scheme is shown in column three of the table.

Table 3-2 does not show ALE cycles; it shows only the order of the data reads and writes.

Table 3-2. Unpacking Sequence for 32-Bit Data

Transfer	AD7–0, 32-bit to 8-bit (8-bit bus, LSW first)	AD15–0, 32-bit to 16-bit (16-bit bus, LSW first)
First	Word 1; bits 7–0	Word 1; bits 15–0
Second	Word 1; bits 15–8	Word 1; bits 31–16
Third	Word 1; bits 23–16	
Fourth	Word 1; bits 31–24	

Parallel port DMAs can only be performed to 32-bit (normal word) internal memory.

Transfer Protocol

The external interface follows the standard asynchronous SRAM access protocol. The programmable Data Cycle Duration (PPDUR) and optional Bus Hold Cycle (BHC) addition at the end of each data cycle are provided to interface with memories having different access time requirements. The data cycle duration is programmed via the PPDUR bit in the PPCTL register. The hold cycle at the end of the data cycle is programmed via the PPBHC bit in the PPCTL register.

Disabling the parallel port (PPEN bit is cleared) flushes both parallel port FIFOs, RXPP, and TXPP.

For standard asynchronous SRAM there are two transfer modes—8-bit and 16-bit mode. In 8-bit mode, the address range is 0x0 to 0xFFFFFF which is 16M bytes (4M 32-bit words). In 16-bit mode, the address range

is 0x0 to 0xFFFF which is a 128K bytes (32K 32-bit words). Although programs can initiate reads or writes on one and two byte boundaries, the parallel port always transfers 4 bytes (two 16-bit or four 8-bit words).

8-Bit Mode

An ALE cycle always precedes the first transfer of data after the parallel port is enabled. During ALE cycles for 8-bit mode, the upper 16 bits of the external address (EA23-8) are driven on the 16-bit parallel port bus (pins AD15-0). In data cycles (reads and writes), the processor drives the lower 8 bits of address EA7-0 on AD15-8. The 8 bits of external data, ED7-0, that are provided by AD7-0, are sampled by the $\overline{RD}$/$\overline{WR}$ signal respectively. The processor continues to receive and or send data with the same ALE cycle until the upper 16 bits of external address differ from the previous access. For consecutive accesses (EMPP = 1), this occurs once every 256 cycles. Figure 3-2 shows the connection diagram for the 8-bit mode.

Eight-bit mode enables a larger external address range.

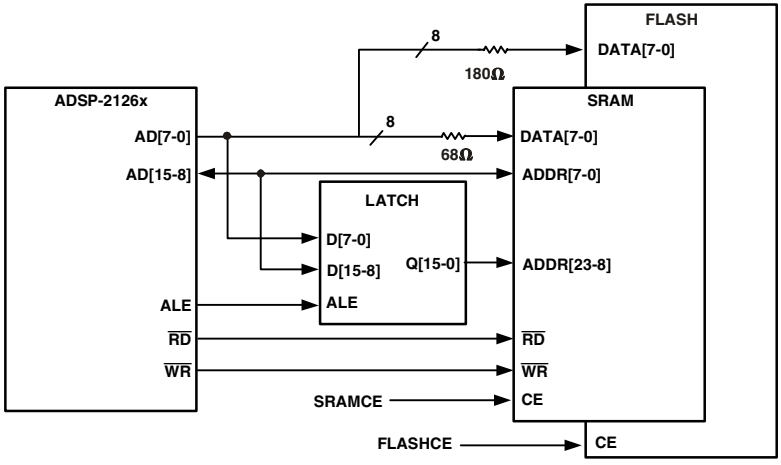


Figure 3-2. External Transfer—8-bit Mode

16-Bit Mode

In 16-bit mode, the external address range is EA15-0 (64K addressable 16-bit words). For a nonzero stride value (EMPP = 0), the transfer of data occurs in two cycles. In cycle one, the processor performs an ALE cycle, driving the 16 bits of external address, EA15-0, onto the 16-bit parallel port bus (pins AD15-0), allowing the external latch to hold this address. In the second cycle, the processor either drives or receives the 16 bits of external data (ED15-0) through the 16-bit parallel port bus (pins AD15-0). This pattern repeats until the transfer completes.

However, a special case occurs when the external address modifier is zero, (EMPP = 0). In this case, the external address is latched only once, using the ALE cycle before the first data transfer. After the address has been latched externally, the processor continues receiving and sending 16-bit data on AD15-0 until the transfer completes. This mode can be used with external FIFOs and high speed A/D and D/A converters and offers the maximum throughput available on the parallel port (132 Mbyte/sec).

Figure 3-3 shows the connection diagram in 16-bit mode.

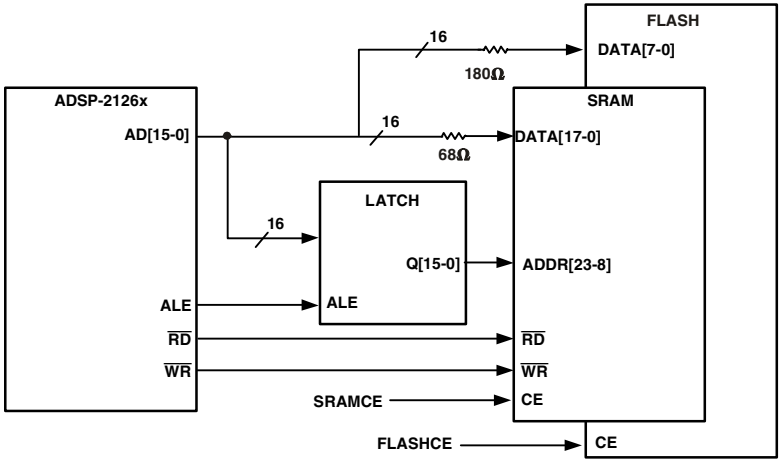


Figure 3-3. External Transfer—16-bit Mode

Comparison of 16-Bit and 8-Bit SRAM Modes

When considering whether to employ the 16- or 8-bit mode in a particular design, a few key points should be considered.

- The 8-bit mode provides a 24-bit address, and therefore can access 16M bytes of external memory. In contrast, the 16-bit mode can only address 64K x 16 bit words, which is equivalent to 128K bytes. Therefore, the 8-bit mode provides 128 times the storage capacity of the 16-bit mode.
- For sequential accesses, the 8-bit mode requires only one ALE cycle per 256 bytes. With minimum wait states selected, this represents a worst case overhead of:
  (1 ALE cycle)/(256 accesses + 1 ALE) x 100% = 0.39% overhead for ALE cycles. In contrast, the 16-bit mode requires one ALE cycle per external sequential access. Regardless of length (N), this represents a worst case overhead of:
  (N ALE cycles)/(N accesses + N ALE cycles) x 100% = 50% overhead for ALE cycles. However, the 16-bit mode delivers two bytes per cycle. Therefore, the total data transfer speed for sequential accesses is nearly identical for both 8-bit and 16-bit modes.

The question that arises at this point is: If the total transfer rates are the same for both 8-bit and 16-bit modes, and the 8-bit mode can also address 128 times as much external memory, why would a system use the 16-bit mode?

- Sometimes an external device is only capable of interfacing to a 16-bit bus.
- When the DMA external modifier is set to zero, the address does not change after the first cycle, therefore an ALE cycle is only inserted on the first cycle. In this case, the 16-bit port can run

twice as fast as the 8-bit port, as the overhead for ALE cycles is zero. This is convenient when interfacing to high speed 16-bit FIFO-based devices, including A/D and D/A converters.

- In situations where a majority of address accesses are non-sequential and cross 256 byte boundaries, the overhead of the ALE cycles in the 8-bit mode approaches 20%[1]. In this particular situation, the 16-bit memory can provide a 40% speed advantage over the 8-bit mode.

Parallel Port Interrupt

The parallel port has one interrupt signal, PPI, (bit 3 in the LIRPTL register). When DMA is enabled, the maskable interrupt PPI occurs when the DMA block transfer has completed (when the DMA Internal Word Count register ICPP decrements to zero). When DMA is disabled, the maskable interrupt is latched in every cycle the receive buffer is not empty or the transmit buffer is not full.

The parallel port receive (RXPP) and transmit (TXPP) buffers are memory mapped IOP registers. The PPI bit is located at vector address 0x50. The latch (PPI), mask (PPIMSK) and mask pointer (PPIMSKP) bits associated with the parallel port interrupt are all located in the LIRPTL register.

Parallel Port Throughput

As described in "Parallel Port Operation", each 32-bit word transferred through the parallel port takes a specific period of time to complete. This throughput depends on a number of factors, namely parallel port speed

[1] This can be realized by recalling that four bytes must be packed/unpacked into a single 32-bit word. For example when a 32-bit word is written/read, there is a single ALE cycle inserted per four consecutive addresses. This results in: (N/4 ALE cycles)/(N accesses + N/4 ALE cycles) x 100% = 20%.

(1/3 core instruction rate), memory width (8 bits or 16 bits), and memory access constraints (occurrence of ALE cycles at page boundaries, duration of data cycles, and/or addition of hold time cycles).

The maximum parallel port speed is 1/3 of the core. The relationship between core clock and parallel port speed is static. For a 200 MHz core clock, the parallel port runs at 66 MHz. Since there is no parallel port clock signal, it is easiest to think of parallel port throughput in terms of core clock cycles.

As described in "Parallel Port Operation", parallel port accesses require both ALE cycles to latch the external address and additional data cycles to transmit or receive data. Therefore, the throughput on the parallel port is determined by the duration and number of these cycles per word. The duration of each type of cycle is shown below and the frequency is determined by the external memory width.

There is one case where the frequency is also determined by the external address modifier register (EMPP).

- ALE cycles are fixed at 3 core cycles (CCLK) and are not affected by the PPDUR or BHC bit settings. In this case, the ALE is high for 2 core clock cycles. Address for ALE is set up a half core clock cycle before ALE goes HIGH (active) and remains on bus a half cycle after ALE goes LOW (inactive). Therefore, the total ALE cycles on the bus are 1/2 + 2 + 1/2 = 3 core clock cycles. Please refer to the data sheet for more precise timing characteristics.
- Data cycle duration is programmable with a range of 3 to 31 CCLK cycles. They may range from 4 to 32 cycles if the BHC bit is set (=1).

The following sections show examples of transfers that demonstrate the expected throughput for a given set of parameters. Each word transfer sequence is made up of a number of data cycles and potentially one additional ALE cycle.

8-Bit Access

In 8-bit mode, the first data-access (whether a read or a write) always consists of one ALE cycle followed by four data cycles. As long as the upper 16 bits of address do not change, each subsequent transfer consists of four data cycles. The ALE cycle is inserted only when the parallel port address crosses an 8-bit boundry page, in other words, after every 256 bytes that are transferred.

For example, if PPDUR3, BHC = 0, and the processor is in 8-bit mode. The first byte on a new page takes six core cycles (three for the ALE cycle and three for the data cycle), and the next sequential 255 bytes consume three core cycles each.

Therefore, the average data rate for a 256 byte page is:

(3 CCLK x 255 + 6 CCLK x 1) / 256 = 3.01 core clock cycles per byte.

For a 200 MHz core, this results in:

(200M CCLK /sec) x (1 byte/3.008 CCLK) = 66.4M Bytes/sec

16-Bit Access

In 16-bit mode, every word transfer consists of two ALE cycles and two data cycles. Therefore, for every 32-bit word transferred, at least six CCLK cycles are needed to transfer the data plus an additional six CCLK cycles for the two ALE cycles, for a total of 12 CCLK cycles per 32-bit transfer (four bytes). For a 200 MHz core clock, this results in a maximum sustained data rate device of:

200 MHz /12 = 16.67 Million 32-bit words/sec = 66.6M Bytes/sec

There is a specific case which allows this maximum rate to be exceeded. If the external address modifier (EMPP) is set to a stride of zero, then only one ALE cycle is needed at the very start of the transfer. Subsequent words, essentially written to the same address, do not require any ALE cycles, and every parallel port cycle may be a 16-bit data cycle. In this case, the throughput is nearly doubled (except for the very first ALE cycle) to over 132M bytes per second. This mode is particularly useful for interfacing to FPGA's or other memory-mapped peripherals such as DAC/ADC converters.

Conclusion

For sequential accesses, the average data rates are nearly identical in 8- and 16-bit modes. For help deciding between the two modes, please refer to "Comparison of 16-Bit and 8-Bit SRAM Modes" on page 3-11.

Parallel Port Registers

The ADSP-2126x processor's parallel port contains several user-accessible registers. The Parallel Port Control Register, PPCTL, contains control and status bits and is described below. Two additional registers, RXPP and TXPP, are used for buffering receive and transmit data during DMA operations and can be accessed by the core. Finally, the following registers are used for every parallel port access (both core-driven and DMA-driven).

- "Parallel Port DMA Start External Index Address Register (EIPP)" on page A-59
- "Parallel Port DMA External Modifier Address Register (EMPP)" on page A-59

For DMA transfers only, the following registers must also be initialized:

- "Parallel Port DMA Internal Word Count Register (ICPP)" on page A-59
- "Parallel Port DMA Start Internal Index Address Register (IIPP)" on page A-59
- "Parallel Port DMA Internal Modifier Address Register (IMPP)" on page A-59
- "Parallel Port DMA External Word Count Register (ECPP)" on page A-60

Additional information on Parallel Port registers can be found in "Parallel Port Registers" on page A-54.

Parallel Port Control Register (PPCTL)

The Parallel Port Control (PPCTL) register is a memory-mapped register located at address 0x1800 and is used to configure and enable the parallel port system. This register also contains status information for the TX/RX FIFO, the state of DMA, and for external bus availability. This read/write register is also used to program the data cycle duration and to determine the data transfer format.

Table 3-3 provides the bit descriptions for the PPCTL register.

Parallel Port DMA Registers

The following registers require initialization only when performing DMA-driven accesses.

- DMA Start Internal Index Address Register (IIPP)

  This 19-bit register contains the offset from the DMA starting address of 32-bit internal memory.

Table 3-3. Parallel Port Register (PPCTL) Bit Definitions

Bit	Name	Definition	Default
0	PPEN	**Parallel Port Enable.** Enables (if set, =1) or disables (if cleared, =0) the parallel port. Clearing this bit clears the FIFO and the parallel status information. If an $\overline{RD}$, $\overline{WR}$, or ALE cycle has already started, it completes normally before the port is disabled. The parallel port is ready to transmit or receive two cycles after it is enabled. An ALE cycle always occurs before the first read or write cycle after PPEN is enabled.	0
5–1	PPDUR	**Parallel Port Duration.** The duration of Parallel Port data cycles is determined by these bits. ALE cycles are not affected by this setting and are fixed at 3 CCLK cycles. 00000 = Reserved 00001 = Reserved 00010 = 3 clock cycles; 66 MHz throughput 00011 = 4 clock cycles; 50 MHz throughput 00100 = 5 clock cycles; 40 MHz throughput 00101 = 6 clock cycles; 33 MHz throughput ... 11111 = 32 clock cycles; 6.25 MHz throughput	Bit 1 = 1 Bit 2 = 1 Bit 3 = 1 Bit 4 = 0 Bit 5 = 1
6	PPBHC	**Bus Hold Cycle.** If set (=1), this causes every data-cycle to be prolonged for 1 CCLK period. If cleared (=0) no bus hold cycle occurs, and the duration of data-cycle is exactly the value specified in PPDUR. Bus hold cycles do not apply to ALE cycles which are always 3 CCLK cycles.	1
7	PP16	**Parallel Port External Data Width.** Sets the external data width to 16 bits (if set, =1) or 8 bits (if cleared, =0).	0
8	PPDEN	**Parallel Port DMA Enable.** Enables (if set, =1) DMA on the parallel port or disables DMA (if cleared, =0). When PPDEN is cleared, any DMA requests already in the pipeline complete, and no new DMA requests are made. This does not affect FIFO status.	0
9	PPTRAN	**Parallel Port Transmit/Receive Select.** Indicates whether the processor is reading from external memory (if cleared, =0) or writing to external memory (if set, =1).	0

Table 3-3. Parallel Port Register (PPCTL) Bit Definitions (Cont'd)

Bit	Name	Definition	Default
11–10	PPS	**Parallel Port FIFO Status.** These read-only bits indicate the status of the parallel port FIFO: 00 = RXPP/TXPP is empty 01 = RXPP/TXPP is partially full 11 = RXPP/TXPP is full	0
12	PPBHD	**Parallel Port Buffer Hang Disable.** When this bit is cleared (=0), core stalls occur normally. The core stall occurs when the core attempts to write to a full transmit buffer or read from the empty receive buffer. This bit prevents a core hang, when set (=1). Old data present in the receive buffer is reread if the core tries to read it. If a write to the transmit buffer is performed, the core overwrites the current data in the buffer.	0
13	PPALEPL	**Parallel Port ALE Polarity Level.** Asserts ALE active low (if set, =1) or active high (if cleared, =0).	0
15–14	Reserved		
16	PPDS	**DMA Status.** This read-only bit indicates that the internal DMA interface is active (if set, =1) or not active (if cleared, =0).	0
17	PPBS	**Parallel Port Bus Status.** Indicates that the external bus interface is busy (if set, =1) or available (if cleared, =0). The bus will be "busy" for the duration of the 32-bit transfer, including the ALE cycles. Note: This bit goes high 2 cycles after data is ready to transmit (after a data is written to PPTX, after PPRX is read with PPEN=1, after writing PPCTL to have PPEN=1 and PPDEN=1).	0
31–18	Reserved		

- DMA Internal Modifier Address register (IMPP)

  This 16-bit register contains the internal memory DMA address modifier.

- DMA Internal Word Count register (ICPP)

This 16-bit register contains the number of words in internal memory to be transferred via DMA.

- Parallel Port DMA External Word Count Register (ECPP)

  This 24-bit register contains the number of words in external memory to be transferred via DMA.

Parallel Port External Setup Registers

The following registers must be initialized for both core-driven and DMA-driven transfers.

- Parallel Port DMA External Index Address Register (EIPP)

  This 24-bit register contains the external memory byte address used for core-driven and DMA driven transfers.

- Parallel Port External Address Modifier Register (EMPP)

  This 2-bit register contains the external memory DMA address modifier. It supports only +1, 0, –1. After each data cycle, the EIPP register is modified by this value.

Using the Parallel Port

There are a number of considerations to make when interfacing to parallel external devices. This section describes the different the ways that the parallel port can be used to access external devices. Considerations for choosing between an 8-bit and a 16-bit wide interface are discussed in "Comparison of 16-Bit and 8-Bit SRAM Modes" on page 3-11.

External parallel devices can be accessed in two ways, either using DMA-driven transfers or core-driven transfers. DMA transfers are performed in the background by the I/O Processor and are generally used to move blocks of data. To perform DMA transfers, the address, word-count,

and address-modifier are specified for both the source and destination buffers (one internal, one external). Once initiated, (by setting PPEN = 1 and PPDEN = 1), the IOP performs the specified transfer in the background without further core interaction. The main advantage of DMA transfers over core driven transfers is that the core can continue executing code while sequential data is imported/exported in the background.

Unlike the external port on previous SHARC processors, the ADSP-2126x core cannot directly access the external parallel bus. Instead, the core initializes two registers to indicate the external address and address-modifier and then accesses data through intermediate registers. Then, when the core accesses either the PPTX or PPRX registers, the parallel port writes/fetches data to/from the specified external address. The details of this functionality and the four main techniques to manage each transfer are detailed below. In general, core-driven transfers are most advantageous when performing single-word accesses and/or accesses to non-sequential addresses.

DMA Transfers

To use the parallel port for DMA programs, start by setting up values in the DMA parameter registers. The program then writes to the PPCTL register to enable PPDEN with all of the necessary settings like cycle duration value, transfer direction, and so on. While a parallel port DMA is active, the DMA parameter registers are not writable. Furthermore, only the PPEN and DMAEN bits (in the PPCTL register) can be changed. If any other bit is changed, the parallel port will malfunction. It is recommended that both the PPDEN and PPEN bits be set and reset together to ensure proper DMA operation.

To see an example program that sets up a parallel port DMA, see Listing 3-1 on page 3-25.

Core Driven Transfers

Core-driven transfers can be managed using four techniques. The transfers can 1) use interrupts, 2) poll status bits in the PPCTL register, 3) predict when each access will complete by calculating the data and ALE cycle durations, or 4) rely on the fact that the core stalls on certain accesses to PPRX and PPTX. For all four of these methods, the core uses the same basic steps to initiate the transfer. However, each method uses a different technique to complete it. The following steps provide the basic procedure for setting up and initiating a data transfer using the core.

1. Write the external byte address to the EIPP register and the external address modifier to the EMPP register.

   Before initializing or modifying any of the parallel port parameter registers such as EIPP and EMPP, the parallel port must first be disabled (bit 0, PPEN, of the PPCTL register must be cleared). Only when PPEN=0, can those registers be modified and the port then re-enabled. This sequence is most often used to perform non-sequential, external transfers, such as when accessing taps in a delay line.

   For core-driven transfers, the ECPP, IIPP, IMPP, and ICPP are not used. Although these registers are automatically updated by the parallel port (the ECPP register decrements for example), they may be left uninitialized without consequence.

2. Initialize the PPCTL register with the appropriate settings.

   These include the parallel port data-cycle duration (PPDUR) and whether the transfer is a receive or transmit operation (PPTRAN). For core-driven transfers, be sure to clear the DMA enable bit, PPDEN. In this same write to PPCTL, the port may also be enabled by setting bit 0, PPEN, to 1.

When enabling the parallel port (setting PPEN = 1), the external bus activity varies, depending on the direction of data transfer (receive or transmit). For transmit operations (PPTRAN = 1), the parallel port does not perform any external accesses until valid data is written to the TXPP register by the core.

For read operations (PPTRAN = 0), two core clock cycles after PPEN is set (=1), the parallel port immediately fetches two 32-bit data words from the external byte address indicated by EIPP. Subsequently, additional data is fetched only when the core reads (empties) RXPP.

The following are guidelines that programs must follow when the processor core accesses parallel port registers.

- While a DMA transfer is active, the core may only write the PPEN and PPDEN bits of PPCTL. Accessing any of the DMA parameter registers or other bits in PPCTL during an active transfer will cause the parallel port to malfunction.
- Core reads of the FIFO register during a DMA operation are allowed but do not affect the status of the FIFO.

  If PPEN is cleared while a transfer is underway (whether core or DMA-driven), the current external bus cycle (ALE cycle or data cycle) will complete but no further external bus cycles occur. Disabling the parallel port clears the data in the RXPP and TXPP registers.
- Core reads and writes to the TXPP and RXPP registers update the status of the FIFO when DMA is not active. This happens even when the parallel port is disabled.
- The PPCTL register has a two-cycle effect-latency. This means that if programs write to this register in cycle N, the new settings will not be in effect until cycle N + 2. Avoid sampling PPBS until at least 2 cycles after the PPEN bit in PPCTL is set.

- For core-driven transfers over the parallel port, the IIPP, IMPP, ICPP, and ECPP registers are not used. Only the EIPP and EMPP registers need to be initialized before accessing the TXPP or RXPP buffers.

Known Duration Accesses

Of these methods, known duration accesses are the most efficient because they allow the core to execute code while the transfer to/from the RXPP or TXPP occurs on the external bus. For example, after the core reads the PPTX register, it will take some number N core-cycles for the PP to shift out that data to the memory. During that time, the core can go on doing other tasks. After N core-cycles have passed, the parallel port may be disabled and the external address register updated for another access.

To determine the duration for each access, the designer simply add's the number of data-cycles and the duration of each (measured in CCLK cycles) along with the number of ALE cycles (which are fixed at 3 CCLK cycles). This duration is deterministic, and is based on two settings in the PPCTL register—parallel port data-cycle duration (PPDUR) and Bus Hold Cycle Enable (PPBHC).

Please refer to "Parallel Port Operation" for further explanation of the parallel port bus cycles, but in summary, programs can use the following values:

- each ALE cycle is fixed at 3 CCLK cycles, regardless of the PPDUR or PPBHC settings.
- each Data cycle is the setting in the PPDUR register (+1 if PPBHC =1)

For example, in 8-bit mode, a single-word transfer is comprised of 1 ALE cycle and 4 Data cycles. If PPDUR3 is used (the fastest case) and PPBHC = 0, this transfer completes in:

(1 ALE-cycle x 3 CCLK) + (4 data-cycles x 3 CCLK) = 15 core cycles per 32-bit word.

This means that 15-instructions after data is written to TXPP or read from RXPP, the parallel port has finished writing/fetching that data externally, and the parallel port may be disabled. This case is shown in Listing 3-3 on page 3-29.

Status Driven Transfers (Polling)

The second method that the core may use to manage parallel port transfers involves the status bits in PPCTL register, specifically the Parallel Port Bus Status (PPBS) bit. This bit reflects the status of the external address pins AD0-AD15 and is used to determine when it is safe to disable and modify the parallel port. The PPBS bit is set to 1 at the start of each transfer, and is cleared once the entire 32-bit word has been transmitted/received.

Core-Stall Driven Transfers

The final method of managing parallel port transfers simply relies on the fact that the core will stall execution when reading from an empty RX buffer and when writing to a full TX buffer. This technique can only be used for accesses to sequential addresses in external memory. For sequential external addresses, the parallel port does not need to be disabled after each word in order to manually update the EIPP register. Instead, the external address that is automatically incremented by the modifier (EMPP) register on each access is used.

Interrupt Driven Accesses

With interrupt-driven accesses, parallel port interrupts are generated on a word-by-word basis, rather than on a block transfer basis, as is the case when DMA is enabled. In this non-DMA mode, the interrupt indicates to the core that it is now safe to read a word from the RXPP buffer or to write a word to the TXPP buffer (depending on the value of the PPTRAN bit).

To facilitate this, the PPI (latch) bit of the LIRPTL register is set to one in every core cycle where the TXPP buffer is not full or, in receive mode, in every core cycle in which the RXPP buffer has valid data. When fast 16-bit wide parallel devices are accessed, there may be as few as 10 core cycles between each transfer. Because of this, interrupt-driven transfers are usually the least efficient method to use for core-driven accesses. Interrupt driven transfers are most valuable when parallel port data-cycle durations are very long (allowing the core may do some work between accesses). Generally, interrupts are the best choice for DMA-driven parallel port transfers rather than core-driven transfers.

Parallel Port Programming Examples

This section provides two programming examples written for the ADSP-21262 processor. The first, Listing 3-1, uses the parallel port to transfer a buffer to 16-bit external memory using DMA. The second example Listing 3-2, uses the parallel port to transfer a buffer to 8-bit external memory using status driven core writes. The last example, shows a calculated duration example of core driven parallel port access.

Listing 3-1. Parallel Port DMA Buffer Transfer

```
/* Register Definitions */
#define PPCTL       0x1800
#define EIPP        0x1810
#define EMPP        0x1811
#define ECPP        0x1812
#define IIPP        0x1818
#define IMPP        0x1819
#define ICPP        0x181a

/* Register Bit Definitions */
#define PPEN    0x00000001
```

```
#define PPDUR20 0x00000026
#define PPBHC   0x00000040
#define PP16    0x00000080
#define PPDEN   0x00000100
#define PPTRAN  0x00000200
#define PPBS    0x00020000
/* Source Buffer */
.section/dm seg_dmda;
.var source[8] = 0x11111111,
                 0x22222222,
                 0x33333333,
                 0x44444444,
                 0x55555555,
                 0x66666666,
                 0x77777777,
                 0x88888888;

.global _main;
.section/pm seg_pmco;
_main:
ustat3 = dm(PPCTL);       /*disable parallel port*/
bit clr ustat3 PPEN|PPDEN;
dm(PPCTL) = ustat3;

/* initiate parallel port DMA registers*/
r0 = source;          dm(IIPP) = r0;
r0 = 1;               dm(IMPP) = r0;
r0 = LENGTH(source);  dm(ICPP) = r0;

r0 = 1;               dm(EMPP) = r0;
r0 = 0x1000000;       dm(EIPP) = r0;
/* For 16-bit external memory, the External count is
   double the internal count */
```

```
r0 = LENGTH(source) * 2;      dm(ECPP) = r0;

ustat3 = PP16|      /*for a 16-bit external memory */
        PPTRAN|    /* transmit (write) */
        PPBHC|     /* implement a bus hold cycle*/
        PPDUR20;   /* make pp data cycles last for a duration
                      of 20 cclk cycles */
dm(PPCTL) = ustat3;

/* initiate PP DMA*/
/*Enable Parallel Port and PP DMA in same cycle*/
ustat4 = dm(PPCTL);
bit set ustat4 PPDEN|PPEN;
dm(PPCTL) = ustat4;

_main.end: jump(pc,0);
```

Listing 3-2. Parallel Port Status Driven Core Transfer

```
/* Register Definitions */
#define PPCTL       0x1800
#define TXPP        0x1808
#define RXPP        0x1809
#define EIPP        0x1810
#define EMPP        0x1811
#define ECPP        0x1812

/* Register Bit Definitions */
#define PPEN    0x00000001
#define PPDUR20 0x00000026
#define PPBHC   0x00000040
#define PPTRAN  0x00000200
#define PPBS    0x00020000
```

```
/* Source Buffer */
.section/dm seg_dmda;
.var source[8] = 0x11111111,
                 0x22222222,
                 0x33333333,
                 0x44444444,
                 0x55555555,
                 0x66666666,
                 0x77777777,
                 0x88888888;

/* Main code section */
.global _main;
.section/pm seg_pmco;
_main:
i4 = source;
m4 = 1;

/* setup ppdma registers for core use */
r0 = 1;               dm(EMPP) = r0;
r0 = 0x1000000;       dm(EIPP) = r0;
/* For 8-bit external memory, the External count is
    four times the internal count */
r0 = LENGTH(source) * 4;      dm(ECPP) = r0;

ustat3 = PPEN|      /* enable port */
         PPTRAN|    /* transmit (write) */
         PPBHC|     /* implement a bus hold cycle*/
         PPDUR20;   /* make pp data cycles last for a */
                    /* duration of 20 cclk cycles */
dm(PPCTL) = ustat3;
/* loop to write 10 words into TXPP */
lcntr = 10, do core_writes until lce;
write:
```

```
r0 = dm(i4,m4);
core_writes: dm(TXPP) = r0;

/* poll to ensure parallel port has completed the transfer */
waiting: ustat4 = dm(PPCTL);
bit tst ustat4 PPBS;
if tf jump waiting;
_main.end: jump(pc,0);
```

Listing 3-3. Calculated Duration Core Driven Access

```
_main:
 /* Setup once=========================== */
    ustat3 = PPDUR3 | PPTRAN | PPEN; /* ustat3 enables PP */
    ustat4 = PPDUR3 | PPTRAN;        /* ustat4 disables PP */

dm(PPCTL) = ustat4;       /*  initialize but disable PP */

/* NOTE: Internal DMA registers AND the EXTERNAL COUNT can be
left uninitialized for Core-driven transfers (External count
determined by bus width: 16bit = count of 2, 8-bit = count of 4,
since internal width always = 32-bits.) */

      r0=1;   dm(EMPP)=r0;  /* don't move external ptr */

      /* initialize external address and sample-to-write */
        r1=EZKIT_SRAM_BASE_ADDR;   /* initialize R1 w/ first
                                      ext. byte address */
        r2=0x33221100;        /* and R2 w/ first data to be
                                 written /*
/* ======================================== */

/* for testing */    do (write_loop.end - 1) until forever;
```

```
/* ===Instructions required for each 32-bit word written===*/
/* (18 instructions per sample = 4 cycles overhead + 14 cycles
     work) /*
/*  R1 holds external byte address to be written */
/*  R2 holds data to be written */

write_loop:

    dm(EIPP) = r1;
    dm(PPCTL)= ustat3;  /* enable PP */
    dm(TXPP) = r2;      /*  <-- write to PP FIFO */

/* -----14 core cycles (minimum) available while each word is
being transmitted. Writting to PPCTL has a 2 cycle
effect-latency, so the result of writing this register in the
14th cycle doesn't take effect until the 16th cycle - which is
one cycle after the cycle completes----- */

/* (NOTE: Modifying PP parameters before 14 cycles have passed
will cause the access to fail - Using more than 14 cycles is
fine. */

    nop;nop;
    nop;nop;
    nop;nop;
    nop;nop;
    nop;nop;
    nop;

  /*  update addr and data for next loop iteration:
      r0 = 4;
      r1 = R1 + r0;  /* next ext. destination address += 4
```

```
      r2 = r2 + 1;   /* next data to write /*
/*-------------------------------------------------------------*/
dm(PPCTL)=ustat4;  /*  <-- 14 cycles later, it's safe to
                       disable/alter PP because each access takes
                       15 CCLK cycles, and writing PPCTL has a
                       2-cycle effect-latency. */
                   /* NOTE: PPEN must be cleared before modifying
                      EIPP /*
============================================================
============
```

4 SERIAL PORTS

The ADSP-2126x processors have up to six independent, synchronous serial ports (SPORTs) that provide an I/O interface to a wide variety of peripheral devices. Each serial port 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 number of serial ports varies depending on the specific processor model you are using. This chapter was written using six serial ports for examples. Programs need to be written accordingly.

Serial ports can operate at one-quarter the full clock rate of the processor, at a maximum clock rate of n/4M bit/s, where n equals the processor core-clock frequency (CCLK). If channels A and B are active, each SPORT has 100M bit/s maximum throughput. Bidirectional (transmit or receive) functions provide greater flexibility for serial communications. Serial port data can be automatically transferred to and from on-chip memory using DMA block transfers. In addition to standard synchronous serial mode, each serial port offers a Time Division Multiplexed (TDM) multichannel mode, Left-justified Sample Pair mode, and I^2S mode.

Serial ports offer the following features and capabilities:

- 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.

- All serial data signals have programmable receive and transmit functions and thus have one transmit and one receive data buffer register (double-buffer) and a bidirectional shift register associated with each serial data signal. Double-buffering provides additional time to service the SPORT.
- μ-law and A-law compression/decompression hardware companding on transmitted and received words.
- An internally-generated serial clock and frame sync provide signals in a wide range of frequencies. Alternately, the SPORT can accept clock and frame sync input from an external source, as described in Figure 4-8 on page 4-63.
- Interrupt-driven, single word transfers to and from on-chip memory controlled by the processor core, described in "Single Word Transfers" on page 4-74.
- DMA transfers to and from on-chip memory. Each SPORT can automatically receive or transmit an entire block of data.
- Chained DMA operations for multiple data blocks, see "Chaining DMA Processes" on page 2-10.
- Four operation modes: DSP Standard Serial, Left-justified Sample Pair, I^2S, and multichannel. In standard DSP serial, Left-justified Sample Pair, and I^2S modes, when both A and B channels are used, they transmit or receive data simultaneously, sending or receiving bit 0 on the same edge of the serial clock, bit 1 on the next edge of the serial clock, and so on. In multichannel mode, SPORT1, 3 or 5 can receive A and B channel data, and SPORT0, 2 or 4 transmits A and B channel data selectively from up to 128 channels of a TDM serial bitstream. This mode is useful for H.100/H.110 and other

telephony interfaces. In multichannel mode, SPORT0 and SPORT1 work as a pair, SPORT2 and SPORT3 work as a pair, and SPORT4 and SPORT5 work as a pair. See "SPORT Operation Modes" on page 4-9.

When programming the serial port channel (A or B) as a transmitter, only the corresponding transmit buffers TXSPxA and TXSPxB become active, while the receive buffers (RXSPxA and RXSPxB) remain inactive. Similarly, when SPORT channels A and B are programmed to receive, only the corresponding RXSPxA and RXSPxB buffers are activated.

SPORTs are forced into pairs when in multichannel mode. For more information, see "Multichannel Operation" on page 4-24.

- The serial ports are configurable for transferring data words between 3 and 32 bits in length, either most significant bit (MSB) first or least significant bit (LSB) first. Words must be between 8 and 32 bits in length for I^2S and Left-justified Sample Pair mode. Refer to "Data Word Formats" on page 4-39 and the individual SPORTs operation mode sections for additional information.
- 128-channel TDM is supported in multichannel mode operation, described in "Multichannel Operation" on page 4-24.

Receive comparison and 2-dimensional DMA are not supported in the ADSP-2126x processor.

The SPTRAN bit in the SPCTLx register affects the operation of the transmit or the receive data paths. The data path includes the data buffers and the shift registers. When SPTRAN = 0, the primary and secondary RXSPxy data buffers and receive shift registers are activated, and the transmit path is disabled. When SPTRAN = 1, the primary and secondary TXSPxy data buffers and transmit shift registers are activated, and the receive path is disabled.

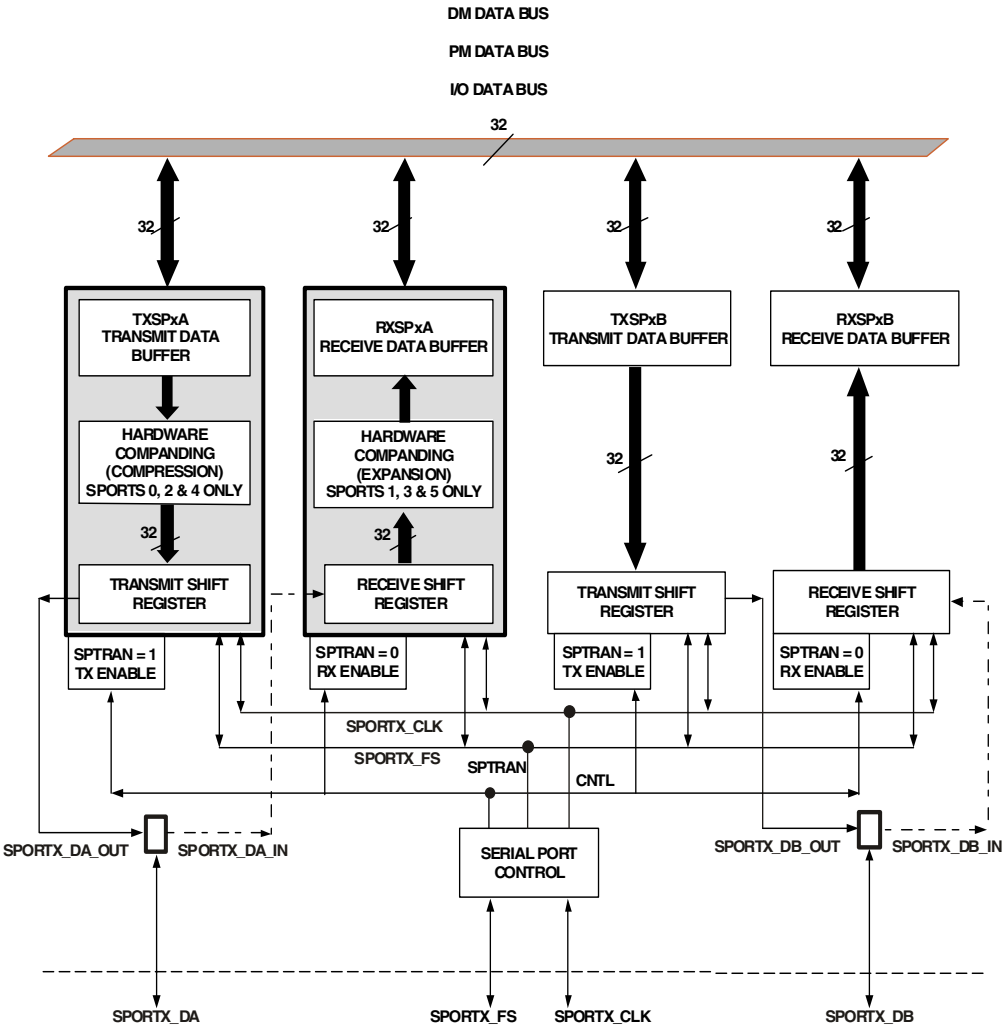


Figure 4-1. Serial Port Block Diagram

Serial Port Signals

Figure 4-2 shows all of the signals used in the serial ports.

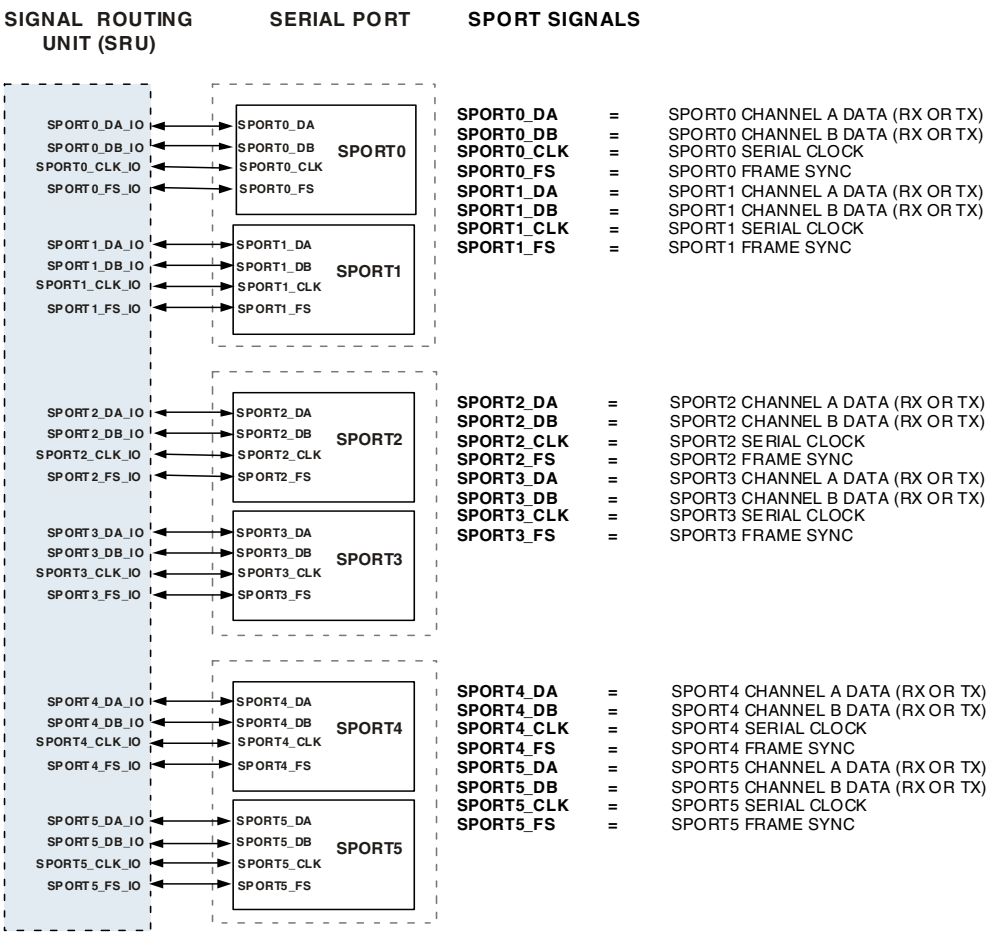


Figure 4-2. DSP Standard Serial Mode – Serial Port Signals

Pairings of SPORTs (0 and 1, 2 and 3, and 4 and 5) are only used in multichannel mode and loopback mode for testing.

Any 20 of these 24 signals can be mapped to Digital Audio Interface (DAI_Px) pins through the signal routing unit (SRU). For more information, see Chapter 7, Digital Audio Interface., Table A-15 on page A-63, and Table A-16 on page A-68.

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. Therefore, the 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. See Figure 4-8 on page 4-63 for more details.

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. See Figure 4-8 on page 4-63 for more details.

Figure 4-1 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 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.

If the serial port is configured as a serial transmitter, the data transmitted is written to the TXSPxA/TXSPxB buffer. The data is (optionally) companded in hardware on the primary A channel (SPORT 0, 2, and 4 only), then automatically transferred to the transmit shift register, because companding is not supported on the secondary B channels. The data in the shift register is then shifted out via the SPORT's SPORTx_DA or SPORTx_DB signal, synchronous to the SPORTx_CLK clock. If framing signals are used, the SPORTx_FS signal indicates the start of the serial word transmission. The SPORTx_DA or SPORTx_DB signal is always driven if the serial port is enabled (SPEN_A or SPEN_B = 1 in the SPCTLx control register), unless it is in multichannel mode and an inactive time slot occurs.

When the SPORT is configured as a transmitter (SPTRAN = 1), the TXSPxA and TXSPxB buffers, and the channel transmit shift registers respond to SPORTx_CLK and SPORTx_FS to transmit data. The receive RXSPxA and RXSPxB buffers, and the receive shift registers are inactive and do not respond to SPORTx_CLK and SPORTx_FS signals. Since these registers are inactive, reading from an empty buffer causes the core to hang indefinitely.

If the SPORTs are configured as transmitters (SPTRAN bit = 1 in SPCTL), programs should not read from the inactive RXSPxA and RXSPxB buffers. This causes the core to hang indefinitely since the receive buffer status is always empty.

If the serial data signal is configured as a serial receiver (SPTRAN = 0), the receive portion of the SPORT shifts in data from the SPORTx_DA or SPORTx_DB signal, synchronous to the SPORTx_CLK receive clock. If framing signals are used, the SPORTx_FS signal indicates the beginning of the serial word being received. When an entire word is shifted in on the primary A channel, the data is (optionally) expanded (SPORT1, 3, and 5 only), then automatically transferred to the RXSPxA buffer. When an entire word is shifted in on the secondary channel, it is automatically transferred to the RXSPxB buffer.

When the SPORT is configured as a receiver (SPTRAN = 0), the RXSPxA and RXSPxB buffers are activated along with the corresponding A and B channel receive shift registers, responding to SPORTx_CLK and SPORTx_FS for reception of data. The transmit TXSPxA and TXSPxB buffer registers and transmit A and B shift registers are inactive and do not respond to the SPORTx_CLK and SPORTx_FS. Since the TXSPxA and TXSPxB buffers are inactive, writing to a transmit data buffer causes the core to hang indefinitely.

If the SPORTs are configured as receivers (SPTRAN bit = 0 in SPCTLx), 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 out of the deactivated transmit data buffers.

The processor 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 FLG pins as asynchronous data receive and transmit signals. Examples of this can be found in the following documents.

- "Software UART", in *Digital Signal Processing Applications Using The ADSP-2100 Family*, Volume 2.
- Engineer-to-Engineer Note (EE-191), *Implementing a Glueless UART Using the SHARC DSP SPORTs*.

SPORT Operation Modes

Serial ports operate in four modes:

- Standard DSP Serial mode, described in "Standard DSP Serial Mode" on page 4-11
- Left-justified Sample Pair mode, described in "Left-Justified Sample Pair Mode" on page 4-14
- I^2S mode, described in "I2S Mode" on page 4-18
- Multichannel mode, described in "Multichannel Operation" on page 4-24

Bit names and their functionality change based on the SPORT operating mode. See the mode specific section for the bit names and their functions.

The SPORT operating mode can be selected via the SPCTLx register. See Table 4-1 for a summary of the control bits as they relate to the four operating modes.

The Operating mode bit (OPMODE) of SPCTLx register selects between I^2S mode, Left-justified Sample Pair mode, and non-I^2S mode (DSP Serial Port/Multichannel mode). In non-I^2S Multichannel mode, the MCEA bit in the SPMCTLxy register enables the A channels and the MCEB bit in the SPMCTLxy register enables the B channels. In addition to these bits, the Data Direction bit (SPTRAN) selects whether the port is a transmitter or receiver in non-multichannel mode.

If the SPTRAN bit is set (= 1), the SPORT becomes a transmitter and all the other control bits are defined accordingly. Similarly, when SPTRAN = 0, the SPORT becomes a receiver.

Companding is **not** supported in I^2S and Left-justified Sample Pair modes.

The SPCTLx register is unique in that the name and functionality of its bits changes depending on the operation mode selected. In each section that follows, the bit names associated with the operating modes are described. Table 4-1 provides values for each of the bits in the SPORT Serial Control (SPCTLx) registers that must be set in order to configure each specific SPORT operation mode. An X in a field indicates that the bit is not supported for the specified operating mode.

Table 4-1. SPORT Operation Modes

OPERATING MODES	Bits					
	OPMODE	LAFS	FRFS	MCEA	MCEB	SLENx
Standard DSP Serial Mode	0	0, 1	X	0	0	3-32[1]
I^2S (Tx/Rx on Left Channel First)	1	0	1	0	0	8-32
I^2S (Tx/Rx on Right Channel First)	1	0	0	0	0	8-32
Left-justified Sample Pair Mode (Tx/Rx on FS Rising Edge)	1	1	0	0	0	8-32
Left-justified Sample Pair (Tx/Rx on FS Falling Edge)	1	1	1	0	0	8-32
Multichannel A Channels	0	0	X	1	0	3-32[1]
Multichannel B Channels	0	0	X	0	1	3-32[1]
Multichannel A and B Channels	0	0	X	1	1	3-32[1]

1 Although serial ports process word lengths of 3 to 32 bits, transmitting or receiving words smaller than 7 bits at core clock frequency/4 of the processor may cause incorrect operation when DMA chaining is enabled. Chaining disables 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.

Standard DSP Serial Mode

The Standard DSP Serial mode lets programs configure serial ports for use by a variety of serial devices such as serial data converters and audio codecs. In order to connect to these devices, a variety of clocking, framing, and data formatting options are available.

Standard DSP Serial Mode Control Bits

Several bits in the SPCTLx Control register enable and configure standard DSP serial mode operation:

- Operation mode, Master mode enable (OPMODE)
- Word length (SLEN)
- SPORT enable (SPEN_A and SPEN_B)

For more information, see Appendix A, Registers Reference.

Clocking Options

In standard DSP serial mode, the serial ports can either accept an external serial clock or generate it internally. The ICLK bit in the SPCTL register determines the selection of these options (see "Clock Signal Options" on page 4-33 for more details). For internally-generated serial clocks, the CLKDIV bits in the DIVx register configure the serial clock rate (see Figure 4-8 on page 4-63 for more details).

Finally, programs can select whether the serial clock edge is used for sampling or driving serial data and/or frame syncs. This selection is performed using the CKRE bit in the SPCTL register (see Table A-5 on page A-26 for more details).

Frame Sync Options

A variety of framing options are available for the serial ports. For detailed descriptions of framing options, see "Frame Sync Options" on page 4-33. In this mode, these options are independent of clocking, data formatting, or other configurations. The frame sync signal (SPORTx_FS) is used as a framing signal for serial word transfers.

Framing is optional for serial communications. The FSR bit in the SPCTL register controls whether the frame sync signal is required for every serial word transfer or if it is used simply to start a block of serial word transfers. See "Framed Versus Unframed Frame Syncs" on page 4-34 for more details on this option. Similar to the serial clock, the frame sync can be an external signal or generated internally. The IFS bit in the SPCTL register allows the selection between these options. See the Internal Frame Sync Select bit description in Figure 4-8 on page 4-63 for more details. For internally-generated frame syncs, the FSDIV bits in the DIVx register configure the frame sync rate. For internally-generated frame syncs, it is also possible to configure whether the frame sync signal is activated based on the FSDIV setting and the transmit or receive buffer status, or by the FSDIV setting only.

All settings are configured through the DIFS bit of the SPCTL register. See "Data-Independent Frame Sync" on page 4-37 for more details. The frame sync can be configured to be active high or active low through the LFS bit in the SPCTL register. See "Active Low Versus Active High Frame Syncs" on page 4-35 for more details. The timing between the frame sync signal and the first bit of data either transmitted or received is also selectable through the LAFS bit in the SPCTL register. See "Early Versus Late Frame Syncs" on page 4-36 for more details.

Data Formatting

Several data formatting options are available for the serial ports in the DSP Standard Serial mode. Each serial port has an A channel and B channel available. Both can be configured for transmitting or receiving. The SPTRAN bit controls the configuration of transmit versus receive operations. Serial ports can transmit or receive a selectable word length, which is programmed by the SLEN bits in the SPCTL register. See "Setting Word Length (SLEN)" on page 4-15 for more details. Serial ports also include companding hardware built in to the A channels that allow sign extension or zero-filling of upper bits of the serial data word. These configurations are selected by the DTYPE bits in the SPCTL register. See "Data Type" on page 4-41 and "Companding" on page 4-42 for more information. The endian format (LSB versus MSB first) is selectable by the LSBF bit of the SPCTL register. See "Endian Format" on page 4-40 for more details. Data packing of two serial words into a 32-bit word is also selectable. The PACK bit in the SPCTL register controls this option. See "Data Packing and Unpacking" on page 4-40 for more details.

Data Transfers

Serial port data can be transferred for use by the processor in two different methods:

- DMA transfers
- Core-driven single word transfers

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. For more information on Sport DMA operations, see "DMA Block Transfers" on page 4-66. 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). See "SPORT Interrupts" on page 4-65 for more details.

Status Information

Serial ports provide status information about data buffers via the DXS_A and DXS_B status bits and error status via ROVF or TUVF bits in the SPCTL register. See "Serial Port Control Registers (SPCTLx)" on page 4-50 for more details.

Depending on the SPTRAN setting, these bits reflect the status of either the TXSPxy or RXSPxy data buffers.

Left-Justified Sample Pair Mode

Left-justified Sample Pair 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.

The programmer has 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. Set the Late Frame Sync bit (LAFS bit) = 1 for Left-justified Sample Pair mode. See Table 4-1 on page 4-10. Then, choose the frame sync edge associated with the first word in the frame sync cycle, using the FRFS bit (1 = Frame on Rising Frame Sync, 0 = Frame on Falling Frame Sync).

Refer to Table 4-1 on page 4-10 for additional information about specifying Left-justified Sample Pair mode.

In Left-justified mode, if both channels on a SPORT are set up to transmit, then the SPORT transmits on channels (TXSPxA and TXSPxB) simultaneously; each transmits a sample pair. If both channels on a SPORT are set up to receive, the SPORT receives channels (RXSPxA and RXSPxB) simultaneously. Data is transmitted in MSB-first format.

Multichannel operation and companding are not supported in Left-justified Sample Pair mode.

Each SPORT transmit or receive channel has a buffer enable, DMA enable, and chaining enable bits in its SPCTLx Control register. The SPORTx_FS signal is used as the transmit and/or receive word select signal. DMA-driven or interrupt-driven data transfers can also be selected using bits in the SPCTLx register.

Setting the Internal Serial Clock and Frame Sync Rates

The serial clock rate (CLKDIV value) for internal clocks can be set using a bit field in the CLKDIV register. For details, see Figure 4-8 on page 4-63.

Left-Justified Sample Pair Mode Control Bits

Several bits in the SPCTLx register enable and configure Left-justified Sample Pair mode operation:

- Operation mode (OPMODE)
- Channel enable (SPEN_A and SPEN_B)
- Word length (SLEN)
- Frame on Rising Frame Sync (FRFS)
- Master mode enable (MSTR)
- Late Frame Sync (LAFS)

For more information, see “Serial Port Registers” on page A-19.

Setting Word Length (SLEN)

SPORTs handle data words containing 8 to 32 bits in Left-justified mode. Programs need to set the bit length for transmitting and receiving data words. For details, see “Word Length” on page 4-39.

The transmitter sends the MSB of the next word in the same clock cycle as the word select (SPORTx_FS) signal changes.

To transmit or receive words continuously in Left-justified Sample Pair mode, load the FSDIV register with the same value as SLEN. For example, for 8-bit data words (SLEN = 7), set FSDIV = 7.

Enabling SPORT Master Mode (MSTR)

The SPORTs transmit and receive channels can be configured for Master or Slave mode. In Master mode, (MSTR = 1) the processor generates the word select and serial clock signals for the transmitter or receiver. In Slave mode, (MSTR = 0) an external source generates the word select and serial clock signals for the transmitter or receiver. For more information, see "Setting the Internal Serial Clock and Frame Sync Rates" on page 4-15.

Selecting Transmit and Receive Channel Order (FRFS)

Using the FRFS bit, it is possible to select which frame sync edge (rising or falling) that the SPORTs transmit or receive the first sample. See Table 4-1 on page 4-10 for more details.

Selecting Frame Sync Options (DIFS)

When using both SPORT channels (SPORTx_DA and SPORTx_DB) as transmitters and MSTR = 1, SPTRAN = 1, and DIFS = 0, the processor generates a frame sync signal only when both transmit buffers contain data because both transmitters share the same CLKDIV and SPORTx_FS. For continuous transmission, both transmit buffers must contain new data.

When using both SPORT channels as transmitters and MSTR = 1, SPTRAN = 1 and DIFS = 1, the processor generates a frame sync signal at the frequency set by FSDIVx whether or not the transmit buffers contain new data. The DMA controller or the application is responsible for filling the transmit buffers with data.

Enabling SPORT DMA (SDEN)

DMA can be enabled or disabled independently on any of the SPORT's transmit and receive channels. For more information, see "Moving Data Between SPORTS and Internal Memory" on page 4-66. Set `SDEN_A` or `SDEN_B` (=1) to enable DMA and set the channel in DMA-driven data transfer mode. Clear `SDEN_A` or `SDEN_B` (=0) to disable DMA and set the channel in an interrupt-driven data transfer mode.

Interrupt-Driven Data Transfer Mode

Both the A and B channels share a common interrupt vector, regardless of whether they are configured as transmitters or receivers.

The SPORT generates an interrupt in every core clock cycle when the transmit buffer has a vacancy or the receive buffer has data. To determine the source of an interrupt, applications must check the transmit or receive data buffer status bits. For details, see "Single Word Transfers" on page 4-74.

DMA-Driven Data Transfer Mode

Each transmitter and receiver has its own DMA registers. For details, see "Selecting Transmit and Receive Channel Order (FRFS)" on page 4-16 and "Moving Data Between SPORTS and Internal Memory" on page 4-66. The same DMA channel drives both samples in the pair for the transmitter or receiver. The software application must stop multiplexing the left and right channel data received by the receive buffer, because the left and right data is interleaved in the DMA buffers.

Channel A and B on each SPORT share a common interrupt vector. The DMA controller generates an interrupt at the end of DMA transfer only.

Figure 4-3 shows the relationship between frame sync (word select), serial clock, and Left-justified mode data. Timing for word select is the same as for frame sync.

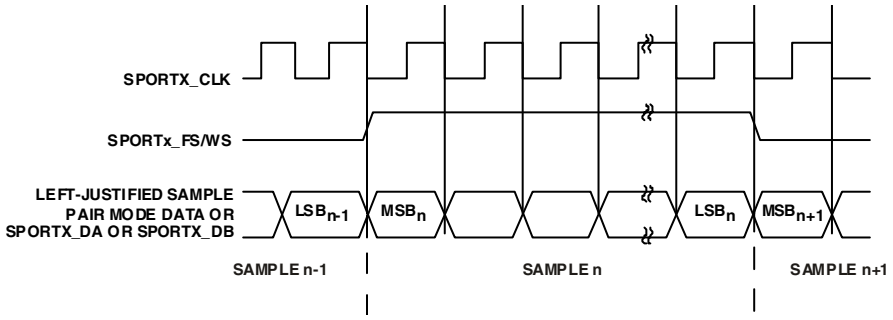


Figure 4-3. Word Select Timing in Left-justified Sample Pair Mode[1]

1 This figure illustrates only one possible combination of settings attainable in the Left-justified Sample Pair mode. In this example case, OPMODE =1, LAFS =1, and FRFS =1. For additional combinations, refer to Table 4-1 on page 4-10.

I²S Mode

I²S mode is a three-wire serial bus standard protocol for transmission of two-channel (stereo) Pulse Code Modulation (PCM) digital audio data, in which each sample is transmitted in MSB-first format. Many of today's analog and digital audio front-end devices support the I²S protocol including:

- 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, SP/DIF, IEC958, CP-340, and CP-1201
- Digital audio signal processors
- Dedicated digital filter chips
- Sample rate converters

The I^2S bus transmits audio data 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 SPORT transmit channels (TXSPxA and TXSPxB) transmit simultaneously, each transmitting left and right I^2S channels. If both channels on a SPORT are set up to receive, the SPORT receive channels (RXSPxA and RXSPxB) receive simultaneously, each receiving left and right I^2S channels. Data is transmitted in MSB-first format.

If the MCEA or MCEB bits are set (=1) in the SPMCTLxy register, the SPEN_A and SPEN_B bits in the SPCTL register must be cleared (=0).

Multichannel operation and companding are not supported in I^2S mode. See "Multichannel Operation" on page 4-24.

Each SPORT transmit or receive channel has a channel enable, a DMA enable, and chaining enable bits in its SPCTLx Control register. The SPORTx_FS signal is used as the transmit and/or receive word select signal. DMA-driven or interrupt-driven data transfers can also be selected using bits in the SPCTLx register.

I^2S Mode Control Bits

Several bits in the SPCTLx Control register enable and configure I^2S mode operation:

- Operation mode, Master mode enable (OPMODE)
- Word length (SLEN)
- SPORT enable (SPEN_A and SPEN_B)

For more information, see "Serial Port Registers" on page A-19.

Setting the Internal Serial Clock and Frame Sync Rates

The serial clock rate (CLKDIV value) for internal clocks can be set using a bit field in the CLKDIV register. For details, see Figure 4-8 on page 4-63.

I²S Control Bits

Table 4-8 on page 4-63 shows that I²S mode is simply a subset of the Left-justified Sample Pair mode which can be invoked by setting OPMODE = 1, LAFS = 0, and FRFS = 0.

If FRFS = 1, the Tx/Rx is on the right channel first. For normal I²S operation (FRFS = 0), the Tx/Rx starts on the left channel first.

Several bits in the SPCTLx register Control register enable and configure I²S operation:

- Channel enable (SPEN_A or SPEN_B)
- Word length (SLEN)
- I²S channel transfer order (FRFS)
- Master mode enable (MSTR)
- DMA enable (SDEN_A and SDEN_B)
- DMA chaining enable (SCHEN_A and SCHEN_B)

Setting Word Length (SLEN)

SPORTs handle data words containing 8 to 32 bits in I²S Mode. Programs need to set the bit length for transmitting and receiving data words. For details, see "Word Length" on page 4-39.

The transmitter sends the MSB of the next word one clock cycle after the word select (TFS) signal changes.

In I^2S mode, load the FSDIV register with the same value as SLEN to transmit or receive words continuously. For example, for 8-bit data words (SLEN = 7), set FSDIV = 7.

Enabling SPORT Master Mode (MSTR)

The SPORTs transmit and receive channels can be configured for Master or Slave mode. In Master mode, the processor generates the word select and serial clock signals for the transmitter or receiver. In slave mode, an external source generates the word select and serial clock signals for the transmitter or receiver. When MSTR is cleared (=0), the processor uses an external word select and clock source. The SPORT transmitter or receiver is a slave. When MSTR is set (=1), the processor uses the processor's internal clock for word select and clock source. The SPORT transmitter or receiver is the master. For more information, see "Setting the Internal Serial Clock and Frame Sync Rates" on page 4-15.

Selecting Transmit and Receive Channel Order (FRFS)

In Master and Slave modes, it is possible to configure the I^2S channel to which each SPORT channel transmits or receives first. The left and right I^2S channels are time-duplexed data channels.

To select the channel order, set the FRFS bit (= 1) to transmit or receive on the left channel first, or clear the FRFS bit (= 0) to transmit or receive on the right channel first.

Selecting Frame Sync Options (DIFS)

When using both SPORT channels (SPORTx_DA and SPORTx_DB) as transmitters and MSTR = 1, SPTRAN = 1, and DIFS = 0, the processor generates a frame sync signal only when both transmit buffers contain data because both transmitters share the same SPORTx_CLK and SPORTx_FS. For continuous transmission, both transmit buffers must contain new data.

When using both SPORT channels (SPORTx_DA and SPORTx_DB) as receivers and MSTR = 1, SPTRAN = 1, and DIFS = 0, the processor generates a frame sync signal only when both receive buffers are not full because they share the same SPORTx_CLK and SPORTxFS.

When using both SPORT channels as transmitters and MSTR = 1, SPTRAN = 1 and DIFS = 1, the processor generates a frame sync signal at the frequency set by FSDIVx whether or not the transmit buffers contain new data. The DMA controller or the application is responsible for filling the transmit buffers with data.

When using both SPORT channels as receivers and MSTR = 1, SPTRAN = 1 and DIFS = 1, the processor generates a frame sync signal at the frequency set by FSDIV, irrespective of the receive buffer status. Bits 31–16 of the DIV register comprise the FSDIV bit field. For more information, see "SPORT Divisor Registers (DIVx)" on page A-35.

Enabling SPORT DMA (SDEN)

DMA can be enabled or disabled independently on any of the SPORT's transmit and receive channels. For more information, see "Moving Data Between SPORTS and Internal Memory" on page 4-66. Set SDEN_A or SDEN_B (=1) to enable DMA and set the channel in DMA-driven data transfer mode. Clear SDEN_A or SDEN_B (=0) to disable DMA and set the channel in an interrupt-driven data transfer mode.

Interrupt-Driven Data Transfer Mode

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, applications must check the transmit or receive data buffer status bits. For more information, see "Single Word Transfers" on page 4-74.

DMA-Driven Data Transfer Mode

Each transmitter and receiver has its own DMA registers. For details, see "Selecting Transmit and Receive Channel Order (FRFS)" on page 4-16 and "Moving Data Between SPORTS and Internal Memory" on page 4-66. The same DMA channel drives the left and right I²S 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 is interleaved in the DMA buffers.

Channel A and B on each SPORT share a common interrupt vector. The DMA controller generates an interrupt at the end of DMA transfer only.

Figure 4-4 shows the relationship between frame sync (word select), serial clock, and I²S data. Timing for word select is the same as for frame sync.

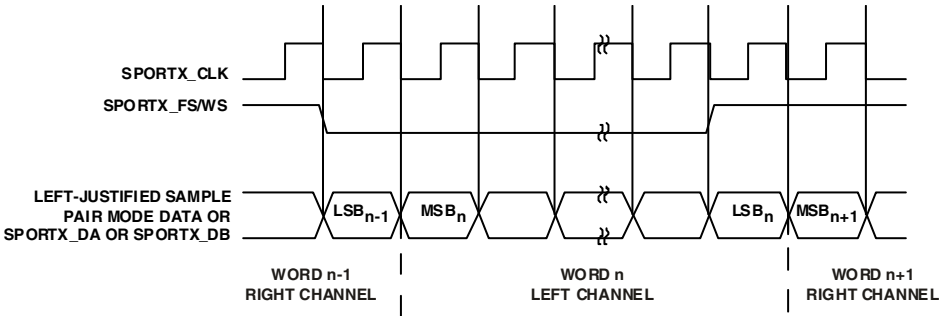


Figure 4-4. Word Select Timing in I²S Mode

The SPL bit applies to DSP Standard Serial and I²S modes only.

Multichannel Operation

The serial ports offer a multichannel mode of operation, which allows the SPORT to communicate in a Time Division Multiplexed (TDM) serial system. In multichannel communications, each data word of the serial bit stream occupies a separate channel. Each word belongs to the next consecutive channel. For example, a 24-word block of data contains one word for each of the 24 channels.

The 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.

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 six SPORTs are programmable for data direction in the standard mode of operation, their programmability is restricted for multichannel operations. The following points summarize these limitations:

1. The primary A channels of SPORT1, 3, and 5 are capable of expansion only, and the primary A channels of SPORT0, 2, and 4 are capable of compression only.
2. In Multichannel mode, SPORT0 and SPORT1 work in pairs; SPORT0 is the transmit channel, and SPORT1 is the receive channel. The same is true for SPORT2, SPORT3, SPORT4, and SPORT5.
3. Receive comparison is not supported.

In multichannel mode, SPORT0_CLK, SPORT2_CLK, and SPORT4_CLK are input signals that are internally connected to their corresponding SPORT1_CLK, SPORT3_CLK, and SPORT5_CLK signals.

Figure 4-5 shows an example of timing for a multichannel transfer with SPORT pairing. The transfer has the following characteristics:

- The transfer uses the TDM method where serial data is sent or received on different channels while sharing the same serial bus.
- The SPORT1_FS signals the start of a frame for each multichannel SPORT pairing.
- The SPORT0_FS is used as transmit data valid for external logic. This signal is active only during transmit channels.
- The transfer is received on channel 0 (word 0), and transmits on channels 1 and 2 (word 1 and 2).

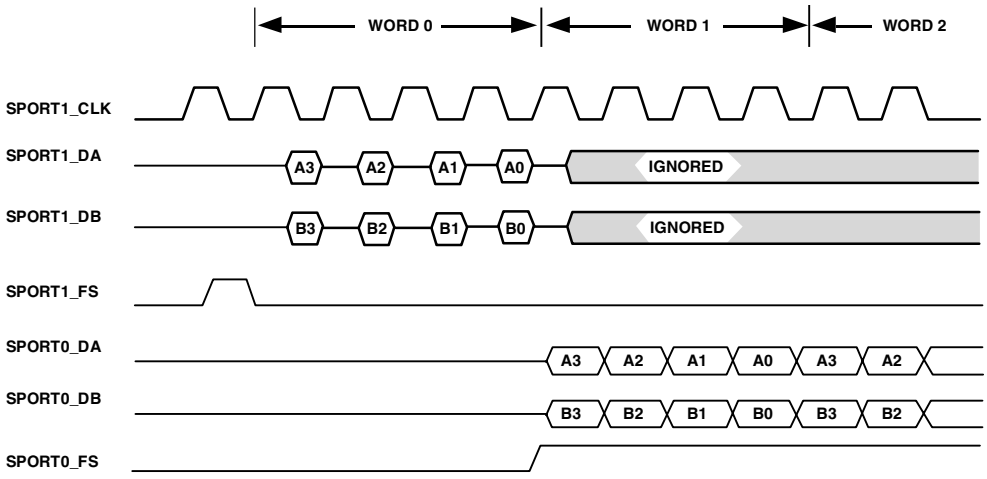


Figure 4-5. Multichannel Operation

Frame Syncs in Multichannel Mode

All receiving and transmitting devices in a multichannel system must have the same timing reference. The SPORT1_FS signal is used for this reference, indicating the start of a block (or frame) of multichannel data words. Pairs of SPORTs share the same frame sync signal for multichannel mode—SPORT1_FS for SPORT0/1, SPORT3_FS for SPORT2/3, and SPORT5_FS for SPORT4/5.

When multichannel mode is enabled on a SPORT0/1, SPORT2/3, or SPORT4/5 pair, both the transmitter and receiver use the SPORT1_FS, SPORT3_FS, or the SPORT5_FS signals respectively as a frame sync. This is true whether SPORT1_FS, SPORT3_FS, or the SPORT5_FS is generated internally or externally. This signal synchronizes the channels and restarts each multichannel sequence. The SPORT1_FS, SPORT3_FS, or SPORT5_FS signal initiates the beginning of the channel 0 data word.

SPORTs are paired when multichannel mode is selected; transmit/receive directions are fixed. SPORTS 0, 2, and 4 act as transmitters, and SPORTs 1, 3, and 5 act as receivers.

The SPORT0_FS, SPORT2_FS or SPORT4_FS is used as a transmit data valid signal, which is active during transmission of an enabled word. Because the serial port's SPORT0_DA/B, SPORT2_DA/B and SPORT4_DA/B signals are three-stated when the time slot is not active, the SPORT0_FS/SPORT2_FS/SPORT4_FS signal specifies if SPORT0_DA/B/SPORT2_DA/B/SPORT4_DA/B is being driven by the processor.

The SPORT0_FS signal is renamed TDV01. The SPORT2_FS signal is renamed TDV23 and the SPORT4_FS signal is renamed TDV45 in multichannel mode. These signals become outputs. Do not connect SPORT2_FS (TDV23) to SPORT0_FS, and SPORT4_FS (TDV45) to SPORT1_FS in multichannel mode. Bus contention between the transmit data valid and multichannel frame sync signals will result.

After the TXSPxA transmit buffer is loaded, transmission begins and the SPORT0_FS, SPORT2_FS/SPORT4_FS signal is generated. When serial port DMA is used, this may occur several cycles after the multichannel transmission is enabled. If a deterministic start time is required, pre-load the transmit buffer.

Active State Multichannel Receive Frame Sync Select

The LRFS bit in the SPCTL1, SPCTL3, and SPCTL5 registers selects the logic level of the multichannel received frame sync signals as active low (inverted) if set (=1) or active high if cleared (=0). Active high (=0) is the default.

Multichannel Mode Control Bits

Several bits in the SPCTLx Control register enable and configure multichannel mode operation:

- Operation mode (OPMODE)
- Word length (SLEN)
- SPORT transmit/receive enable (SDEN_A and SDEN_B)
- Master mode enable (MSTR)

If the MCEA or MCEB bits are set (=1) in the SPMCTLxy register, the SPEN_A and SPEN_B bits in the SPCTL register must be cleared (=0).

The SPCTLx Control registers contain several bits that enable and configure multichannel operations. Refer to Table 4-6 on page 4-51.

Multichannel mode is enabled by setting the MCEA or MCEB bit in the SPMCTL01, SPMCTL23 or SPMCTL45 Control register.

- When the MCEA or MCEB bits are set (=1), multichannel operation is enabled.
- When the MCEA or MCEB bits are cleared (=0), all multichannel operations are disabled.

Multichannel operation is activated three serial clock cycles 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.

Setting the MCEA or MCEB bits enables multichannel operation for both receive and transmit sides of the SPORT0/1, SPORT2/3 or SPORT4/5 pair. A transmitting SPORT0, 2, or 4 must be in multichannel mode if the receiving SPORT1, 3, or 5 is in multichannel mode.

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.

The 4-bit MFD field (bits 4-1) in the Multichannel Control registers (SPMCTL01, SPMCTL23, and SPMCTL45) 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.

Receive Multichannel Frame Sync Source

Bit 14 (IMFS) in the SPCTL1, SPCTL3 and SPCTL5 registers selects whether the serial port uses an internally generated frame sync (if set, =1) or frame sync from an external (if cleared, =0) source.

Active State Transmit Data Valid

Bit 16 (LTDV) in the SPCTL0, SPCTL2 and SPCTL4 registers selects the logic level of the transmit data valid signals (TDV01, TDV23, TDV45) as active low (inverted) if set (=1) or active high if cleared (=0). These signals are actually SPORT0_FS, SPORT2_FS and SPORT4_FS reconfigured as outputs during multichannel operation. They indicate which timeslots have valid data to transmit. Active high (0) is the default.

Multichannel Status Bits

Bit 29 (ROVF) in the SPCTL1, SPCTL3, SPCTL5 registers provides status information. This bit indicates if the channel has received new data if set (=1) or not if cleared (=0) while the RXSPxA buffer is full. New data overwrites existing data.

Bits 31-30 (RXS_A) in the SPCTL1, SPCTL3, SPCTL5 registers indicate the status of the channel's receive buffer contents as follows: 00 = buffer empty, 01 = reserved, 10 = buffer partially full, 11 = buffer full.

The SPCTL0, SPCTL2, SPCTL4 Bit 29 (TUVF_A). The Transmit Underflow Status (sticky, read-only) bit indicates (if set, =1) if the multichannel SPORTx_FS signal (from internal or external source) occurred while the TXS buffer was empty. The SPORTs transmit data whenever they detect a SPORTx_FS signal. If cleared (=0), no SPORTx_FS signal occurred.

This bit applies to Multichannel mode only when the SPORTs are configured as transmitters.

Bits 31-30 (TXS_A) in the SPCTL0, SPCTL2, SPCTL4 registers indicate the status of the serial port channel's transmit buffer as follows: 11= buffer full, 00=buffer empty, 10=buffer partially full. These bits apply to Multichannel mode only.

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 Table 4-2.

Table 4-2. Multichannel Selection Registers

Register Names	Function
MR1CS(0–3) MR3CS(0–3) MR5CS(0–3)	**Multichannel Receive Select** specifies the active receive channels (4x32-bit registers for 128 channels).
MT0CS(0–3) MT2CS(0–3) MT4CS(0–3)	**Multichannel Transmit Select** specifies the active transmit channels (4x32-bit registers for 128 channels).
MR1CCS(0–3) MR3CCS(0–3) MR5CCS(0–3)	**Multichannel Receive Compand Select** specifies which active receive channels (out of 128 channels) are companded.
MT0CCS(0–3) MT2CCS(0–3) MT4CCS(0–3)	**Multichannel Transmit Compand Select** specifies which active transmit channels (out of 128 channels) are companded.

Each of the four Multichannel Enable and Compand Select registers are 32 bits in length. These registers provide channel selection for 128 (32 bits x 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 MT0CS0 or MT2CS0 selects word 0, setting bit 12 selects word 12, and so on. Setting bit 0 in MT0CS1 or MT2CS1 selects word 32, setting bit 12 selects word 44, and so on.

Setting a particular bit to 1 in the MT0CS (0-3), MT2CS (0-3) or MT4CS (0-3) register causes SPORT0, 2, or 4 to transmit the word in that channel's position of the data stream. Clearing the bit in the register causes SPORT0's SPORT0_DA/B, SPORT2's SPORT2_DA/B or SPORT4's SPORT4_DA data transmit signal to three-state during the time slot of that channel.

Setting a particular bit to 1 in the MR1CS(0-3), MR3CS(0-3) or MR5CS(0-3) register causes the serial port to receive the word in that channel's position of the data stream. The received word is loaded into the receive buffer. Clearing the bit in the register causes the serial port to ignore the data.

Companding may be selected on a per-channel basis. Setting a bit to 1 in any of the multichannel registers 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. SPORT1, 3, and 5 expand selected incoming time slot data, while SPORT0, 2, and 4 compress selected outgoing time slot data.

SPORT Loopback

When the SPORT loopback bit, SPL bit 12 is set in the SPMCTL01, SPMCTL23, or SPMCTL45 control registers, the serial port is configured in an internal loopback connection as follows: SPORT0 and SPORT1 work as a pair for internal loopback, SPORT2 and SPORT3 work as pairs, and SPORT4 and SPORT5 work as pairs. The Loopback mode enables programs to test the serial ports internally and to debug applications.

When loopback is configured the:

- SPORTx_DA, SPORTx_DB, SPORTx_CLK and SPORTx_FS signals of SPORT0 and SPORT1 are internally connected (where x = 0 or 1)
- The SPORTy_DA, SPORTy_DB, SPORTy_CLK, and SPORTy_FS signals of SPORT2 and SPORT3 are internally connected (where y = 2 or 3)
- The SPORTz_DA, SPORTz_DB, SPORTz_CLK and SPORTz_FS signals of SPORT4 and SPORT5 are internally connected (where z = 4 or 5)

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. The processor ignores external activity on the SPORTx_CLK, SPORTx_FS A and B channel data signals when the SPORT is configured in Loopback mode. This prevents contention with the internal loopback data transfer.

Only transmit clock and transmit frame sync options may be used in loopback mode—programs must ensure that the serial port is set up correctly in the SPCTLx control registers. Multichannel mode is not allowed. Only Standard DSP Serial, Left-justified Sample Pair, and I^2S modes support internal loopback. In loopback, each SPORT can be configured as transmitter or receiver, and each one is capable of generating internal frame sync and clock.

Any of the three paired SPORTs can be set up to transmit or receive, depending on their SPTRAN bit configurations.

Clock Signal Options

Each serial port has a clock signal (SPORTx_CLK) for transmitting and receiving data on the two associated data signals. The clock signals are configured by the ICLK and CKRE bits of the SPCTLx Control registers. A single 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 serial clock can be independently generated internally or input from an external source. The ICLK bit of the SPCTLx Control registers determines the clock source.

When ICLK is set (=1), the clock signal is generated internally by the processor and the SPORTx_CLK signals are outputs. The clock frequency is determined by the value of the serial clock divisor (CLKDIV) in the DIVx registers.

When ICLK is cleared (=0), the clock signal is accepted as an input on the SPORTx_CLK 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 system clock. Refer to Table 4-8 on page 4-63.

Frame Sync Options

Framing signals indicate the beginning of each serial word transfer. A variety of framing options are available on the SPORTs. The SPORTx_FS signals are independent and are separately configured in the Control register.

Framed Versus Unframed Frame Syncs

The use of frame sync signals is optional in serial port communications. The FSR (transmit frame sync required) control bit determines whether frame sync signals are used. Active low or 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.

When DMA is enabled in a mode where frame syncs are not required, DMA requests may be held off by chaining or may not be serviced frequently enough to guarantee continuous unframed data flow.

Figure 4-6 illustrates framed serial transfers.

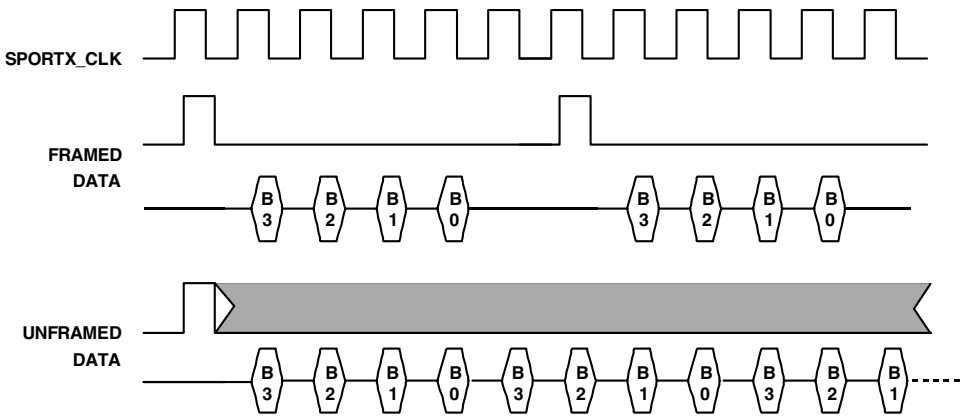


Figure 4-6. Framed Versus Unframed Data

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. Refer to Figure 4-8 on page 4-63.

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.

Active Low Versus Active High Frame Syncs

Frame sync signals may be active high or active low (for example, inverted). The LFS bit of the SPCTLx Control register determines the frame sync's logic level.

- When LFS is cleared (=0), the corresponding frame sync signal is active high.
- When LFS is set (=1), the corresponding frame sync signal is active low.

Active high frame syncs are the default. The LFS bit is initialized to zero after a processor reset.

Active low or active high frame syncs are selected using the LTDV and LRFS bits. These bits are located in the SPCTLx Control registers.

Sampling Edge for Data and Frame Syncs

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.

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 normal mode of operation. In this mode, the first bit of the transmit data word is available (and the first bit of the receive data word is latched) in the serial clock cycle after the frame sync is asserted. The frame sync is not checked again until the entire word has been transmitted (or received). In multichannel operation, this is the case when the frame delay is one.

If data transmission is continuous in early Framing mode (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 4-7 illustrates the two modes of frame signal timing.

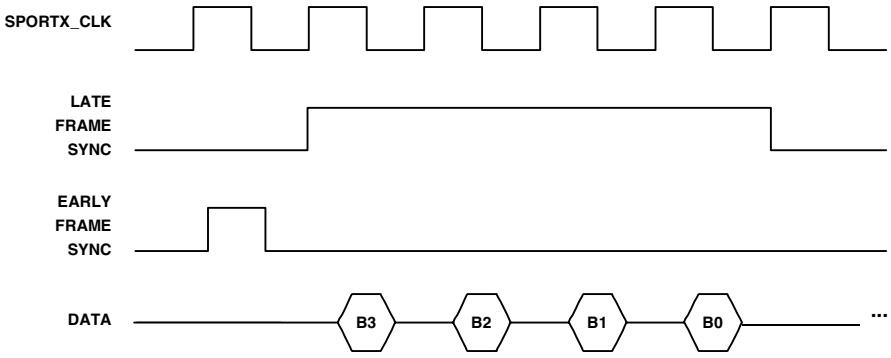


Figure 4-7. Normal Versus Alternate Framing

Data-Independent Frame Sync

When transmitting data out of the SPORT (SPTRAN = 1), the internally-generated frame sync signal normally is output-only when the transmit buffer has data ready to transmit. The Data-Independent Frame Sync (DIFS) mode allows the continuous generation of the SPORTx_FS signal, with or without new data in the register. The DIFS bit of the SPCTLx Control register configures this option.

When SPTRAN = 1, the DIFS bit selects whether the serial port uses a data-independent *transmit* frame sync (sync at selected interval, if set to 1) or a data-dependent transmit frame sync. When SPTRAN = 0, this bit selects whether the serial port uses a data-independent *receive* frame sync or a data-dependent receive 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.

When DIFS = 1 and SPTRAN = 1, the internally-generated transmit frame sync is output at its programmed interval regardless of whether new data is available in the transmit buffer. The processor generates the transmit SPORTx_FS signal at the frequency specified by the value loaded in the DIV register. If a frame sync occurs when the transmitter FIFO is empty, the MSB or LSB (depending on how the LSBF bit in SPCTL is set) of the previous word is transmitted. When DIFS = 1 and SPTRAN = 0, a receive SPORTx_FS signal is generated regardless of the receive data buffer status.

Depending on the SPORT operating mode, the Transmitter Underflow (TUVF_A or TUVF_B) bit is set if the transmit buffer does not have new data when a frame sync occurs; or a Receive Overflow bit (ROVF_A or ROVF_B) is set if the receive buffers are full and a new data word is received.

If the internally-generated frame sync is used and DIFS=0, a single write to the transmit data register is required to start the transfer.

Data Word Formats

The format of the data words transmitted over the serial ports is configured by the DTYPE, LSBF, SLEN, and PACK bits of the SPCTLx control registers.

Word Length

Serial ports can process word lengths of 3 to 32 bits for Serial and Multichannel modes and 8 to 32 bits for I^2S mode. Word length is configured using the 5-bit SLEN field in the SPCTLx Control registers. Refer to Table 4-1 on page 4-10 for further information.

The value of SLEN is:

SLEN = serial word length – 1

Do not set the SLEN value to 0 or 1. Words smaller than 32 bits are right-justified in the receive and transmit buffers, residing in the least significant (LSB) bit positions.

Although serial ports process word lengths of 3 to 32 bits, transmitting or receiving words smaller than 7 bits at one-quarter the full clock rate of the processor may cause incorrect operation when DMA chaining is enabled. Chaining disables 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.

Transmitting or receiving words smaller than five bits may cause incorrect operation when all the DMA channels are enabled with no DMA chaining.

Endian Format

Endian format determines whether serial words transmit MSB-first or LSB-first. Endian format is selected by the LSBF bit in the SPCTLx Control registers. When LSBF = 0, serial words transmit (or receive) MSB-first. When LSBF = 1, serial words transmit (or receive) LSB-first.

Data Packing and Unpacking

Received data words of 16 bits or less may be packed into 32-bit words, and 32-bit words being transmitted 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 in the Control register, two successive words received 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 when word packing or unpacking is being used.

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 processor internal memory, the 16-bit words can be read or written with short word space addresses.

Data Type

The DTYPE field of the SPCTLx Control registers specifies one of four data formats (for non-multichannel operation) shown in Table 4-3. This bit field is reserved for I^2S mode. In DSP Serial mode, if companding is selected for primary A channel, the secondary B channel performs a zero-fill.

In Multichannel mode, channel B looks at XDTYPE[0] only.
If DTYPE[0] = 1 sign-extend
If DTYPE[0] = 0 zero-fill

Table 4-3. DTYPE and Data Formatting (DSP Serial Mode)

DTYPE	Data Formatting
00	Right-justify, zero-fill unused MSBs
01	Right-justify, sign-extend into unused MSBs
10	Compand using μ-law (primary A channels only)
11	Compand using A-law (primary A channels only)

These formats are applied to serial data words loaded into the receive and transmit buffers. Transmit data words are not zero-filled or sign-extended, because only the significant bits are transmitted.

Table 4-4. DTYPE and Data Formatting (Multichannel)

DTYPE	Data Formatting
x0	Right-justify, zero-fill unused MSBs
x1	Right-justify, sign-extend into unused MSBs
0x	Compand using μ-law (primary A channels only)
1x	Compand using A-law (primary A channels only)

Linear transfers occur in the primary channel, if the channel is active and companding is not selected for that channel. Companded transfers occur if the channel is active and companding is selected for that channel. The Multichannel Compand Select registers, MTxCCSy and MRxCCSy, specify the transmit and receive channels that are companded.

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 channels that do not have companding selected. If this bit is not set, the word contains zeros in the MSB positions. Companding is not supported for B channel. For B channels, transmit or receive sign extension is selected by bit 0 of DTYPE in the SPCTLx register.

Companding

Companding (compressing/expanding) is the process of logarithmically encoding and decoding data to minimize the number of bits that must be sent. The serial ports support the two most widely used companding algorithms, A-law and μ-law, performed according to the CCITT G.711 specification. 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. SPORTs 0, 2, and 4 primary channels are capable of compression, while SPORTs 1, 3, and 5 primary channels are capable of expansion.
In Multichannel mode, when companding is enabled, the number of channels must be programmed via the NCH bit in the SPMCTLxy register before writing to the transmit FIFO. The MTxCSn and MTxCCsn registers should also be written before writing to transmit FIFO.

When companding is enabled, the data in the RXSPxA buffers is the right-justified, sign-extended expanded value of the eight received LSBs. A write to TXSPxA compresses the 32-bit value to eight LSBs (zero-filled to

the width of the transmit word) before it is transmitted. If the 32-bit value is greater than the 13-bit A-law or 14-bit μ-law maximum, it is automatically compressed to the maximum value.

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 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 compand data in place without transmitting:

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 one cycle. A NOP instruction can be used to cause this delay; if a NOP is not inserted, the processor core is paused for one cycle anyway. This allows the serial port companding hardware to reload the transmit buffer with the companded value.
5. Read the 8-bit companded value from the transmit buffer.

To expand data in place, use the same sequence of operations (above) 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 Control 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.

SPORT Control Registers and Data Buffers

The ADSP-2126x processor has six serial ports. Each SPORT has two data paths corresponding to channel A and channel B. These data buffers are `TXSPxA` and `RXSPxA` (primary) and `TXSPxB` and `RXSPxB` (secondary). Channel A and B in all six SPORTS operate synchronously to their respective `SPORTx_CLK` and `FSx` signals. Companding is supported only on primary A channels.

The registers used to control and configure the serial ports are part of the IOP register set. Each SPORT has its own set of 32-bit control registers and data buffers. The SPORT registers are described in Table 4-5.

The SPORT Control registers are programmed by writing to the appropriate address in memory. The symbolic names of the registers and individual control bits can be used in programs. The definitions for these symbols are contained in the file `def2126x.h` located in the `INCLUDE` directory of the ADSP-21xxx DSP Development Software. All control and status bits in the SPORT registers are active high unless otherwise noted.

Since the SPORT registers are memory-mapped, they cannot be written with data directly from memory. Instead, they must be written from (or read into) processor core registers, usually one of the general-purpose Universal registers (`R0-R15`) of the register file or one of the general-purpose Universal Status registers (`USTAT1-USTAT4`). The SPORT Control registers can also be written or read by external devices (for example, another processor or a host processor) to set up a serial port DMA operation.

Table 4-5 provides a complete list of the SPORT registers in IOP address order, showing the memory-mapped IOP address and a brief description of each register.

Table 4-5. SPORT Registers

IOP Address	Register	Reset	Description
0x400	SPCTL2	0x0000 0000	SPORT2 Serial Control Register
0x401	SPCTL3	0x0000 0000	SPORT3 Serial Control Register
0x402	DIV2	None	SPORT2 Divisor for Transmit/Receive SPORT2_CLK and SPORT2_FS
0x403	DIV3	None	SPORT3 Divisor for Transmit/Receive SPORT3_CLK and SPORT3_FS
0x404	SPMCTL23	None	SPORT 2/3 Multichannel Control Register
0x405	MT2CS0	None	SPORT2 Multichannel Transmit Select 0 (Channel 31-0)
0x406	MT2CS1	None	SPORT2 Multichannel Transmit Select 1 (Channel 63-32)
0x407	MT2CS2	None	SPORT2 Multichannel Transmit Select 2 (Channel 95–64)
0x408	MT2CS3	None	SPORT2 Multichannel Transmit Select 3 (Channel 127–96)
0x409	MR3CS0	None	SPORT3 Multichannel Receive Select 0 (Channel 31–0)
0x40A	MR3CS1	None	SPORT3 Multichannel Receive Select 1 (Channel 63–32)
0x40B	MR3CS2	None	SPORT3 Multichannel Receive Select 2 (Channel 95–64)
0x40C	MR3CS3	None	SPORT3 Multichannel Receive Select 3 (Channel 127–96)
0x40D	MT2CCS0	None	SPORT2 Multichannel Transmit Compand Select 0 (Channel 31–0)

Table 4-5. SPORT Registers (Cont'd)

IOP Address	Register	Reset	Description
0x40E	MT2CCS1	None	SPORT2 Multichannel Transmit Compand Select 1 (Channel 63–32)
0x40F	MT2CCS2	None	SPORT2 Multichannel Transmit Compand Select 2 (Channel 95–64)
0x410	MT2CCS3	None	SPORT2 Multichannel Transmit Compand Select 3 (Channel 127–96)
0x411	MR3CCS0	None	SPORT3 Multichannel Receive Compand Select 0 (Channel 31–0)
0x412	MR3CCS1	None	SPORT3 Multichannel Receive Compand Select 1 (Channel 63–32)
0x413	MR3CCS2	None	SPORT3 Multichannel Receive Compand Select 2 (Channel 95–64)
0x414	MR3CCS3	None	SPORT3 Multichannel Receive Compand Select 3 (Channel 127–96)
0x460	TXSP2A	None	SPORT2 Transmit Data Buffer; A channel data
0x461	RXSP2A	None	SPORT2 Receive Data Buffer; A channel data
0x462	TXSP2B	None	SPORT2 Transmit Data Buffer; B channel data
0x463	RXSP2B	None	SPORT2 Receive Data Buffer; B channel data
0x464	TXSP3A	None	SPORT3 Transmit Data Buffer; A channel data
0x465	RXSP3A	0x0000 0000	SPORT3 Receive Data Buffer; A channel data
0x466	TXSP3B	0x0000 0000	SPORT3 Transmit Data Buffer; B channel data
0x467	RXSP3B	0x0000 0000	SPORT3 Receive Data Buffer; B channel data
0x800	SPCTL4	0x0000 0000	SPORT4 Serial Control Register
0x801	SPCTL5	0x0000 0000	SPORT5 Serial Control Register
0x802	DIV4	0x0000 0000	SPORT4 Divisor for Transmit/Receive SPORT4_CLK and SPORT4_FS

Table 4-5. SPORT Registers (Cont'd)

IOP Address	Register	Reset	Description
0x803	DIV5	0x0000 0000	SPORT5 Divisor for Transmit/Receive SPORT4_CLK and SPORT5_FS
0x804	SPMCTL45	0x0000 0000	SPORT 4/5 Multichannel Control Register
0x805	MT4CS0	0x0000 0000	SPORT4 Multichannel Transmit Select 0 (Channel 31–0)
0x806	MT4CS1	0x0000 0000	SPORT4 Multichannel Transmit Select 1 (Channel 63–32)
0x807	MT4CS2	0x0000 0000	SPORT4 multichannel transmit select 2 (Channel 95–64)
0x808	MT4CS3	0x0000 0000	SPORT4 multichannel transmit select 3 (Channel 127–96)
0x809	MR5CS0	0x0000 0000	SPORT5 Multichannel Receive Select 0 (Channel 31–0)
0x80A	MR5CS1	0x0000 0000	SPORT5 Multichannel Receive Select 1 (Channel 63–32)
0x80B	MR5CS2	0x0000 0000	SPORT5 Multichannel Receive Select 2 (Channel 95–64)
0x80C	MR5CS3	0x0000 0000	SPORT5 Multichannel Receive Select 3 (Channel 127–96)
0x80D	MT4CCS0	0x0000 0000	SPORT4 Multichannel Transmit Compand Select 0 (Channel 31–0)
0x80E	MT4CCS1	0x0000 0000	SPORT4 Multichannel Transmit Compand Select 1 (Channel 63–32)
0x80F	MT4CCS2	0x0000 0000	SPORT4 Multichannel Transmit Compand Select 2 (Channel 95–64)
0x810	MT4CCS3	0x0000 0000	SPORT4 Multichannel Transmit Compand Select 3 (Channel 127–96)
0x811	MR5CCS0	0x0000 0000	SPORT5 Multichannel Receive Compand Select 0(Channel 31–0)

Table 4-5. SPORT Registers (Cont'd)

IOP Address	Register	Reset	Description
0x812	MR5CCS1	0x0000 0000	SPORT5 Multichannel Receive Compand Select 1 (Channel 63–32)
0x813	MR5CCS2	0x0000 0000	SPORT5 Multichannel Receive Compand Select 2 (Channel 95–64)
0x814	MR5CCS3	0x0000 0000	SPORT5 Multichannel Receive Compand Select 3 (Channel 127–96)
0x860	TXSP4A	0x0000 0000	SPORT4 Transmit Data Buffer; A channel data
0x861	RXSP4A	0x0000 0000	SPORT4 Receive Data Buffer; A channel data
0x862	TXSP4B	0x0000 0000	SPORT4 Transmit Data Buffer; B channel data
0x863	RXSP4B	0x0000 0000	SPORT4 Receive Data Buffer; B channel data
0x864	TXSP5A	0x0000 0000	SPORT5 Transmit Data Buffer; A channel data
0x865	RXSP5A	0x0000 0000	SPORT5 Receive Data Buffer; A channel data
0x866	TXSP5B	0x0000 0000	SPORT5 Transmit Data Buffer; B channel data
0x867	RXSP5B	0x0000 0000	SPORT5 Receive Data Buffer; B channel data
0xC00	SPCTL0	0x0000 0000	SPORT0 Serial Control Register
0xC01	SPCTL1	0x0000 0000	SPORT1 Serial Control Register
0xC02	DIV0	0x0000 0000	SPORT0 Divisor for Transmit/Receive SPORT0_CLK and SPORT0_FS
0xC03	DIV1	0x0000 0000	SPORT1 Divisor for Transmit/Receive SPORT1_CLK and SPORT1_FS
0xC04	SPMCTL01	0x0000 0000	SPORT 0/1 Multichannel Control Register
0xC05	MT0CS0	0x0000 0000	SPORT0 Multichannel Transmit Select 0 (Channels 31–0)
0xC06	MT0CS1	0x0000 0000	SPORT0 Multichannel Transmit Select 1 (Channels 63–32)
0xC07	MT0CS2	0x0000 0000	SPORT0 Multichannel Transmit Select 2 (Channels 95–64)

Table 4-5. SPORT Registers (Cont'd)

IOP Address	Register	Reset	Description
0xC08	MT0CS3	0x0000 0000	SPORT0 Multichannel Transmit Select 3 (Channels 127–96)
0xC09	MR1CS0	0x0000 0000	SPORT1 Multichannel Receive Select 0 (Channels 31–0)
0xC0A	MR0CS1	0x0000 0000	SPORT0 Multichannel Receive Select 1 (Channels 63–32)
0xC0B	MR0CS2	0x0000 0000	SPORT0 Multichannel Receive Select 2 (Channels 95–64)
0xC0C	MR1CS3	0x0000 0000	SPORT0 Multichannel Receive Select 3 (Channels 127–96)
0xC0D	MT0CCS0	0x0000 0000	SPORT0 Multichannel Transmit Compand Select 0 (Channels 31–0)
0xC0E	MT0CCS1	0x0000 0000	SPORT0 Multichannel Transmit Compand Select 1 (Channels 63–32)
0xC0F	MT0CCS2	0x0000 0000	SPORT0 multichannel transmit compand select 2 (Channels 95–64)
0xC10	MT0CCS3	0x0000 0000	SPORT0 Multichannel Transmit Compand Select 3 (Channels 127–96)
0xC11	MR1CCS0	0x0000 0000	SPORT1 Multichannel Receive Compand Select 0 (Channels 31–0)
0xC12	MR1CCS1	0x0000 0000	SPORT1 Multichannel Receive Compand Select 1 (Channels 63–32)
0xC13	MR1CCS2	0x0000 0000	SPORT1 Multichannel Receive Compand Select 2 (Channels 95–64)
0xC14	MR1CCS3	0x0000 0000	SPORT1 Multichannel Receive Compand select 3 (Channels 127–96)
0xC60	TXSP0A	0x0000 0000	SPORT0 Transmit Data Buffer; A channel data
0xC61	RXSP0A	0x0000 0000	SPORT0 Receive Data Buffer; A channel data
0xC62	TXSP0B	0x0000 0000	SPORT0 Transmit Data Buffer; B channel data

Table 4-5. SPORT Registers (Cont'd)

IOP Address	Register	Reset	Description
0xC63	RXSP0B	0x0000 0000	SPORT0 Receive Data Buffer; B channel data
0xC64	TXSP1A	0x0000 0000	SPORT1 Transmit Data Buffer; A channel data
0xC65	RXSP1A	0x0000 0000	SPORT1 Receive Data Buffer; A channel data
0xC66	TXSP1B	0x0000 0000	SPORT1 Transmit Data Buffer; B channel data
0xC67	RXSP1B	0x0000 0000	SPORT1 Receive Data Buffer; B channel data

Register Writes and Effect Latency

SPORT register writes are internally completed at the end of three (worst case) or two (best case) core clock cycles. The newly written value to the SPORT register can be read back on the next cycle. Reads of the SPORT registers take four core clock cycles.

After a write to a SPORT register, control and mode bit changes take effect in the second serial clock cycle. The serial ports are ready to start transmitting or receiving three serial clock cycles after they are enabled in the SPCTLx control register. No serial clocks are lost from this point on.

Serial Port Control Registers (SPCTLx)

The main control register for each serial port is the Serial Port Control register, SPCTLx. These registers are described in "SPORT Serial Control Registers (SPCTLx)" on page A-19. When changing operating modes, clear the Serial Port Control register before the new mode is written to the register.

There is one Global Control and Status register for each paired SPORT (SPORT0/1, SPORT 2/3 and SPORT 4/5) for multichannel operation. These are SPMCTL01, SPMCTL23, or SPMCTL45. These registers define the number of channels, provide the status of the current channel, enable

multichannel operation, and set the multichannel frame delay. These registers are described in "SPORT Multichannel Control Registers (SPMCTLxy)" on page A-28.

The SPCTLx registers control the operating modes of the serial ports for the I/O processor. Table 4-6 lists all the bits in the SPCTLx register.

Table 4-6. SPCTLx Control Bit Comparison in Four SPORT Operation Modes

Bit	Standard DSP Serial Mode	Left-justified and I²S Sample Pair Mode	Multichannel Mode	
			Transmit Control Bits (SPORT0, 2, and 4)	Receive Control Bits (SPORT1, 3, and 5)
0	SPEN_A	SPEN_A	Reserved	Reserved
1	DTYPE	Reserved	DTYPE	DTYPE
2	DTYPE	Reserved	DTYPE	DTYPE
3	LSBF	Reserved	LSBF	LSBF
4	SLEN0	SLEN0	SLEN0	SLEN0
5	SLEN1	SLEN1	SLEN1	SLEN1
6	SLEN2	SLEN2	SLEN2	SLEN2
7	SLEN3	SLEN3	SLEN3	SLEN3
8	SLEN4	SLEN4	SLEN4	SLEN4
9	PACK	PACK	PACK	PACK
10	ICLK	MSTR	Reserved	ICLK
11	OPMODE	OPMODE	OPMODE	OPMODE
12	CKRE	Reserved	CKRE	CKRE
13	FSR	Reserved	Reserved	Reserved
14	IFS	Reserved	Reserved	IMFS
15	DIFS	DIFS	Reserved	Reserved

Table 4-6. SPCTLx Control Bit Comparison in Four SPORT Operation Modes (Cont'd)

Bit	Standard DSP Serial Mode	Left-justified and I^2S Sample Pair Mode	Multichannel Mode	
			Transmit Control Bits (SPORT0, 2, and 4)	Receive Control Bits (SPORT1, 3, and 5)
16	LFS	FRFS	LTDV	LRFS
17	LAFS	LAFS	Reserved	Reserved
18	SDEN_A	SDEN_A	SDEN_A	SDEN_A
19	SCHEN_A	SCHEN_A	SCHEN_A	SCHEN_A
20	SDEN_B	SDEN_B	SDEN_B	SDEN_B
21	SCHEN_B	SCHEN_B	SCHEN_B	SCHEN_B
22	FS_BOTH	No effect	Reserved	Reserved
23	BHD	BHD	BHD	BHD
24	SPEN_B	SPEN_B	Reserved	Reserved
25	SPTRAN	SPTRAN	Reserved	Reserved
26	ROVF_B, or TUVF_B	ROVF_B, or TUVF_B	TUVF_B	ROVF_B
27	DXS_B	DXS_B	TXS_B	RXS_B
28	DXS_B	DXS_B	TXS_B	RXS_B
29	ROVF_A, or TUVF_A	ROVF_A, or TUVF_A	TUVF_A	ROVF_A
30	DXS_A	DXS_A	TXS_A	RXS_A
31	DXS_A	DXS_A	TXS_A	RXS_A

The following bits, listed in bit number order, control serial port modes and are part of the SPCTLx (transmit and receive) Control registers. Other bits in the SPCTLx registers set up DMA and I/O processor-related serial

port features. For information about configuring a specific operation mode, refer to Table 4-1 on page 4-10 and "Standard DSP Serial Mode" on page 4-11.

Serial Port Enable. SPCTLx bits 0 and 24 (SPEN_A and SPEN_B). This bit enables (if set, = 1) or disables (if cleared, = 0) the corresponding serial port channel A or B. Clearing this bit aborts any ongoing operation and clears the status bits. The SPORTS are ready to transmit or receive two serial clock cycles after enabling.

This description applies to I^2S and DSP Standard Serial modes only.

Data Type Select. SPCTLxx bits 2–1 (DTYPE). These bits select the companding and MSB data type formatting of serial words loaded into the transmit and receive buffers. This bit applies to DSP standard Serial and Multichannel modes only. The Transmit Shift register does not zero-fill or sign-extend transmit data words; this only takes place for the receive shift register.

For Standard mode, selection of Companding mode and MSB format are exclusive:

00 = Right-justify; fill unused MSBs with 0s
01 = Right-justify; sign-extend into unused MSBs
10 = Compand using μ_law, (primary channels only)
11 = Compand using A_law, (primary channels only)

For Multichannel mode, selection of companding mode and MSB format are independent:

x0 = Right-justify; fill unused MSBs with 0s
x1 = Right-justify; sign-extend into unused MSBs
0x = Compand using μ_law
1x = Compand using A_law

This description applies only to DSP Standard Serial mode and Multichannel modes only.

Serial Word Endian Select. SPCTLx Bit 3 (LSBF). This bit selects little endian words (LSB first, if set, = 1) or big endian words (MSB first, if cleared, = 0). This description applies to DSP Standard Serial And Multichannel modes only.

Serial Word Length Select. SPCTLx Bit 8–4 (SLENx). These bits select the word length in bits. Word sizes can be from 3 bits (SLEN = 2) to 32 bits (SLEN = 31). This bit applies to all operation modes.

Use this formula to calculate the value for SLEN:

SLEN = Actual serial word length – 1

In this case, the SLEN bit cannot equal 0 or 1, I^2S, Left-justified Sample Pair word length is limited to 8-32 bits, and DSP Standard mode word length varies from 3 to 32 bits.

16-bit to 32-bit Word Packing Enable. SPCTLx bit 9 (PACK). This bit enables (if set, = 1) or disables (if cleared, = 0) 16- to 32-bit word packing. This bit applies to all operation modes.

Internal Clock Select. SPCTLx bit 10 (ICLK). This bit selects the internal (if set, =1) or external (if cleared, =0) transmit or receive clock. This bit applies to DSP Standard Serial mode and SPORTs 1, 3 and 5 for multichannel modes.

Sport Operation Mode. SPCTLx bit 11 (OPMODE). This bit enables I^2S/Left-justified Sample Pair modes if set (= 1), or disables if cleared (= 0). This bit applies to all operation modes. See Table 4-1 on page 4-10 and "Standard DSP Serial Mode" on page 4-11.

Clock Rising Edge Select. SPCTLx bit 12 (CKRE). This bit selects whether the serial port uses the rising edge (if set, = 1) or falling edge (if cleared, = 0) of the clock signal for sampling data and the frame sync. This bit applies to DSP Standard Serial and Multichannel modes only.

Frame Sync Required Select. SPCTLx bits 13 (FSR). This bit selects whether the serial port requires (if set, = 1) or does not require (if cleared, = 0) a transfer frame sync. See "Frame Sync Options" on page 4-33 for more details. This bit applies to DSP Standard Serial mode only.

Internal Frame Sync Select. SPCTLx bit 14 (IFS). This bit selects whether the serial port uses an internally-generated frame sync (if set, = 1) or a frame sync from an external (if cleared, = 0) source. This bit is used for Standard DSP Serial mode only.

Low Active Frame Sync Select. SPCTLx bit 16 (LFS). This bit selects the logic level of the (transmit or receive) frame sync signals. This bit selects an active low frame sync (if set, = 1) or active high frame sync (if cleared, = 0). Active high (0) is the default. This bit applies to DSP Standard Serial mode only.

Late Transmit Frame Sync Select. SPCTLx bit 17 (LAFS). 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. See "Frame Sync Options" on page 4-33 for more details.

This bit applies to DSP Standard Serial mode. This bit is also used to select between I^2S and Left-justified Sample Pair modes. See Table 4-1 on page 4-10 and "Standard DSP Serial Mode" on page 4-11 for more information.

Serial Port DMA Enable. SPCTLx bits 18 and 20 (SDEN_A and SDEN_B). This bit enables (if set, = 1) or disables (if cleared, = 0) the serial port's channel DMA. Bits 18 and 20 apply to all operating modes.

Serial Port DMA Chaining Enable. SPCTLx bits 19 and 21 (SCHEN_A and SCHEN_B). These bits enable (if set, = 1) or disables (if cleared, = 0) serial port's channels A and B DMA chaining. Bits 19 and 21 apply to all operating modes.

Frame Sync Both Enable. SPCTLx bit 22 (FS_BOTH). This bit applies when the SPORTS channels A and B are configured to transmit/receive data. If set (= 1), this bit issues frame sync only when data is present in *both* transmit buffers, TXA and TXB. If cleared (= 0), a frame sync is issued if data is present in either transmit buffers. This bit applies to DSP Standard Serial mode only.

When a SPORT is configured as a receiver, if FS_BOTH is set (= 1), frame sync is issued only when both the Rx FIFOs (RXSPA and RXSPB) are not full.

This bit is not used for I^2S and Left-justified Sample Pair modes. If only channel A or channel B is selected, the frame sync behaves as if FS_BOTH is cleared (= 0). If both A and B channels are selected, the word select acts as if FS_BOTH is set (= 1).

Buffer Hang Disable. SPCTLx bit 23 (BHD). When cleared (= 0), this bit causes the processor core to hang when it attempts to write to a full buffer or read from an empty buffer. When set (= 1), this bit disables the core-hang. In this case, 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. This bit is used in all modes.

Data Direction Control. SPCTLx bit 25 (SPTRAN). This bit controls the data direction of the serial port channel A and B signals.

- 0 = SPORT is configured to receive on both channels A and B. In this configuration, 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 = SPORT is configured to transmit on both channels A and B. In this configuration, 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.

This bit applies to I²S, Left-justified Sample Pair, and DSP Standard Serial modes.

Reading from or writing to inactive buffers cause the core to hang indefinitely until the SPORT is cleared.

Data Buffer Error Status (sticky, read-only). SPCTLx bit 29 and 26 (ROVF, TUVF). These bits indicate whether the serial transmit operation has underflowed (if set, = 1 and SPTRAN = 1) or a receive operation has overflowed (if set, = 1 and SPTRAN = 0) in the TXSPxA/RXSPxA and TXSPxB/RXSPxB data buffers.

This description applies to I²S, Left-justified Sample Pair, and DSP Standard Serial modes. In multichannel modes, corresponding bits (TUVF, ROVF) are used for this function.

When the SPORT is configured as a transmitter, this bit provides transmit underflow status. As a transmitter, if FSR = 1, this bit indicates whether the SPORTx_FS signal (from an internal or external source) occurred while the DXS buffer was empty. If FSR = 0, ROVF or TUVF is set whenever the SPORT is required to transmit and the transmit buffer is empty. The SPORTs transmit data whenever they detect a SPORTx_FS signal.

- 0 = No SPORTx_FS signal occurred while TXSPxA/B buffer is empty.
- 1 = SPORTx_FS signal occurred while TXSPxA/B buffer is empty.

When the SPORT is configured as a receiver, these bits provide receive overflow status. As a receiver, it indicates when the channel has received new data while the RXS_A buffer is full. New data overwrites existing data.

- 0 = No new data while RXSPxA/B buffer is full.
- 1 = New data while RXSPxA/B buffer is full.

Transmit Underflow Status (sticky, read-only). SPCTL0, SPCTL2, and SPCTL4 bit 29 (TUVF_A). This bit indicates (if set, = 1) whether the multichannel SPORTx_FS signal (from an internal or external source) occurred

while the TXS buffer was empty. SPORTs transmit data whenever they detect a SPORTx_FS signal. If cleared (= 0), no SPORTx_FS signal occurs because the TXS buffer is empty.

The Transmit Underflow Status bit (TUVF_A/ROVF_A or TUVF_A and TUVF_B/ROVF_B or TUVF_B) is set when the SPORTx_FS signal occurs from either an external or internal source while the TXSPxA or TXSPxB buffer is empty. The internally-generated SPORTx_FS signal may be suppressed whenever TXSPxA or TXSPxB is empty by clearing the DIFS control bit when SPTRAN = 1.

When the DIFS bit is cleared (the default setting) the frame sync signal (SPORTx_FS) is dependent upon new data being present in the transmit buffer. The SPORTx_FS signal is only generated for new data. Setting DIFS to 1 selects data-independent frame syncs which causes the SPORTx_FS signal to be generated whether or not new data is present. With each SPORTx_FS signal, the SPORT transmits the contents of the transmit buffer. Serial port DMA typically keeps the transmit buffer full.

The DIFS bit applies to Multichannel mode only when the SPORTs are configured as transmitters.

Receive Overflow Status (read-only, sticky). SPCTL1, SPCTL3 and SPCTL5 Bit 29 (ROVF). This bit indicates if the channel has received new data if set (=1) or not if cleared (=0) while the RXS_A/B buffer is full. New data overwrites existing data.

This bit applies to Multichannel mode only.

Data Buffer Status Channel A (read-only). SPCTL1, SPCTL3 and SPCTL5 bits 31–30 (RXS_A). These bits indicate the status of the channel's receive buffer contents as follows: 00 = buffer empty, 01 = reserved, 10 = buffer partially full, 11 = buffer full.

DXS Data Buffer Status. SPCTLx Bits 31–30 (DXS_A) and bits 28-27 (DXS_B). These read-only bits indicate the status of the serial port's data buffer as follows: 11 = buffer full, 00 = buffer empty, 10 = buffer partially full, 01 = reserved.

The DXS_A or DXS_B Status bits indicate whether the TXSPxA/RXSPxA or TXSPxB/RXSPxB buffer is full (11), empty (00), or partially full (10). To test for space in TXSPxA/B or RXSPxA/B, test whether DXS_A (bit 30) is equal to zero for the A channel, or whether DXS_B (bit 27) is equal to zero for the B channel. To test for the presence of any data in TXSPxA/B or RXSPxA/B, test whether DXS_A (bit 31) is equal to one for the A channel, or whether DXS_B (bit 28) is equal to one for the B channel.

This description applies to I^2S, Left-justified Sample Pair, and DSP Standard Serial modes.

When the SPORT is configured as a transmitter, these bits reflect transmit buffer status for the TXSPxA and TXSPxB buffers. When the SPORT is configured as a receiver, these bits reflect receive buffer status for the RXSPxA and RXSPxB buffers.

Transmit Data Buffer Status (read-only). SPCTL0, SPCTL2 and SPCTL4 Bits 30 and 31(TXS_A). These bits indicate the status of the serial port channel's transmit buffer as follows: 11 = buffer full, 00 = buffer empty, 10 = buffer partially full.

Transmit and Receive Data Buffers

The transmit buffers (TXSP0A, TXSP0B, TXSP1A, TXSP1B, TXSP2A, TXSP2B, TXSP3A, TXSP3B, TXSP4A, TXSP4B, TXSP5A, and TXSP5B) are the 32-bit transmit data buffers for SPORT0, SPORT1, SPORT2, SPORT3, SPORT4, and SPORT5 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 receive buffers (RXSP0A, RXSP0B, RXSP1A, RXSP1B, RXSP2A, RXSP2B, RXSP3A, RXSP3B, RXSP4A, RXSP4B, RXSP5A, and RXSP5B) are the 32-bit receive data buffers for SPORT0, SPORT1, SPORT2, SPORT3, SPORT4, and SPORT5 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.

Word lengths of less than 32 bits are automatically right-justified in the receive and transmit buffers.

The transmit buffers act like a two-location FIFO 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 register is set.

In non-Multichannel modes (I^2S, Left-justified Sample Pair, and DSP Standard Serial modes), the ROVF_A or TUVF_A and ROVF_B, or TUVF_B Overflow/Underflow status bits are set when an overflow or underflow occurs. In multichannel mode, the ROVF_A or TUVF_A bits are redefined due to the fixed-directional functionality of the SPCTLx registers. When the SPCTL1, SPCTL3 and SPCTL5 registers are configured for Multichannel mode, the Receive Overflow bit ROVF_A indicates when the A channel has received new data while the RXS_A buffer is full. Similarly, when the SPCTL0, SPCTL2 and SPCTL4 registers are configured for Multichannel

mode, the transmit overflow bit (TUVF_A) indicates that a new frame sync signal (SPORT0_FS/SPORT2_FS/SPORT4_FS) was generated while the TXSPxA buffer was empty.

The ROVF_A or TUVF_A (bit 29) Overflow/Underflow status bit in the SPCTLx register becomes fixed in Multichannel mode only as either the ROVF Overflow Status bit (SPORTs 1, 3, and 5) or TUVF_A Underflow Status bit (SPORTs 0, 2, and 4).

When the SPORT is configured as a transmitter (SPTRAN =1), and a transmit frame sync occurs and no new data has been loaded into the transmit buffer, a Transmit Underflow status bit is set in the Serial Port Control register. The TUVF_A/ROVF_A or TUVF_A status bit is sticky and is only cleared by disabling the serial port.

When the SPORT is configured as a receiver (SPTRAN = 0), the receive buffers are activated. The receive buffers act like a three-location FIFO because they have two data registers plus an input shift register. 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 ROVF_A/ROVF_A or TUVF_A status bit is sticky and is cleared only by disabling the serial port.

An interrupt is generated when the receive buffer has been loaded with a received word (for example, the receive buffer is not empty). This interrupt is masked if serial port DMA is enabled or if the corresponding bit in the LIRPTL register is set.

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.

To support debugging buffer transfers, the processor has 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. For more information, see the BHD bit description on on page 4-56.

The status bits in SPCTLx are updated during reads and writes from the core processor even when the serial port is disabled. Disable the serial port when writing to the receive buffer or reading from the transmit buffer.

When programming the serial port channel (A or B) as a transmitter, 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 the corresponding RXSPxA and RXSPxB is 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.

Clock and Frame Sync Frequencies (DIV)

The DIVx registers contain divisor values that determine frequencies for internally-generated clocks and frame syncs. The DIVx registers are described in Appendix A in "SPORT Divisor Registers (DIVx)" on page A-35.

The CLKDIV bit field specifies how many times the processor's internal clock (CCLK) 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, allowing a wide range of serial clock rates.

Use the following equation to calculate the serial clock frequency:

$$f_{SPORTx\_CLK} = \frac{f_{CCLK}}{4(CLKDIV+1)}$$

The maximum serial clock frequency is equal to one-quarter the processor's internal clock (`CCLK`) frequency, which occurs when `CLKDIV` is set to zero. Use the following equation to determine the value of `CLKDIV`, given the `CCLK` frequency and desired serial clock frequency:

$$CLKDIV = \frac{f_{CCLK} - 1}{4(f_{SPORTx\_CLK})}$$

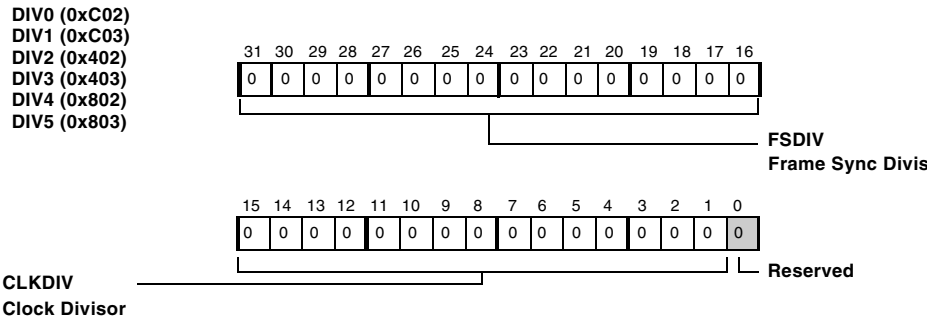


Figure 4-8. DIVx Register

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:

```
# 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:

$$FSDIV = \frac{f_{SPORTx\_CLK}}{f_{SPORTx\_FS}} - 1$$

The frame sync is continuously active when FSDIV = 0. The value of FSDIV should not be less than the serial word length minus one (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. 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.

Exercise caution when operating with externally-generated transmit clocks near the frequency of one-quarter 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 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 product specific data sheet for exact timing specifications.

SPORT Reset

There are two ways to reset the serial ports, via software or hardware. Each method has a different effect on the serial port.

A software reset of the SPEN_A or SPEN_B Enable bit disables the serial port(s) and aborts any ongoing operations. Status bits are also cleared. The serial ports are ready to start transmitting or receiving data two serial clock cycles after they are enabled in the SPCTLx Control register. No serial clocks are lost from this point on.

A hardware reset ($\overline{\text{RESET}}$) disables the entire processor, including the serial ports, by clearing the SPCTLx control register. Any ongoing operations are aborted.

SPORT Interrupts

Each serial port has an interrupt associated with it. For each SPORT, both the A and B channel transmit and receive data buffers share the same interrupt vector. The 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. They can also be used to perform single word transfers. Refer to "Single Word Transfers" on page 4-74. The priority of the serial port interrupts is shown in Table 4-7.

The interrupt names are defined in the def2126x.h file supplied with the ADSP-21xxx DSP Development Software.

Table 4-7. Priority of the Serial Port Interrupts

Interrupt Name	Interrupt
SPR1I	SPORT1 DMA Channels (Highest Priority)
SPR3I	SPORT3 DMA Channels
SPR5I	SPORT5 DMA Channels

Table 4-7. Priority of the Serial Port Interrupts (Cont'd)

Interrupt Name	Interrupt
SPR0I	SPORT0 DMA Channels
SPR2I	SPORT2 DMA Channels
SPR4I	SPORT4 DMA Channels

SPORT interrupts occur on the second system clock (CLKIN) after the last bit of the serial word is latched in or driven out.

Moving Data Between SPORTS and Internal Memory

Transmit and receive data can be transferred between the serial ports and on-chip memory with single word transfers or with DMA block transfers. Both methods are interrupt-driven, and use the same internally-generated interrupts.

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.

DMA Block Transfers

The processor's on-chip DMA controller allows automatic DMA transfers between internal memory and each of the two channels of each serial port. Each SPORT has two channels for transferring data, and each can be con-

figured to receive or to transmit. There are twelve DMA channels for serial port operations. The serial port DMA channels are numbered as shown in Table 4-8.

Table 4-8. Serial Port DMA Channels

Channel	Data Buffer	Description	Priority
0	RXSP1A/TXSP1A	SPORT1 A data	Highest
1	RXSP1B/TXSP1B	SPORT1 B data	
2	RXSP0A/TXSP0A	SPORT0 A data	
3	RXSP0B/TXSP0B	SPORT0 B data	
4	RXSP3A/TXSP3A	SPORT3 A data	
5	RXSP3B/TXSP3B	SPORT3 B data	
6	RXSP2A/TXSP2A	SPORT2 A data	
7	RXSP2B/TXSP2B	SPORT2 B data	
8	RXSP5A/TXSP5A	SPORT5 A data	
9	RXSP5B/TXSP5B	SPORT5 B data	
10	RXSP4A/TXSP4A	SPORT4 A data	
11	RXSP4B/TXSP4B	SPORT4 B data	Lowest

Data-direction programmability is supported in Standard DSP Standard Serial, Left-justified Sample Pair, and I^2S modes. The value of the SPTRAN bit in SPCTLx (0 = RX, 1 = TX) determines whether the receive or transmit register for the SPORT becomes active.

The SPORT DMA channels are assigned higher priority than all other DMA channels (for example, the SPI port and the parallel 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.

Although the DMA transfers are performed with 32-bit words, serial ports can handle word sizes from 3 to 32 bits, with 8 to 32 bits for I^2S mode. If serial words are 16 bits or smaller, they can be packed into 32-bit words for each DMA transfer. DMA transfers are configured using the PACK bit in the SPCTLx Control registers. When serial port data packing is enabled (PACK = 1), the transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.

The following sections present an overview of serial port DMA operations; additional details are covered in the Memory chapter in the *ADSP-2126x SHARC DSP Core Manual*.

- For information on SPORT DMA Channel Setup, see "Selecting Transmit and Receive Channel Order (FRFS)" on page 4-16.
- For information on SPORT DMA Parameter Registers, see "Selecting Transmit and Receive Channel Order (FRFS)" on page 4-16.
- For information on SPORT DMA Chaining, see "SPORT DMA Chaining" on page 4-73.

Setting Up DMA on SPORT Channels

Each SPORT DMA channel has an Enable bit (SDEN_A and SDEN_B) in its SPCTLx Control 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 "Single Word Transfers" on page 4-74.

Each channel also has a DMA Chaining Enable bit (SCHEN_A and SCHEN_B) in its SPCTLx control register.

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 4-9.

Table 4-9. SPORT DMA Parameter Registers

Register (Y = A or B, and x = 0 – 5)	Width	Description
IISPxy	19 bits	DMA channel; x index; start address for data buffer
IMSPxy	16 bits	DMA channel; x modify; address increment
CSPxy	16 bits	DMA channel; x count; number of words to transmit
CPSPxy	20 bits	DMA channel; x chain pointer; address containing the next set of data buffer parameters

Load the `II`, `IM`, and `C` registers with a starting address for the buffer, an address modifier, and a word count, respectively. These registers can be written from the core processor or from an external processor.

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 memory. Likewise, when the serial port is ready to transmit data, the DMA controller automatically transfers a word from internal memory to the transmit buffer. The controller continues these transfers until the entire data buffer is received or transmitted.

When the count register of an active DMA channel reaches zero (0), the SPORT generates the corresponding interrupt.

SPORT DMA Parameter Registers

A DMA channel consists of a set of parameter registers that implements a data buffer in internal memory and the hardware the serial port uses to request DMA service. The parameter registers for each SPORT DMA channel and their addresses are shown in Table 4-10 below. These registers are part of the processor's memory-mapped IOP register set.

The DMA channels operate similarly to the processor's Data Address Generators (DAGs). Each channel has an Index register (IISPxy) and a Modify register (IMSPxy) for setting up a data buffer in internal memory. It is necessary to initialize the Index register with the starting address of the data buffer. After it transfers each serial I/O word to (or from) the SPORT, the DMA controller adds the modify value to the Index register to generate the address for the next DMA transfer. The modify value in the IM register is a signed integer, which provides capability for both incrementing and decrementing the buffer pointer.

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 disables the DMA channel.

Each SPORT DMA channel also has a Chain Pointer register (CPSPxy). The CPSPxy register functions are used in chained DMA operations. For more information on SPORT DMA chaining registers, see Table 4-9 on page 4-69.

Table 4-10. SPORT DMA Parameter Registers Addresses

Register	Address	DMA Channel	SPORT Buffer
IISP0A	0xc40	0	RXSP0A or TXSP0A
IMSP0A	0xc41	0	RXSP0A or TXSP0A
CSP0A	0xc42	0	RXSP0A or TXSP0A
CPSP0A	0xc43	0	RXSP0A or TXSP0A
IISP0B	0xc44	1	RXSP0B or TXSP0B
IMSP0B	0xc45	1	RXSP0B or TXSP0B
CSP0B	0xc46	1	RXSP0B or TXSP0B
CPSP0B	0xc47	1	RXSP0B or TXSP0B
IISP1A	0xc48	2	RXSP1A or TXSP1A

Table 4-10. SPORT DMA Parameter Registers Addresses (Cont'd)

Register	Address	DMA Channel	SPORT Buffer
IMSP1A	0xc49	2	RXSP1A or TXSP1A
CSP1A	0xc4A	2	RXSP1A or TXSP1A
CPSP1A	0xc4B	2	RXSP1A or TXSP1A
IISP1B	0xc4C	3	RXSP1B or TXSP1B
IMSP1B	0xc4D	3	RXSP1B or TXSP1B
CSP1B	0xc4E	3	RXSP1B or TXSP1B
CPSP1B	0xc4F	3	RXSP1B or TXSP1B
Reserved			
IISP2A	0x440	4	RXSP2A or TXSP2A
IMSP2A	0x441	4	RXSP2A or TXSP2A
CSP2A	0x442	4	RXSP2A or TXSP2A
CPSP2A	0x443	4	RXSP2A or TXSP2A
IISP2B	0x444	5	RXSP2B or TXSP2B
IMSP2B	0x445	5	RXSP2B or TXSP2B
CSP2B	0x446	5	RXSP2B or TXSP2B
CPSP2B	0x447	5	RXSP2B or TXSP2B
IISP3A	0x448	6	RXSP3A or TXSP3A
IMSP3A	0x449	6	RXSP3A or TXSP3A
CSP3A	0x44A	6	RXSP3A or TXSP3A
CPSP3A	0x44B	6	RXSP3A or TXSP3A
IISP3B	0x44C	7	RXSP3B or TXSP3B
IMSP3B	0x44D	7	RXSP3B or TXSP3B
CSP3B	0x44E	7	RXSP3B or TXSP3B
CPSP3B	0x44F	7	RXSP3B or TXSP3B

Table 4-10. SPORT DMA Parameter Registers Addresses (Cont'd)

Register	Address	DMA Channel	SPORT Buffer
Reserved			
IISP4A	0x840	8	RXSP4A or TXSP4A
IMSP4A	0x841	8	RXSP4A or TXSP4A
CSP4A	0x842	8	RXSP4A or TXSP4A
CPSP4A	0x843	8	RXSP4A or TXSP4A
IISP4B	0x844	9	RXSP4B or TXSP4B
IMSP4B	0x845	9	RXSP4B or TXSP4B
CSP4B	0x846	9	RXSP4B or TXSP4B
CPSP4B	0x847	9	RXSP4B or TXSP4B
IISP5A	0x848	10	RXSP5A or TXSP5A
IMSP5A	0x849	10	RXSP5A or TXSP5A
CSP5A	0x84A	10	RXSP5A or TXSP5A
CPSP5A	0x84B	10	RXSP5A or TXSP5A
IISP5B	0x84C	11	RXSP5B or TXSP5B
IMSP5B	0x84D	11	RXSP5B or TXSP5B
CSP5B	0x84E	11	RXSP5B or TXSP5B
CPSP5B	0x84F	11	RXSP5B or TXSP5B
Reserved (0x850 to 0x85F)			

When programming a serial port channel (either A or B) as a transmitter, only the corresponding TXSPxA and TXSPxB SPORT buffer becomes active, while the receive buffers (RXSPxA and RXSPxB) remain inactive. Similarly, when the SPORT channel A and B is programmed as a receiver, only the corresponding RXSP0A and RXSP0B SPORT buffer is activated.

When performing core-driven transfers, write to the buffer designated by the SPTRAN bit setting in the SPCTLx register. 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. If the inactive SPORT data buffers are read or written to by core while the port is being enabled, the core will hang. For example, if a SPORT is programmed to be a transmitter, while at the same time the core reads from the receive buffer of the same SPORT, the core hangs just as it would if it were reading an empty buffer that is currently active. This locks up the core until the SPORT is reset.

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 DSP operates on the inactive transmit or receive buffers while the SPORT is enabled, it can cause unpredictable results.

SPORT DMA Chaining

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 memory. The DMA controller automatically downloads these buffer parameters to set up the next DMA sequence. For more information on SPORT DMA chaining, see "Setting Up DMA Parameter Registers" on page 2-20.

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.

Single Word 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. Note that both channel A and B buffers share the same interrupt vector. Single word interrupts can be used to implement interrupt-driven I/O on the serial ports.

To avoid hanging the processor core—check the buffer's full/empty status when the core's program reads a word from a serial port's receive buffer or writes a word to its transmit buffer. This condition can also happen to an external device, for example a host processor, when it is reading or writing a serial port 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 (or external device) to hang, while it waits for the status to change.

To support debugging buffer transfers, the processor has 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. For more information, see the BHD bit discussion on on page 4-56.

Multiple interrupts can occur if both SPORTs transmit or receive data in the same cycle. Any interrupt can be masked in the IMASK register; if the interrupt is later enabled in the LIRPTL register, the corresponding interrupt latch bit in the IRPTL or LIRPTL registers must be cleared in case the interrupt has occurred in the same time period.

When serial port data packing is enabled (PACK=1 in the SPCTLx Control registers), the transmit and receive interrupts are generated for 32-bit packed words, not for each 16-bit word.

SPORT Programming Examples

The third listing, Listing 4-1, transmits a buffer of data from SPORT1 to SPORT0 using DMA chaining and the internal loopback feature of the serial port. In this example, SPORT5 drives the clock and frame sync, and the two TCBs for each SPORT are set up to ping-pong back and forth to continually send and receive data.

The second listing, Listing 4-2, transmits a buffer of data from SPORT5 to SPORT4 using DMA and the internal loopback feature of the serial port. In this example, SPORT5 drives the clock and frame sync, and the buffer will be transferred only one time.

This section provides three programming examples written for the ADSP-21262 processor. The first listing, Listing 4-3, transmits a buffer of data from SPORT2 to SPORT3 using direct core reads and writes and the internal loopback feature of the serial port. In this example, SPORT2 drives the clock and frame sync, and the buffer is transferred only one time.

Listing 4-1. SPORT Transmit Using DMA Chaining

```
/* SPORT DMA Parameter Registers */
#define CPSP0A  0xC43
#define CPSP1A  0xC4B

/* SPORT Control Registers */
#define DIV0    0xC02
#define DIV1    0xC03
#define SPCTL0  0xC00
#define SPCTL1  0xC01
#define SPMCTL01 0xC04

/* SPMCTL Bits */
#define SPL     0x00001000
```

```
/* SPCTL Bits */
#define SPEN_A  0x00000001
#define SDEN_A  0x00040000
#define SCHEN_A 0x00080000
#define SLEN32  0x000001F0
#define SPTRAN  0x02000000
#define IFS     0x00004000
#define FSR     0x00002000
#define ICLK    0x00000400

/* Default Buffer Length */
#define BUFSIZE 10

.SECTION/DM seg_dmda;
/* TX Buffers */
.var tx_buf1a[BUFSIZE] = 0x11111111,
                         0x22222222,
                         0x33333333,
                         0x44444444,
                         0x55555555,
                         0x66666666,
                         0x77777777,
                         0x88888888,
                         0x99999999,
                         0xAAAAAAAA;

.var tx_buf1b[BUFSIZE] = 0x12345678,
                         0x23456789,
                         0x3456789A,
                         0x456789AB,
                         0x56789ABC,
                         0x6789ABCD,
                         0x789ABCDE,
                         0x89ABCDEF,
```

```
                        0x9ABCDEF0,
                        0xABCDEF01;
/* RX Buffers */
.var rx_buf0a[BUFSIZE];
.var rx_buf0b[BUFSIZE];

/* TX Transfer Control Blocks */
.var tx_tcb1[4] = 0,BUFSIZE,1,tx_buf1a;
.var tx_tcb2[4] = 0,BUFSIZE,1,tx_buf1b;

/* RX Transfer Control Blocks */
.var rx_tcb1[4] = 0,BUFSIZE,1,rx_buf0a;
.var rx_tcb2[4] = 0,BUFSIZE,1,rx_buf0b;

/* Main code section */
.global _main;
.SECTION/PM seg_pmco;
_main:

/* SPORT loopback: use SPORT0 as RX and SPORT1 as TX.
For no loopback (TDM mode), program MTaCSb [a=0,2,4 & b=0,1,2,3]
and MRcCSd [a=1,3,5 & b=0,1,2,3], */

/* initially clear SPORT control register */
r0 = 0x00000000;
dm(SPCTL0) = r0;
dm(SPCTL1) = r0;
dm(SPMCTL01) = r0;

SPORT_DMA_setup:
/* set internal loopback bit for SPORT0 & SPORT1 */
bit set ustat3 SPL;
dm(SPMCTL01) = ustat3;
```

```
/* Configure SPORT1 as a transmitter */
/* internally generating clock and frame sync */
/* CLKDIV = [fCCLK(200 MHz)/4xFSCLK(20 MHz)]-1 = 0x004 */
/* FSDIV = [FSCLK(20 MHz)/TFS(.625 MHz)]-1 = 31 = 0x001F */
R0 = 0x001F0004;    dm(DIV1) = R0;
ustat4 = SPEN_A|    /* Enable Channel A */
         SLEN32|    /* 32-bit word length */
         FSR|       /* Frame Sync Required */
         SPTRAN|    /* Transmit on enabled channels */
         SDEN_A|    /* Enable Channel A DMA */
         SCHEN_A|   /* Enable Channel A DMA Chaining */
         IFS|       /* Internally-generated Frame Sync */
         ICLK;      /* Internally-generated Clock */
dm(SPCTL1) = ustat4;

/* Configure SPORT0 as a receiver */
/* externally generating clock and frame sync */
r0 = 0x0;     dm(DIV0) = R0;
ustat3 = SPEN_A|    /* Enable Channel A */
         SLEN32|    /* 32-bit word length */
         FSR|       /* Frame Sync Required */
         SDEN_A|    /* Enable Channel A DMA */
         SCHEN_A;   /* Enable Channel A DMA Chaining */
dm(SPCTL0) = ustat3;

/* Next TCB location for tx_tcb2 is tx_tcb1 */
/* Mask the first 19 bits of the TCB location */
r0 = (tx_tcb1 + 3) & 0x7FFFF;
dm(tx_tcb2) = r0;

/* Next TCB location for rx_tcb2 is rx_tcb1 */
/* Mask the first 19 bits of the TCB location */
r0 = (rx_tcb1 + 3) & 0x7FFFF;
dm(rx_tcb2) = r0;
```

```
/* Next TCB location for rx_tcb1 is rx_tcb2 */
/* Mask the first 19 bits of the TCB location */
r0 = (rx_tcb2 + 3) & 0x7FFFF;
dm(rx_tcb1) = r0;
/* Initialize SPORT DMA transfer by writing to the CP reg */
dm(CPSP0A) = r0;

/* Next TCB location for tx_tcb1 is tx_tcb2 */
/* Mask the first 19 bits of the TCB location */
r0 = (tx_tcb2 + 3) & 0x7FFFF;
dm(tx_tcb1) = r0;
/* Initialize SPORT DMA transfer by writing to the CP reg */
dm(CPSP1A) = r0;

_main.end:  jump (pc,0);
```

Listing 4-2. SPORT Transmit Using Direct Core Access

```
/* SPORT Control Registers */
#define TXSP2A  0x460
#define RXSP3A  0x465
#define DIV2    0x402
#define DIV3    0x403
#define SPCTL2  0x400
#define SPCTL3  0x401
#define SPMCTL23 0x404

/* SPMCTL Bits */
#define SPL     0x00001000

/* SPCTL Bits */
#define SPEN_A  0x00000001
#define SDEN_A  0x00040000
```

```
#define SLEN32  0x000001F0
#define SPTRAN  0x02000000
#define IFS     0x00004000
#define FSR     0x00002000
#define ICLK    0x00000400

/* Default Buffer Length */
#define BUFSIZE 10

.SECTION/DM seg_dmda;
/* Transmit Buffer */
.var tx_buf2a[BUFSIZE] = 0x11111111,
                         0x22222222,
                         0x33333333,
                         0x44444444,
                         0x55555555,
                         0x66666666,
                         0x77777777,
                         0x88888888,
                         0x99999999,
                         0xAAAAAAAA;

/* Receive Buffer */
.var rx_buf3a[BUFSIZE];

/* Main code section */
.global _main;
.SECTION/PM seg_pmco;
_main:
bit set mode1 CBUFEN;    /* enable circular buffers */

/*  SPORT Loopback: Use SPORT2 as RX & SPORT3 as TX.
    For no loopback (TDM mode), program MTaCSb
    [a=0,2,4 & b=0,1,2,3] and MRcCSd [a=1,3,5 & b=0,1,2,3], /*
```

```
/* Initially clear SPORT control registers */
r0 = 0x00000000;
dm(SPCTL2) = r0;
dm(SPCTL3) = r0;
dm(SPMCTL23) = r0;

/* Set up DAG registers */
i4 = tx_buf2a;
m4 = 1;
i12 = rx_buf3a;
m12 = 1;

SPORT_DMA_setup:
/* set internal loopback bit for SPORT2 & SPORT3 */
bit set ustat3 SPL;
dm(SPMCTL23) = ustat3;

/* Configure SPORT2 as a transmitter */
/* internally generating clock and frame sync */
/* CLKDIV = [fCCLK(200MHz)/4 x FSCLK(20MHz)] - 1 = 0x004 */
/* FSDIV = [FSCLK(20 MHz)/TFS(.625 MHz)] - 1 = 31 = 0x001F */
R0 = 0x001F0004;    dm(DIV2) = R0;
ustat4 = SPEN_A|    /* Enable Channel A */
         SLEN32|    /* 32-bit word length */
         FSR|       /* Frame Sync Required */
         SPTRAN|    /* Transmit on enabled channels */
         IFS|       /* Internally Generated Frame Sync */
         ICLK;      /* Internally Generated Clock */
dm(SPCTL2) = ustat4;

/* Configure SPORT3 as a receiver */
/* externally generating clock and frame sync */
r0 = 0x0;     dm(DIV3) = R0;
ustat3 = SPEN_A|    /* Enable Channel A */
```

```
          SLEN32|    /* 32-bit word length */
          FSR;       /* Frame Sync Required */
dm(SPCTL3) = ustat3;

/* Set up loop to transmit and receive data */
lcntr = LENGTH(tx_buf2a), do (pc,4) until lce;
/* Retrieve data using DAG1 and send TX via SPORT2 */
r0 = dm(i4,m4);
dm(TXSP2A) = r0;
/* Receive data via SPORT3 and save via DAG2 */
r0 = dm(RXSP3A);
pm(i12,m12) = r0;

_main.end:  jump (pc,0);
```

Listing 4-3. SPORT Transmit Using DMA

```
/* SPORT DMA Parameter Registers */
#define IISP4A  0x840
#define IISP5A  0x848
#define IMSP4A  0x841
#define IMSP5A  0x849
#define CSP4A   0x842
#define CSP5A   0x84A

/* SPORT Control Registers */
#define DIV4    0x802
#define DIV5    0x803
#define SPCTL4  0x800
#define SPCTL5  0x801
#define SPMCTL45 0x804

/* SPMCTL Bits */
#define SPL     0x00001000
```

```
/* SPCTL Bits */
#define SPEN_A  0x00000001
#define SDEN_A  0x00040000
#define SLEN32  0x000001F0
#define SPTRAN  0x02000000
#define IFS     0x00004000
#define FSR     0x00002000
#define ICLK    0x00000400

/* Default Buffer Length */
#define BUFSIZE 10

.SECTION/DM seg_dmda;
/*Transmit buffer*/
.var tx_buf5a[BUFSIZE] = 0x11111111,
                         0x22222222,
                         0x33333333,
                         0x44444444,
                         0x55555555,
                         0x66666666,
                         0x77777777,
                         0x88888888,
                         0x99999999,
                         0xAAAAAAAA;

/*Receive buffer*/
.var rx_buf4a[BUFSIZE];

/* Main code section */
.global _main;
.SECTION/PM seg_pmco;
_main:
```

```
/* SPORT Loopback: Use SPORT4 as RX & SPORT5 as TX.
    For no loopback (TDM mode), program MTaCSb
    [a=0,2,4 & b=0,1,2,3] and MRcCSd [a=1,3,5 & b=0,1,2,3], */

/* initially clear SPORT control register */
r0 = 0x00000000;
dm(SPCTL4) = r0;
dm(SPCTL5) = r0;
dm(SPMCTL45) = r0;

SPORT_DMA_setup:
/* SPORT 5 Internal DMA memory address */
r0 = tx_buf5a;    dm(IISP5A) = r0;
/* SPORT 5 Internal DMA memory access modifier  */
r0 = 1;           dm(IMSP5A) = r0;
/* SPORT 5 Number of DMA transfers to be done */
r0 = @tx_buf5a;   dm(CSP5A) = r0;

/* SPORT 4 Internal DMA memory address */
r0 = rx_buf4a;    dm(IISP4A) = r0;
/* SPORT 4 Internal DMA memory access modifier */
r0 = 1;           dm(IMSP4A) = r0;
/* SPORT 4 Number of DMA5 transfers to be done */
r0 = @rx_buf4a;   dm(CSP4A) = r0;

/* set internal loopback bit for SPORT4 & SPORT5 */
bit set ustat3 SPL;
dm(SPMCTL45) = ustat3;

/* Configure SPORT5 as a transmitter */
/* internally generating clock and frame sync */
/* CLKDIV = [fCCLK(200 MHz)/4 x FSCLK(20 MHz)] - 1 = 0x004 */
/* FSDIV = [FSCLK(20 MHz)/TFS(.625 MHz)] - 1 = 31 = 0x001F */
R0 = 0x001F0004;    dm(DIV5) = R0;
```

```
ustat4 = SPEN_A|    /* Enable Channel A */
         SLEN32|    /* 32-bit word length */
         FSR|       /* Frame Sync Required */
         SPTRAN|    /* Transmit on enabled channels */
         SDEN_A|    /* Enable Channel A DMA */
         IFS|       /* Internally Generated Frame Sync */
         ICLK;      /* Internally Generated Clock */
dm(SPCTL5) = ustat4;

/* Configure SPORT4 as a receiver */
/* externally generating clock and frame sync */
r0 = 0x0;     dm(DIV4) = R0;
ustat3 = SPEN_A|    /* Enable Channel A */
         SLEN32|    /* 32-bit word length */
         FSR|       /* Frame Sync Required */
         SDEN_A;    /* Enable Channel A DMA */
dm(SPCTL4) = ustat3;

_main.end:  jump (pc,0);
```

5 SERIAL PERIPHERAL INTERFACE PORT

The ADSP-2126x processor is equipped with a synchronous serial peripheral interface port that is compatible with the industry-standard Serial Peripheral Interface (SPI). The SPI port supports 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 processor's SPI port provides the following features and capabilities:

- A simple 4-wire interface consisting of two data pins, a device select pin, and a clock pin
- Full-duplex operation that allows the ADSP-2126x processor to transmit and receive data simultaneously on the same port
- Special data formats to accommodate little and big endian data, different word lengths, and packing modes
- Master and Slave modes as well as Multimaster mode in which the ADSP-2126x processor can be connected to up to four other SPI devices
- Open drain outputs to avoid data contention and to support multimaster scenarios
- Programmable baud rates, clock polarities, and phases
- Master or slave booting from a master SPI device
- DMA capability to allow transfer of data without core overhead

Functional Description

The SPI interface contains a Transmit Shift (TXSR) and a Receive Shift (RXSR) register. The TXSR register serially transmits data and the RXSR register receives data synchronously with the SPI clock signal (SPICLK). Figure 5-1 shows a block diagram of the 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.

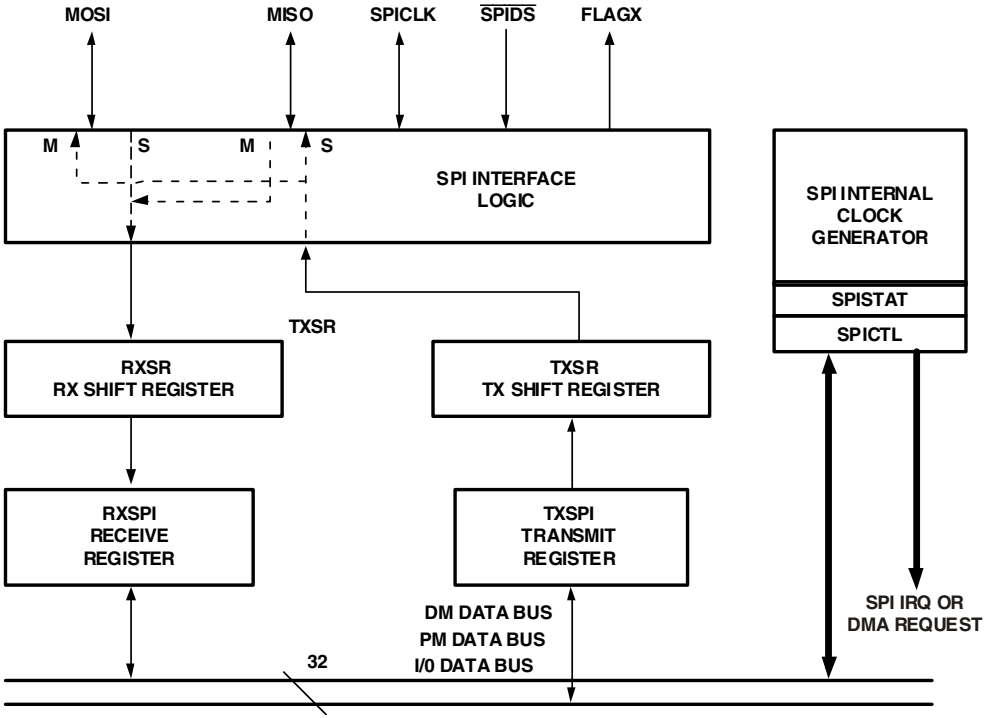


Figure 5-1. SPI Block Diagram

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 (SPIDS). 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.

The SPI port contains a transmit data buffer (TXSPI) and a receive data buffer (RXSPI). Data to be transmitted is written to TXSPI 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 RXSPI, 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 $\overline{\text{SPIDS}}$ of the slave devices.

Different CPUs or DSPs 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.

In a multimaster or multidevice ADSP-2126x processor environment where multiple processors are connected via their SPI ports, all MOSI pins are connected together, all MISO pins are connected together, and the SPICLK pins are connected together as well. The FLGx pins connect to each of the slave SPI devices in the system via their $\overline{\text{SPIDS}}$ pins.

SPI Interface Signals

The SPI protocol uses a 4-wire protocol to enable bidirectional serial communication. This section describes the signals used to connect the SPI ports in a system that has multiple devices. Figure 5-2 shows the master-slave connections between two ADSP-2126x processor devices.

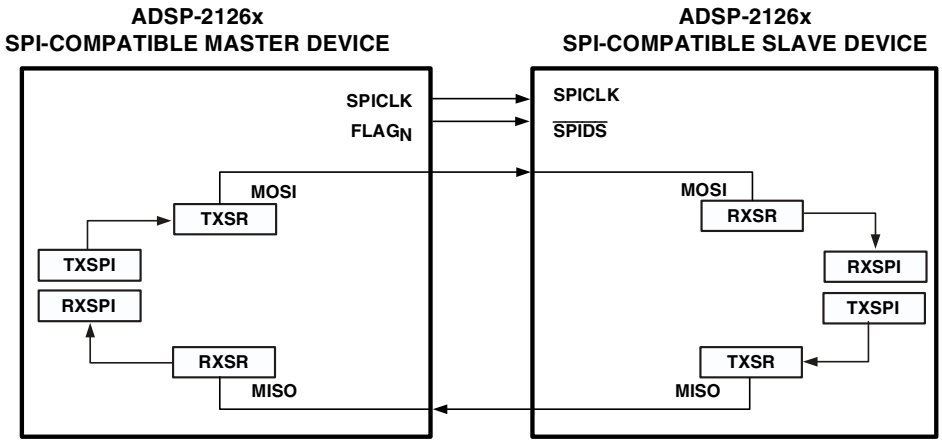


Figure 5-2. Master-Slave Interconnections

SPI Clock Signal (SPICLK)

The SPICLK signal is the Serial Peripheral Interface Clock signal. This control signal is driven by the master and regulates the flow of data bits. The master may transmit data at a variety of baud rates. The SPICLK cycles once for each bit transmitted.

The SPICLK signal is a gated clock that is only active during data transfers, and only for the duration of the transferred word. The number of active edges is equal to the number of bits driven on the data lines. The clock rate can be as high as one-fourth the core clock rate. For master devices, the clock rate is determined by the 15-bit value of the Baud Rate register (SPIBAUD). For more information, see “SPI Baud Setup Register (SPIBAUD)” on page 5-36. For slave devices, the value in the SPIBAUD register is ignored. When the SPI device is a master, SPICLK is an output signal; when the SPI is a slave, SPICLK is an input signal. Slave devices ignore the serial clock if the slave-select input is deasserted (HIGH).

The SPICLK signal is used to shift out the data driven onto the MISO lines and shift in the data driven onto the MOSI lines. The data is always shifted out on one edge of the clock (referred to as the active edge) and sampled on the opposite edge of the clock (referred to as the sampling edge). Clock polarity and clock phase relative to data are programmable via bit 11 (CLKPL) and bit 10 (CPHASE) in the SPICTL control register.

SPICLK Timing

When the processor is configured as an SPI-Slave, the SPI-master must drive an SPICLK signal that conforms with Figure 5-3. For exact timing parameters, please refer to the appropriate ADSP-2126x processor data sheet.

The $\overline{\text{SPIDS}}$ lead time (T1), the $\overline{\text{SPIDS}}$ 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).

SPI Slave Select Outputs (SPIDS0-3)

When CPHASE=0, the SPI port hardware controls the device-select signal automatically (determined by DSxEN bits in SPIFLG). Setting CPHASE=1 requires these signals to be manually controlled in software via the SPIDSx bits in the SPIFLG register (the SPIDSx bits are ignored when CPHASE=0).

SPI Device Select Signal

The $\overline{\text{SPIDS}}$ signal is the Serial Peripheral Interface Device Select Input signal. This is an active low signal used to enable an ADSP-2126x processor configured as a slave device. This input-only pin behaves like a chip select, and is provided by the master device for the slave devices. When the pro-

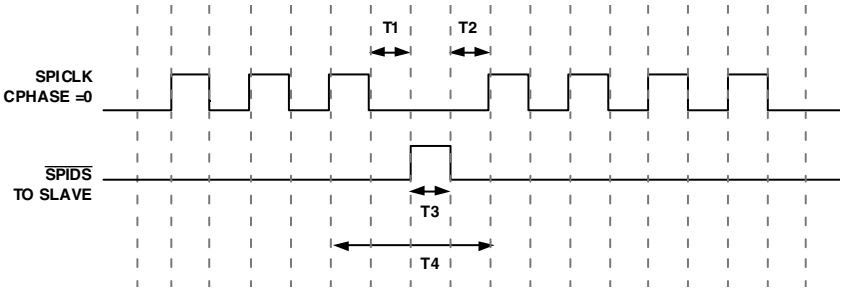


Figure 5-3. SPICLK Timing

cessor is the SPI-master in a multimaster environment, the SPIDS pin acts as an error signal. In multimaster mode, if the SPIDS input signal of a master is asserted (driven low), an error has occurred. This means that another device is also trying to be the master device.

Master Out Slave In (MOSI)

The MOSI pin is one of the bidirectional I/O data pins. If the processor is configured as a master, the MOSI pin becomes a data transmit (output) pin. If the processor is configured as a slave, the MOSI pin becomes a data receive (input) pin. In an ADSP-2126x processor processor SPI interconnection, the data is shifted out from the MOSI output pin of the master and shifted into the MOSI input of the slave.

Master In Slave Out (MISO)

The MISO pin is one of the bidirectional I/O data pins. If the ADSP-2126x processor is configured as a master, the MISO pin becomes a data receive (input) pin. If the ADSP-2126x processor is configured as a slave, the MISO pin becomes a data transmit (output) pin. In an SPI interconnection, the data is shifted out from the MISO output pin of the slave and shifted into the MISO input pin of the master.

Only one slave is allowed to transmit data at any given time. Figure 5-4 illustrates how the ADSP-2126x processor can be used as the slave SPI device. The 8-bit host microcontroller 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.

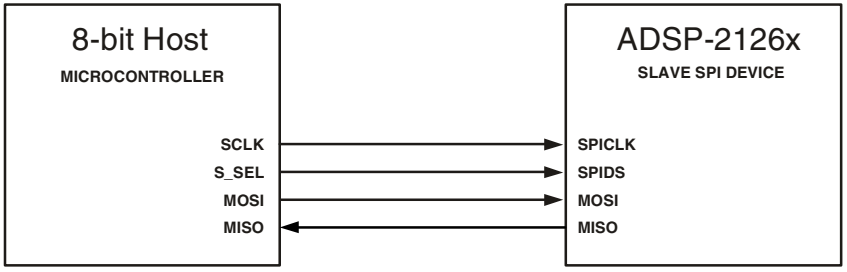


Figure 5-4. ADSP-2126x Processor as SPI Slave

Figure 5-5 illustrates an example of an ADSP-2126x processor SPI interface where the processor is the SPI master. When it uses 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.

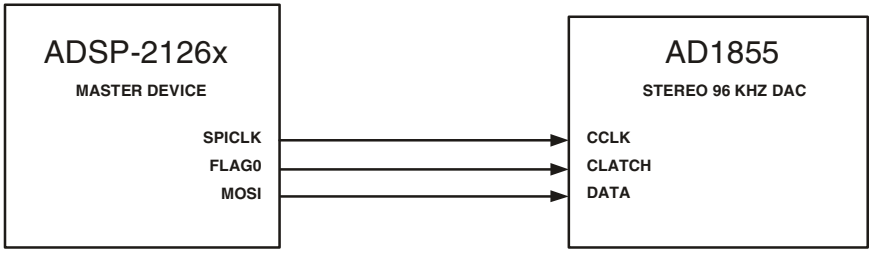


Figure 5-5. ADSP-2126x Processor as SPI Master

SPI General Operations

The SPI in the ADSP-2126x processor can be used in a single master as well as in a multimaster environment. In both configurations, 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. SPI transmission and reception are always enabled simultaneously, unless the Broadcast mode has been selected. In Broadcast mode, several slaves can be configured to receive, but only one of the slaves can be in Transmit mode, driving the MISO line. If the transmit or receive is not needed, MISO can be ignored. This section describes the clock signals, SPI operation as a master and as a slave, and error generation.

SPI Enable

When the SPI is disabled (SPIEN = 0), the flag pins used as slave device selects (FLG0–FLG3) are controlled by the general-purpose flag I/O module, and no data transfers will occur. For slaves, the slave-select input acts like a reset for the internal SPI logic.

When the SPIPDN bit (bit 30 in the PMCTL register) is set (= 1 which shuts down the clock to the SPI), the FLGx pins cannot be used (via the FLGS7-0 register bits) because the FLGx pins are synchronized with the clock.

For this reason, the $\overline{\text{SPIDS}}$ line must be error free. The SPIEN signal can also be used as a software reset of the internal SPI logic. An exception to this is the W1C-type (write 1-to-clear) bits in the SPISTAT Status register will remain set if they are already set. For a list of write 1 to-clear-bits, see Table 5-8 on page 5-45.

Always clear the W1C-type bits before re-enabling the SPI, as these bits will not get cleared even if SPI is disabled. This can be done by writing 0xFF to the SPISTAT register. In the case of an MME error, enable the SPI port after $\overline{\text{SPIDS}}$ is deasserted.

Open Drain Mode (OPD)

In a multimaster or multislave 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 and MISO pins when this option is selected.

When the OPD is set and the SPI port is configured as a master, the MOSI pin is three-stated when the data driven out on MOSI is logic-high. The MOSI pin is not three-stated when the driven data is logic-low. A zero is driven on the MOSI pin in this case. Similarly, when OPD is set and the SPI port is configured as a slave, the MISO pin is three-stated if the data driven out on MISO is logic-high.

Master Mode Operation

When the SPI is configured as a master (and DMA mode is not selected), the SPI port should be configured and transfers started using the following steps:

1. When CPHASE is set to 0, the slave selects are automatically controlled by the SPI port. Otherwise [CPHASE = 1] the slave selects are controlled by the core, and user software controls the pins through the SPIFLGx bits. Before enabling the SPI port, programs should specify which slave-select signal to use by writing to the SPIFLG register, setting one or more of the SPI Flag Select bits (DSxEN).

2. Write to the SPICTL and SPIBAUD registers, enabling the device as a master and configuring the SPI system by specifying the appropriate word length, transfer format, baud rate, and other necessary information.

3. If CPHASE = 1 (user-controlled slave-select signals), activate the desired slaves by clearing one or more of the SPI flag bits (SPIFLG) in the SPIFLG register.

4. Initiate the SPI transfer. The trigger mechanism for starting the transfer is dependant upon the TIMOD bits in the SPICTL register. See Table 5-1 on page 5-17 for details.
5. The SPI generates the programmed clock pulses on SPICLK and simultaneously shifts data out of MOSI and shifts data in from MISO. Before starting to shift, the Transmit Shift register is loaded with the contents of the TXSPI register. At the end of the transfer, the contents of the Receive Shift register are loaded into RXSPI.
6. With each new transfer initiate command, the SPI continues to send and receive words, according to the SPI Transfer mode (TIMOD in SPICTL). See Table 5-1 on page 5-17 for more details.

If the transmit buffer remains empty, or the receive buffer remains full, the device operates according to the states of the SENDZ and GM bits in the SPICTL register.

- If SENDZ = 1 and the transmit buffer is empty, the device repeatedly transmits zero's 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 it 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.

Slave Mode Operation

When a device is enabled as a slave, the start of a transfer is triggered by a transition of the $\overline{\text{SPIDS}}$ Select signal to the active state (LOW) or by the first active edge of the clock (SPICLK), depending on the state of CPHASE.

The following steps illustrate SPI operation in the slave mode:

1. Write to the SPICTL register to make the mode of the serial link the same as the mode that is setup in the SPI master.
2. To prepare for the data transfer, write the data to be transmitted into the TXSPI register.
3. Once the $\overline{\text{SPIDS}}$ signal's falling edge is detected, the slave starts sending and receiving data on active SPICLK edges.
4. The reception or transmission continues until $\overline{\text{SPIDS}}$ is released or until the slave has received the proper number of clock cycles.
5. The slave device continues to receive or transmit with each new falling-edge transition on $\overline{\text{SPIDS}}$ or active SPICLK clock edge.

If the transmit buffer remains empty, or the receive buffer remains full, the devices operate according to the states of the SENDZ and GM bits in the SPICTL register.

- If SENDZ = 1 and the transmit buffer is empty, the device repeatedly transmits zero's on the MISO pin.
- If SENDZ = 0 and the transmit buffer is empty, it repeatedly transmits the last word it 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 MOSI 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.

Multimaster Conditions

A Multimaster mode is implemented to allow an SPI system to transition mastership from one SPI device to another. In a multidevice SPI configuration, several SPI ports are connected and any one of them can become a master at a given time, but only one master is allowed at any one time.

If a processor is a slave and wishes to become the SPI master, it asserts the $\overline{\text{SPIDS}}$ pin for the processor that is currently master and then drives the SPICLK signal. Once it receives the $\overline{\text{SPIDS}}$ signal, the device that was master is immediately reconfigured as a slave. In order to safely transition from one master to the other the SPI port features the use of open drain outputs for the data pad drivers in order to avoid data contention.

More information on this topic is described in "Mode Fault Error (MME)" on page 5-53.

SPI Data Transfer Operations

The following sections provide information on the two methods the ADSP-2126x processor uses to transfer data; through the core or through DMA.

Core Transmit and Receive Operations

For core-driven SPI transfers, the software has to read from or write to the RXSPI and TXSPI registers to control the transfer. It is important to check the buffer status before reading from or writing to these registers because the core *does not* hang when it attempts to read from an empty buffer or

write to a full buffer. When the core writes to a full buffer, the data that is in that buffer is overwritten and the SPI begins transmitting the new data. Invalid data is obtained when the core reads from an empty buffer.

For a master, when the transmit buffer becomes empty, or the receive buffer becomes full, the SPI device stalls the SPI clock until it reads all the data from the receive buffer or it detects that the transmit buffer contains a piece of data.

- For a master configured with TIMOD = 01: When the transmit buffer becomes empty, the SPI device stalls the SPI clock until a piece of data is written to the transmit buffer.
- For a master configured with TIMOD = 00: When the receive buffer becomes full the SPI device stalls the SPI clock until all of the data is read from the receive buffer.

SPI DMA

The SPI has a single DMA channel associated with it that can be configured to support either an SPI transmit or a receive channel, but not both simultaneously. In addition to the TXSPI and RXSPI data buffers, there is a four-word deep DMA FIFO the SPI port uses to improve throughput.

The SPI port supports both Master mode and Slave mode DMA. The following sections describe Slave and Master mode DMA operation, DMA chaining, switching between transmit and receive DMA operations, and processing DMA interrupt errors.

Do not write to the TXSPI register during an active SPI transmit DMA operation because DMA data will be overwritten. However, writes to the TXSPI register during an active SPI receive DMA operation are permitted. The RXS register is cleared when the RXSPI register is read. Reads from the RXSPI register are allowed at any time during transmit DMA. Interrupts are generated based on DMA events and are configured in the SPIDMAC register.

Similarly, do not read from the RXSPI register during active SPI DMA receive operations.

In order for a transmit DMA operation to begin, the transmit buffer must initially be empty (TXS = 0). While this is normally the case, this means that the TXSPI register should not be used for any purpose other than SPI transfers. For example, the TXSPI register should not be used as a scratch register for temporary data storage. Writing to the TXSPI register via the software sets the TXS bit.

When the SPI DMA engine is configured for transmitting:

1. The receive interface cannot generate an interrupt, but the status can be polled.
2. The four-deep FIFO is not available in the receive path.

Similarly, when the SPI DMA engine is configured for receiving,

1. The transmit interface cannot generate an interrupt, but the status can be polled.
2. The four-deep FIFO is not available in the transmit path.

Master Mode DMA Operation

To configure the SPI port for Master mode DMA transfers:

1. Specify which FLG pin(s) to use as the slave-select signal(s) by setting one or more of the SPI Flag (SPIFLG register) Select bits (DSxEN bits 3–0).
2. Enable the device as a master and configure the SPI system by selecting the appropriate word length, transfer format, baud rate, and so on in the SPIBAUD and SPICTL registers. The TIMOD field (bits 1–0) in the SPICTL register is configured to select transmit or receive with DMA mode (TIMOD = 10).

3. Activate the desired slaves by clearing one or more of the SPI flag bits (SPIFLGx) of SPIFLG if CPHASE = 1.

4. For a single DMA, define the parameters of the DMA transfer by writing to the IISPI, IMSPI, and CSPI registers. For DMA chaining, write the chain pointer address to the CPSPI register. The CPSPI register is a 20-bit read-write register that can contain address information.

5. Write to the SPI DMA configuration register, (SPIDMAC), to specify the DMA direction (SPIRCV, bit 1) and to enable the SPI DMA engine (SPIDEN, bit 0). If DMA chaining is desired, set (= 1) the SPICHEN bit (bit 4) in the SPIDMAC register.

To avoid data corruption, enable the SPI port before enabling DMA.

If flags are used as slave selects, programs should activate the flags by clearing the flag after SPICTL and SPIBAUD are configured, but before enabling the DMA. When CPHASE = 0, and a program is using DMA, the program must use automatic flags using SPIFLGx.

When enabled as a master, the DMA engine transmits or receives data as follows:

1. If the SPI system is configured for transmitting, the DMA engine reads data from memory into the SPI DMA FIFO. Data from the DMA FIFO is loaded into the TXSPI register and then into the Transmit Shift register. This initiates the transfer on the SPI port.

2. If configured to receive, data from RXSPI is automatically loaded into the SPI DMA FIFO, the DMA engine reads data from the SPI DMA FIFO and writes to memory. Finally, the SPI initiates the receive transfer.

3. The SPI generates the programmed signal pulses on SPICLK and simultaneously shifts data out of MOSI and shifts data in from MISO.

4. The SPI continues sending or receiving words until the SPI DMA word count register transitions from 1 to 0.

If the DMA engine is unable to keep up with the transmit stream during a transmit operation because the IOP requires the IOD (I/O data) bus to service another DMA channel (or for another reason), the SPICLK stalls until data is written into the TXSPI register. All aspects of SPI receive operation should be ignored. The data in the RXSPI register is not intended to be used, and the RXS (bits 28–27 and 31–30 in the SPCTLx register) and SPISTAT bits (bits 26 and 29) should be ignored. The ROVF overrun condition cannot generate an error interrupt in this mode.

If the DMA engine cannot keep up with the receive data stream during receive operations, then SPICLK stalls until data is read from RXSPI. While performing a receive DMA, the processor core assumes the transmit buffer is empty. If SENDZ = 1, the device repeatedly transmits 0's on the MOSI pin. If SENDZ = 0, it repeatedly transmits the contents of the TXSPI register. The TUNF underrun condition cannot generate an error interrupt in this mode.

For receive DMA in master mode the SPICLK stops only when the FIFO and RXSPI buffer is full (even if the DMA count is zero). Therefore, SPICLK runs for an additional five word transfers filling junk data in the FIFO and the RXSPI buffer. This data must be cleared before a new DMA is initiated.

A master SPI DMA sequence may involve back-to-back transmission and/or reception of multiple DMA transfers. The SPI controller supports such a sequence with minimal processor core interaction.

Master Transfer Preparation

When the processor is enabled as a master, the initiation of a transfer is defined by the two bit fields (bits 1–0) of TIMOD in the SPICTL register. Based on these two bits and the status of the interface, a new transfer is started upon either a read of the RXSPI register or a write to the TXSPI register. This is summarized in Table 5-1.

Table 5-1. Transfer Initiation

TIMOD	Function	Transfer Initiated Upon	Action, Interrupt
00	Transmit and Receive	Initiate new single word transfer upon read of RXSPI and previous transfer completed.	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	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. For more information, see “DMA Transfer Direction” on page 2-21.
11	Reserved		

Slave Mode DMA Operation

A Slave mode DMA transfer occurs when the SPI port is enabled and configured in Slave mode, and DMA is enabled. When the $\overline{\texttt{SPIDS}}$ signal transitions to the active-low state or when the first active edge of SPICLK is detected, it triggers the start of a transfer.

To configure for Slave mode DMA:

1. Write to the SPICTL register to make the mode of the serial link the same as the mode that is set up in the SPI master. Configure the TIMOD field to select transmit or receive DMA mode (TIMOD = 10).
2. Define a DMA receive (or transmit) transfer by writing to the IISPI, IMSPI, and CSPI registers. For DMA chaining, write to the chain pointer address of the CPSPI register.
3. Write to the SPIDMAC register to enable the SPI DMA engine and configure:
   - A receive access (SPIRCV = 1) or
   - A transmit access (SPIRCV = 0)

   If DMA chaining is desired, set the SPICHEN bit in the SPIDMAC register.

Enable the SPI port before enabling DMA to avoid data corruption.

Slave Transfer Preparation

When enabled as a slave, the device prepares for a new transfer according to the function and actions described in Table 5-1.

The following steps illustrate the SPI receive or transmit DMA sequence in an SPI slave in response to a master command:

1. Once the slave-select input is active, the processor starts receiving and transmitting data on active SPICLK edges. The data for one channel (TX or RX) is automatically transferred to/from memory by the IOP. The function of the other channel is dependant on the GM and SENDZ bits in the SPICTL register.

2. Reception or transmission continues until the SPI DMA word count register transitions from 1 to 0.
3. A number of conditions can occur while the processor is configured for Slave mode:
    - If the DMA engine cannot keep up with the receive data stream during receive operations, the receive buffer operates according to the state of the GM bit in the SPICTL register.
    - If GM = 0 and the DMA buffer is full, the incoming data is discarded, and the RXSPI register is not updated. While performing a receive DMA, the transmit buffer is assumed to be empty. If SENDZ = 1, the device repeatedly transmits zero's on the MOSI pin. If SENDZ = 0, it repeatedly transmits the contents of the TXSPI register.
    - If GM = 1 and the DMA buffer is full, the device continues to receive new data from the MISO pin, overwriting the older data in the DMA buffer.
    - If the DMA engine cannot keep up with the transmit data stream during a transmit operation because another DMA engine has been granted the bus (or for another reason), the transmit port operates according to the state of the SENDZ bit in the SPICTL register.

      If SENDZ = 1 and the DMA buffer is empty, the device repeatedly transmits zero's on the MOSI pin. If SENDZ = 0 and the DMA buffer is empty, it repeatedly transmits the last word it transmitted before the DMA buffer became empty. All aspects of SPI receive operation should be ignored. The data in the RXSPI register is not intended to be used, and the RXS and ROVF bits should be ignored. The ROVF overrun condition cannot generate an error interrupt in this mode.

While a DMA transfer may be used on one channel (TX or RX), the core, (based on the RXS and TXS status bits), can transfer data in the other direction.

Changing SPI 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 an SPI communication link consists of the following: 1) a single master and a single slave, 2) CPHASE = 1, and 3) the slave's slave select input is tied low, 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.

When performing transmit operations with the SPI port, disabling the SPI port prematurely can cause data to be corrupted and or not fully transmitted. Before the program disables the SPI port in order to reconfigure it, the status bits should be polled to ensure that all valid data has been completely transferred. For core-driven transfers, data moves from the TXSPI buffer into a shift register. The following bits should be checked before disabling the SPI port:

1. Wait for the TXSPI buffer to empty into the shift register. This is done when the TXS bit, (bit 3) of the SPISTAT register becomes zero.

2. Wait for the SPI Shift register to finish shifting out data This is done when the SPIF bit, (bit 0) of the SPISTAT register becomes one.

3. Disable the SPI Port by setting the SPIEN bit, (bit 0) in the SPICTL register, to zero.

When performing transmit DMA transfers, data moves through a four deep SPI DMA FIFO, then into the TXSPI buffer, and finally into the shift register. DMA interrupts are latched when the I/O processor moves the last word from memory to the peripheral. For the SPI, this means that the SPI "DMA complete" interrupt is latched when there are still six words left to be fully transmitted (four in the FIFO, one in the TXSPI buffer, and one being shifted out of the Shift register). To disable the SPI port after a DMA transmit operation, use these steps:

1. Wait for DMA FIFO to empty. This is done when the SPISx bits (bits 13–12) in the SPIDMAC register become zero.

2. Wait for the TXSPI register to empty. This is done when the TXS bit (bit 3) in the SPISTAT register becomes zero.

3. Wait for the SPI Shift register to finish transferring the last word. This is done when the SPIF bit, (bit 0) of the SPISTAT register, becomes one.

4. Disable the SPI Port by setting the SPIEN bit, (bit 0) of the SPICTL register, to zero.

Switching From Transmit To Receive DMA

The following sequence details the steps for switching from transmit to receive DMA.

With disabling the SPI:

1. Write 0x00 to the SPICTL register to disable SPI. Disabling the SPI also clears the RXSPI/TXSPI registers and the buffer status.
2. Disable DMA by writing 0x00 to the SPIDMAC register.
3. Clear all errors by writing to the W1C-type bits in the SPISTAT register. This ensures that no interrupts occur due to errors from a previous DMA operation.
4. Reconfigure the SPICTL register and enable the SPI port.
5. Configure DMA by writing to the DMA parameter registers and enable DMA.

Without disabling the SPI:

1. Clear RXSPI/TXSPI without disabling SPI. This can be done by ORing 0xc0000 with the present value in the SPICTL register. For example programs can use the RXFLSH and TXFLSH bits to clear TXSPI/RXSPI.
2. Disable DMA by writing 0x00 to the SPIDMAC register.
3. Clear all errors by writing to the MME bit (bit 1) in the SPISTAT register. This ensures that no interrupts occur due to errors from a previous DMA operation.
4. Reconfigure the SPICTL register to clear the TXSPI/RXSPI register values.
5. Configure DMA by writing to the DMA parameter registers and enabling DMA.

Switching From Receive to Transmit DMA

Use the following sequence to switch from receive to transmit DMA. Note that TXSPI and RXSPI are registers but they may not contain any bits, only address information.

With disabling of the SPI:

1. Write 0x00 to the SPICTL register to disable SPI. Disabling SPI also clears the RXSPI/TXSPI register contents and the buffer status.
2. Disable DMA and clear the DMA FIFO by writing 0x80 to the SPIDMAC register. This ensures that any data from a previous DMA operation is cleared because the SPICLK signal runs for five more word transfers even after the DMA count falls to zero in receive DMA.
3. Clear all errors by writing to the SPISTAT register. This ensures that no interrupts occur due to errors from a previous DMA operation.
4. Reconfigure the SPICTL register and enable SPI.
5. Configure DMA by writing to the DMA parameter registers and the SPIDMAC register.

Without disabling the SPI:

1. Clear RXSPI/TXSPI without disabling the SPI. This can be done by ORing 0xc0000 with the present value in the SPICTL register. Use the RXFLSH (bit 19) and TXFLSH (bit 18 in the SPICTL register) bits to clear the RXSPI/TXSPI registers.
2. Disable DMA and clear the FIFO. For example, write 0x80 to the SPIDMAC register. This ensures that any data from a previous DMA operation clears because the SPICLK runs for five more word transfers even after the DMA count is zero in receive DMA.

3. Clear all errors by writing to the W1C-type bits in the SPISTAT register. This ensures that no interrupts occur due to errors from a previous DMA operation.
4. Reconfigure the SPICTL register to clear the TXSPI/RXSPI registers.
5. Configure DMA by writing to the DMA parameter registers and the SPIDMAC register using the SPIDEN bit (bit 0). These registers are described in Table 2-4 on page 2-25.

DMA Error Interrupts

The SPIUNF and SPIOVF bits of the SPIDMAC register 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 disabling the SPI:

1. Disable the SPI port by writing 0x00 to the SPICTL register.
2. Disable DMA and clear the FIFO. For example, write 0x80 to the SPIDMAC register. This ensures that any data from a previous DMA operation clears before configuring a new DMA operation.
3. Clear all errors by writing to the W1C-type bits (see Table 5-8 on page 5-45) in the SPISTAT register. This ensures that the error bits SPIOVF and SPIUNF (in the SPIDMAC register) clear when a new DMA is configured.
4. Reconfigure the SPICTL register and enable SPI using the SPIEN bit.
5. Configure DMA by writing to the DMA parameter registers and the SPIDMAC register.

Without disabling the SPI:

1. Disable DMA and clear the FIFO. For example, write 0x80 to the SPIDMAC register. This ensures that any data from a previous DMA operation clears before configuring a new DMA operation.
2. Clear RXSPI/TXSPI without disabling SPI. This can be done by ORing 0xc0000 with the present value in the SPICTL register. Use the RXFLSH and TXFLSH bits to clear the RXSPI/TXSPI registers.
3. Clear all errors by writing to the W1C-type bits in the SPISTAT register. This ensures that error bits SPIOVF and SPIUNF in the SPIDMAC register are cleared when a new DMA is configured.
4. Reconfigure SPICTL to clear the RXSPI/TXSPI register bits.
5. Configure DMA by writing to the DMA parameter registers and the SPIDMAC register.

DMA Chaining

DMA chaining is enabled when the SPICHEN bit is set to 1 in the SPIDMAC register. In this mode, the DMA registers are loaded using a DMA transfer from a predefined Transfer Control Block (TCB). When this load occurs, it causes the Chaining Status bit (SPICHS) to be set. Once the chain pointer load completes, the SPICHS bit is cleared. Upon completion of the transfer block load, the normal DMA transfer is initiated. Table 5-2 describes the order of loading. For more information about chaining, refer to "Chaining DMA Processes" on page 2-10.

Table 5-2. DMA Chaining Sequence

Address	Register	Description
CPSPI	DMA Start Address	Address in Memory
CPSPI – 1	DMA Address Modifier	Address increment

Table 5-2. DMA Chaining Sequence (Cont'd)

Address	Register	Description
CPSPI – 2	DMA Word Count	Number of words to transfer
CPSPI – 3	DMA Next TCB	Pointer to address of next TCB

SPI Transfer Formats

The ADSP-2126x processor SPI supports four different combinations of serial clock phases and polarity. The application code can select any of these combinations using the CLKPL and CPHASE bits in the SPICTL register.

Figure 5-6 on page 5-27 shows the transfer format when CPHASE = 0 and Figure 5-7 on page 5-28 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 $\overline{\text{SPIDS}}$ 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, $\overline{\text{SPIDS}}$, must be inactive (HIGH) between each word in the transfer. When CPHASE = 1, $\overline{\text{SPIDS}}$ may either remain active (LOW) between successive transfers or be inactive (HIGH).

Figure 5-6 shows the SPI transfer protocol for CPHASE = 0. Note that SPICLK starts toggling in the middle of the data transfer, WL = 0, and MSBF = 0.

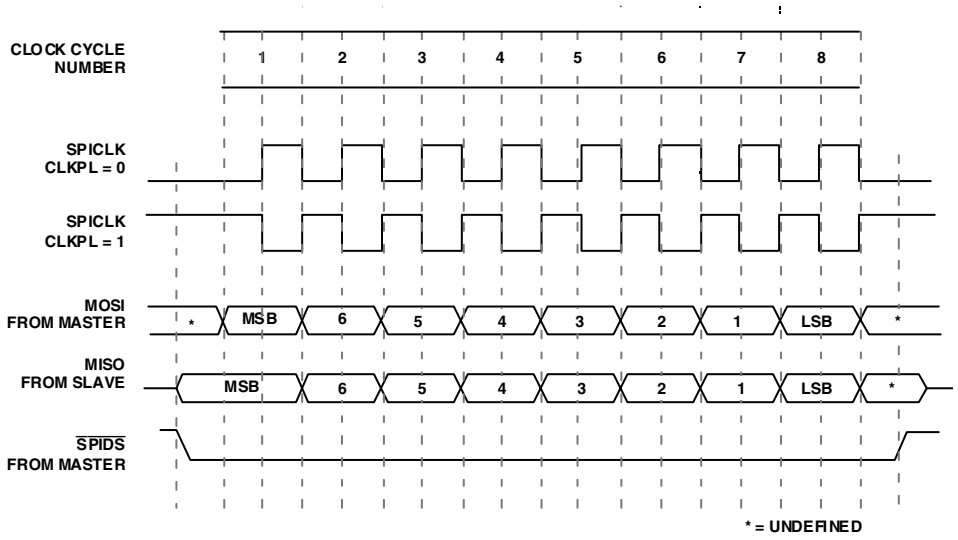


Figure 5-6. SPI Transfer Protocol for CPHASE = 0

Figure 5-7 shows the SPI transfer protocol for CPHASE = 1. Note that SPICLK starts toggling at the beginning of the data transfer, WL = 0, and MSBF = 0.

Beginning and Ending an SPI Transfer

An SPI transfer's defined start and end depend on the following: whether the device is configured as a master or a slave, whether the CPHASE mode is selected, and whether the transfer initiation mode is (TIMOD) selected. For

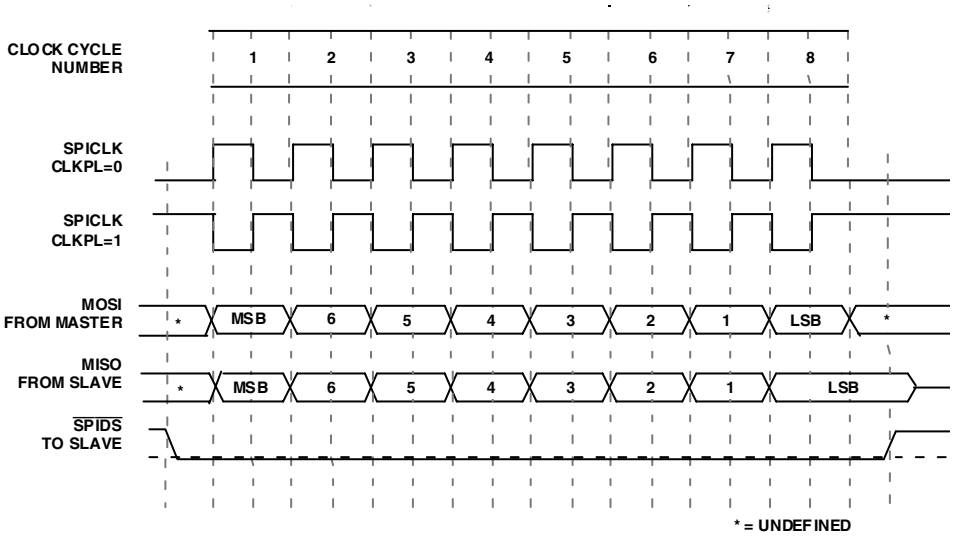


Figure 5-7. SPI Transfer Protocol for CPHASE = 1

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 the SPICLK period. For a slave with CPHASE = 0, the transfer starts as soon as the $\overline{\text{SPIDS}}$ 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.

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 just been received and latched into the receive buffer, RXSPI. The RXS bit is set

shortly after the last sampling edge of SPICLK. The latency is typically a few core clock cycles and is independent of CPHASE, TIMOD, and the baud rate. If configured to generate an interrupt when RXSPI is full (TIMOD = 00), the interrupt becomes active one core clock cycle after RXS is set. When not relying on this interrupt, the end of a transfer can be detected by polling the RXS bit.

To maintain software compatibility with other SPI devices, 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 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 is set, and consequently before new data has been latched into the RXSPI buffer. 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 is set.

SPI Word Lengths

The processor's SPI port can transmit and receive the word widths described in the following sections.

8-Bit Word Lengths

Eight-bit word lengths can be used when transmitting or receiving. When transmitting, the SPI port sends out only the lower eight bits of the word written to the SPI buffer.

For example, if the processor executes the instructions below, the SPI port transmits `0x78`.

```
r0 = 0x12345678
dm(TXSPI) = r0;
```

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.

For example, if an SPI host sends the processor the 32-bit word `0x12345678`, the processor receives the following words:

```
0x00000078 //first word
0x00000056 //second word
0x00000034 //third word
0x00000012 //forth word
```

This code works only if the `MSBF` bit is zero in both the transmitter and receiver, and the `SPICLK` frequency is small. If `MSBF` = 1 in the transmitter and receiver, and `SPICLK` has a small frequency, the received words follow the order `0x12`, `0x34`, `0x56`, `0x78`.

16-Bit Word Lengths

Sixteen-bit word lengths can be used when transmitting or receiving. When transmitting, the SPI port sends out only the lower 16 bits of the word written to the SPI buffer.

For example, if the processor executes the following instructions, the SPI port transmits `0x5678`.

```
r0 = 0x12345678
dm(TXSPI) = r0;
```

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.

For example, if an SPI host sends the processor the 32-bit word `0x12345678`, the processor receives the following words:

```
0x00005678  //first word
0x00001234  //second word
```

32-Bit Word Lengths

Thirty-two bit word lengths can be used when transmitting or receiving. No packing of the `RXSPI` or `TXSPI` registers is necessary as the entire 32-bit register is used for the data word.

Packing

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. Packing is enabled through the `PACKEN` bit in the `SPICTL` register. The SPI is unpacks data when it transmits and packs data when it receives. When packing is enabled, two 8-bit words are packed into one 32-bit word.

When the SPI port is transmitting, two eight-bit words are packed into one 32-bit word. When receiving, words are unpacked from one 32-bit word into two eight-bit words.

Transmitter packing example:

The value 0xXXLMXXJK (where XX is any random value and JK and LM are the 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.

Receiver packing example:

The receiver unpacks the value and two words are received, 0xJK and then 0xLM. They appear in the RXSPI register as:

0x00LM00JK => if SGN is configured to 0
0xFFLMFFJK => if SGN is configured to 1 and L, J > 7.

SPI Interrupts

The SPI port can generate an interrupt in five different situations. During core-driven transfers, an SPI interrupt is triggered in these instances:

1. When the TXSPI buffer has the capacity to accept another word from the core
2. When the RXSPI buffer contains a valid word to be retrieved by the core

The TIMOD (Transfer Initiation and Interrupt) register determines whether the interrupt is based on the TXSPI or RXSPI buffer status. For more information, refer to the TIMOD bit descriptions in the SPICTL register in Table 5-6 on page 5-37.

During IOP-driven transfers (DMA), an SPI interrupt is triggered in these instances:

1. When a single DMA transfer completes
2. When a number of DMA sequences (if DMA chaining is enabled) completes
3. When a DMA error has occurred

Again, the TIMOD register must be initialized properly to enable DMA interrupts.

All of these interrupts are serviced using the high priority (SPIHI) or low priority (SPILI) SPI interrupt. Whenever an SPI interrupt occurs (regardless of the cause), both SPILI and SPIHI are latched. Programs specify the SPI interrupt priority by masking (disabling) one of the interrupts. To service the SPI port using the high priority interrupt, unmask (set = 1) the SPIHI bit (bit 12) in the IMASK register. To service the SPI port using the low priority interrupt, unmask (set = 1) the SPILIMSK bit (bit 19) in the LIRPTL register. For a list of these bits, see Table 2-1 on page 2-5.

To globally enable interrupts set (= 1), the IRPTEN bit in the MODE1 register. When using DMA transfers, programs must also specify whether to generate interrupts based on transfer or error status. For DMA transfer status based interrupts, set the INTEN bit in the SPIDMAC register; otherwise, set the INTERR bit to trigger the interrupt if one of the error conditions is triggered during the transmission—multimaster error (MME), transmit buffer underflow (TUNF – only if SPIRCV = 0), or receive buffer overflow (ROVF – only if SPIRCV = 1). During core-driven transfers, the TUNF and ROVF error conditions do not generate interrupts.

When DMA is disabled, the processor core may read from the RXSPI register or write to the TXSPI data buffer. The RXSPI and TXSPI buffers are memory-mapped IOP registers. A maskable interrupt is generated when the receive buffer is not empty or the transmit buffer is not full. The TUNF and ROVF error conditions do not generate interrupts in these modes.

- See the *ADSP-2126x SHARC DSP Core Manual* for IRPTL and LIRPTL register bit descriptions.
- See "SPI DMA Configuration Register (SPIDMAC)" on page A-50 for SPIDMAC register bit descriptions.

SPI Registers

The SPI peripheral in the ADSP-2126x SHARC processor includes several memory mapped registers, some of which are accessible by the IOP. Four registers contain control and status information—SPIBAUD, SPICTL, SPIFLG, and SPISTAT. Two registers are used for buffering receive and transmit data—RXSPI and TXSPI. Five registers are related to DMA functionality—SPIDMAC, IISPI, IMSPI, CSPI and CPSPI. Additionally, the four-deep SPI DMA FIFO and the SPI Transmit and Receive Shift registers, TXSR and RXSR, are not accessible.

SPI registers are provided in several functional groups.

Table 5-3. SPI Registers

SPI Register Type	Registers	Found On
Control and Status Registers	SPIBAUD	page 5-36
	SPICTL	page 5-37
	SPIFLG	page 5-41
	SPISTAT	page 5-45
Buffering and Transmit/Receive Registers	TXSPI	on page 5-47
	RXSPI	on page 5-48
Shift Registers	RXSR	on page 5-52
	TXSR	on page 5-52
Debugging Register	RXSPI_SHADOW	on page 5-53

Table 5-3. SPI Registers (Cont'd)

SPI Register Type	Registers	Found On
DMA Registers	SPIDMAC	on page 5-48
	IISPI	on page 5-48
	IMSPI	on page 5-48
	CSPI	on page 5-48
	CPSPI	on page 5-48

Also see "Error Signals and Flags" on page 5-53 for more information about how the bits in these registers are used to signal errors and other conditions, and on page 5-48 for a table that shows the mapping of all SPI registers.

Control and Status Registers

The following registers are used to control certain functions of the SPI or to provide SPI status information:

- SPIBAUD
- SPICTL
- SPIFLG
- SPISTAT

SPI Baud Setup Register (SPIBAUD)

The SPI Baud Rate register (SPIBAUD) is used to set the bit transfer rate for a master device. When configured as a slave, the value written to this register is ignored. The serial clock frequency is determined by the following formula:

$$\text{SPI Baud Rate} = \frac{\text{Core Clock Rate}}{(4)\text{x}(\text{BAUDR})}$$

Writing a value of zero to the register disables the serial clock. Therefore, the maximum serial clock rate is one-fourth the core clock rate (CCLK).

Table 5-4 provides the bit descriptions for the SPIBAUD register.

Table 5-4. SPIBAUD Register Bits

Bit(s)	Name	Function	Default
0		Reserved	
15:1	BAUDR	**Baud Rate e**nables the SPICLK baud rate per the following equation: SPI Baud Rate = Core clock (CCLK) divided by (4* BAUDR)	0
31:16	Reserved		

Table 5-5 lists several possible baud rate values for the SPIBAUD register.

Table 5-5. SPI Master Baud Rate Example

BAUDR Decimal Value	SPI Clock Divide Factor	Baud Rate for CCLK @ 200 MHz
0	N/A	N/A
1	4	50 MHz
2	8	25 MHz
3	12	16.67 MHz
4	16	12.5 MHz

Table 5-5. SPI Master Baud Rate Example (Cont'd)

BAUDR Decimal Value	SPI Clock Divide Factor	Baud Rate for CCLK @ 200 MHz
5	20	10.0 MHz
and up to 32,767 (0x7FFF)[1]	131,068	1.526 kHz

1 BAUDR decimal values of 6 to 32,766 are also possible.

SPI Control Register (SPICTL)

The SPI Control register (SPICTL) is used to configure and enable the SPI system. This read/write register is used to enable the SPI interface, select the device as a master or slave, and determine the data transfer format and word size. Table 5-6 provides the bit descriptions for the SPICTL register.

Table 5-6. SPICTL Register Bits

Bit(s)	Name	Function	Default
1:0	TIMOD	**Transfer Initiation Mode.** Defines the transfer initiation mode and interrupt generation as follows: 00 = Initiate transfer by read of receive buffer. Interrupt is active when receive buffer is full. 01 = Initiate transfer by write to transmit buffer. Interrupt is latched when transmit buffer is empty 10 = Enable DMA transfer mode. Interrupt is configured by DMA. 11 = Reserved	00
2	SENDZ	**Send Zero.** When receiving via the RXSPI register, the SPI also transmits the contents of the TXSPI register. When TXSPI is empty, the SPI data output pin (MOSI or MISO) can either transmit zeros or retransmit the last word written to TXSPI. 0 = Resend last word written to TXSPI 1 = Send zero	0

Table 5-6. SPICTL Register Bits (Cont'd)

Bit(s)	Name	Function	Default
3	GM	**Get Data.** When transmitting, the RXSPI register also latches in data. When RXSPI is full, it can either latch in more data or discard all further incoming data. 0 = Discard incoming data 1 = Get more data (overwrites the previous data)	0
4	ISSEN	**Input Slave Select Enable**. Enables if set (= 1) or disables if cleared (= 0) Slave-Select ($\overline{\text{SPIDS}}$) input for master. When not used, $\overline{\text{SPIDS}}$ can be disabled, freeing up a chip pin as general-purpose I/O. Note: This bit is ignored for slave mode.	0
5	DMISO	**Disable MISO Pin.** Disables the MISO pin as an output. This is needed when the master wishes to transmit to various slaves at one time (broadcast). Only one slave is allowed to transmit data back to the master. All slaves (except for the one from whom the master wishes to receive) should have this bit set. 1 = MISO disabled 0 = MISO enabled	0
6	Reserved		
8:7	WL	**Word Length.** The SPI port can transmit the least significant 8 bits, 16 bits, or all 32 bits of the TXSPI register. Likewise, when receiving, the RXSPI buffer can be considered "full" after 8 bits, 16 bits, or 32 bits have been shifted in. These bits select word length as follows: 00 = 8 bits 01 = 16 bits 10 = 32 bits	00
9	MSBF	**Data Format.** Selects the data format as follows: 1 = MSB sent/received first 0 = LSB sent/received first	0

Table 5-6. SPICTL Register Bits (Cont'd)

Bit(s)	Name	Function	Default
10	CPHASE	**Clock Phase.** Selects the relationship between SPICLK and the SPI data. It also determines whether the slave-select output flags are controlled by the SPI port or manually by the core. 0 = SPICLK starts toggling at the middle of the first data bit; slave-select flags are automatically asserted by the SPI port. 1 = SPICLK starts toggling at the beginning of the first data bit; slave-select flags require manual assertion by the core.	1
11	CLKPL	**Clock Polarity.** Selects the clock polarity as follows: 0 = Active high SPICLK (SPICLK low is the idle state) 1 = Active low SPICLK (SPICLK high is the idle state)	0
12	SPIMS	**Master Select.** This bit configures the SPI as master or slave as follows: 0 = Device is a slave device 1 = Device is a master device	0
13	OPD	**Open Drain Data Output Enable.** Enables an open drain data output for MOSI and MISO. 0 = Normal 1 = Open Drain	0
14	SPIEN	**SPI Port Enable.** Enables the SPI port if set (= 1) or disables the port if cleared (= 0).	0
15	PACKEN	**Packing Enable.** Selects the word packing mode in receive mode as follows: 0 = No Packing 1 = 8- to 32-bit packing is enabled This is valid when WL = 00. When in Transmit mode, this bit will unpack data.	0
16	SGN	**Sign Extend Bit.** When packing is enabled, only the eight LSB's of each word contain valid data. This bit determines whether the upper 24 MSB's are sign-extended based on bit 7 or left as zeros.	0

Table 5-6. SPICTL Register Bits (Cont'd)

Bit(s)	Name	Function	Default
17	SMLS	**Seamless Transfer Bit.** Enables seamless transfers. When it is set (= 1), this bit indicates no delay before the next word starts, a seamless operation. When this bit is cleared (= 0), it indicates that after each word transfer there is a delay before the next word transfer starts. The delay is 2.5 SPICLK cycles.	0
18	TXFLSH	**Clear TXS Bit.** When set to 1, clears the TXS bit but does not clear the buffer. Any data in the buffer becomes invalid. 0 = TXS not cleared 1 = TXS cleared	0
19	RXFLSH	**Clear RXS Bit.** When set to 1, clears the RXS bit but does not clear the buffer. Any data in the buffer becomes invalid. 0 = RXS not cleared 1 = RXS cleared	0
31:20	Reserved		

SPI Flag Register (SPIFLG)

The SPI Flag register (SPIFLG) is a read/write register used to enable individual SPI slave-select output flags when the SPI is enabled as a master. The SPIFLG register has four bits to select the outputs to be driven as slave-select lines (DSxEN) and four bits to activate the selected slave-selects (SPIFLG).

If the SPI is enabled and configured as a master, up to four of the chip's general-purpose flag I/O pins may be used as slave-select outputs. For each DSxEN bit set in the SPIFLG register, the corresponding FLG pin is configured as a slave-select output. For example, if bit DS1EN is set in the SPIFLG register, the SPIFLG1 pin is driven as a slave-select.

For those DSxEN bits that are not set, the corresponding FLG pins are configured and controlled by the chip's FLG I/O module.

Table 5-7 provides the bits for the SPIFLG register.

Table 5-7. SPIFLG Register Bits

Bit	Name	Function	Default
3:0	DSxEN (3–0)	**SPI Device Select Enable Bits**. Selects (if set = 1) the corresponding FLG as a pin to be used for SPI slave-select.	0
6:4	Reserved		
7	ISSS	**Input Slave Select Status.** This read-only bit reflects the status of the slave-select input pin.	
11:8	SPIFLGx (3–0)	**SPI Flags.** Drives the value of the FLG selected using DSxEN (if CPHASE = 1). If cleared (= 0) the corresponding slaves will be selected.	1
15:12	Reserved		
31:16	Reserved		

When the FLG pins are configured as slave-select outputs, the value that is driven onto these outputs depends on the value of the CPHASE bit in the SPICTL register:

- If CPHASE = 1, the slave-select flags are manually controlled in software by setting and clearing the appropriate FLGx bit.
- If CPHASE = 0, the slave-select flags enabled by DSxEN bits of the SPIFLG register are automatically asserted by the SPI port when appropriate, and the SPIFLGx bit is ignored.

When CPHASE = 1, the SPI protocol permits the slave-select line to either remain asserted (LOW) or be deasserted between transferred words. This requires a program write to the SPIFLG register, setting or clearing the appropriate SPIFLG bits as needed. For example, to drive FLG1 as a slave-select, set the DS1EN bit field (bits 3–0) in the SPIFLG register to one. Clear the SPIFLG1 bit field (bit 2) in the FLAGS register to drive FLG1 low; set the SPIFLG1 bit to drive it high. As soon as this SPIFLG register write

takes effect, the FLG1 slave-select output pin becomes active (LOW). If needed, FLG1 can be cycled high and low between transfers by setting SPIFLG1 and then clearing SPIFLG1. Otherwise, FLG1 remains active (LOW) between transfers. This feature can be used to emulate the DCPH functionality supported by the ADSP-21161 processor. For more information on FLG pins, see the *ADSP-2126x SHARC Processor Core Manual*.

When CPHASE = 1 and DMA is used for the SPI transfer, it is not possible to deselect the flag pins between transfers.

When CPHASE = 0, the SPI protocol requires that the slave-select be deasserted between transferred words. In this case, the SPI hardware controls the pins. For example, to configure FLG1 as a slave-select pin, it is only necessary to set the DS1EN bit in the SPIFLG register. Writing to the SPIFLG1 bit is not required, because the SPI hardware automatically drives the FLG1 pin.

Note that the flag outputs behave as a slave-select output only if the SPI port is enabled as a master. Otherwise, none of the bits in the SPIFLG register have any effect and the FLG pins are controlled by the FLAGS register.

Use of DSxEN Bits in SPIFLG for Multiple Slave SPI Systems

The DSxEN bits in the SPIFLG register are used in a multiple slave SPI environment. For example, if there are five SPI devices in the system with an ADSP-2126x processor master, then the master ADSP-2126x processor can support the SPI mode transactions across all four other devices. This configuration requires that only one ADSP-2126x processor be a master. For example, assume that SPI0 is the master. The four flag pins on the ADSP-2126x processor master can be connected to each of the slave SPI device's $\overline{\text{SPIDS}}$ pins. In this configuration, the DSxEN bits in the SPIFLG register can be used in three ways.

In cases 1 and 2, the processor acts as the master, and the four SPI microcontrollers/peripherals act as slaves. In this configuration, the ADSP-2126x processor can:

1. Transmit to all four SPI devices at the same time in Broadcast mode. Here, all the DSxEN bits are set.
2. Receive and transmit from one SPI device by enabling only one slave SPI device at a time.

In case 3, all five devices connected via SPI ports can be ADSP-2126x processors.

3. If all the slaves are also ADSP-2126x processors, then the requestor can receive data from only one ADSP-2126x processor (enable this by setting the DMISO bit in the other slave processors) at a time and transmit broadcast data to all four at the same time. This DMISO feature may be available in some other microcontrollers. Therefore, it would be possible to use the DMISO feature with any other SPI device which includes this functionality.

Figure 5-8 shows one ADSP-2126x processor as a master with three ADSP-2126x processors (or other SPI-compatible devices) as slaves.

SPI Device Select Input Pin

The behavior of the $\overline{\text{SPIDS}}$ input depends on the configuration of the SPI. If the SPI is a slave, $\overline{\text{SPIDS}}$ acts as the slave-select input. When enabled as a master, $\overline{\text{SPIDS}}$ can serve as an error-detection input for the SPI in a multimaster environment. The ISSEN bit (bit 4) in the SPICTL register enables the SPI master mode feature. When ISSEN=1, the $\overline{\text{SPIDS}}$ input is the master mode error input; otherwise, $\overline{\text{SPIDS}}$ is ignored. The state of these input pins can be observed in the flag I/O module's data register.

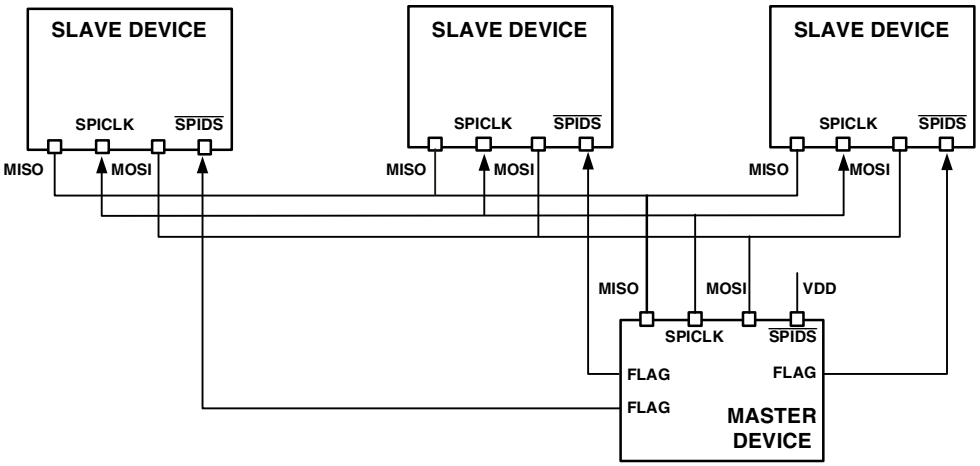


Figure 5-8. Single Master, Multiple Slave Configuration

SPI Status Register (SPISTAT)

The SPI Status register (SPISTAT) is used to detect when an SPI transfer is complete or if transmission/reception errors occur. The SPISTAT register can be read at any time.

Some of the bits in the SPISTAT register are read-only (RO) and cannot be cleared. The remainder of the bits can be read, but can also be cleared by a write one-to-clear (W1C-type) operation. Bits that provide information about the SPI are also read-only; these bits get set and cleared by the hardware. Bits that are W1C-type are set when an error condition occurs (see Table 5-8 on page 5-45); these bits are set by hardware and must be cleared by software. To clear a W1C-type bit, the program must write a one to the desired bit position of the SPISTAT register. For example, if the TUNF bit is set, the program must write a one to bit 2 of SPISTAT to clear the TUNF error condition. This allows the program to read the status register without changing its value.

Write one-to-clear (W1C-type) bits can only be cleared by writing 1 to them. Writing zero does not clear (or have any effect on) a W1C-type bit. Table 5-8 provides the bit descriptions for the SPISTAT register.

Table 5-8. SPISTAT Register Bits

Bit	Name	Function	Type	Default
0	SPIF	**SPIRCV bits in SPICTL.** Set to 1 when the appropriate shift-register has completed shifting in or out data.	RO	1
1	MME	**Multimaster Error or Mode-fault Error.** Set when the processor is configured as the SPI master and another device tries to become the master by driving the $\overline{\text{SPIDS}}$ signal low. See "Mode Fault Error (MME)" on page 5-53.	W1C	0
2	TUNF	**Transmission Error (Underflow).** Set when a transmission occurred with no new data in the TXSPI register. See "Transmission Error Bit (TUNF)" on page 5-55.	W1C	0
3	TXS	**Transmit Data Buffer Status.** Indicates the TXSPI data buffer status. 0 = Empty 1 = Full	RO	0
4	ROVF	**Reception Error (Overflow).** Set when data is received and the receive buffer is full. 1 = New data received with full RXSPI register. See "Reception Error Bit (ROVF)" on page 5-55.	W1C	0
5	RXS	**Receive Data Buffer Status.** Indicates the RXSPI data buffer status. 0 = Empty 1 = Full	RO	0

Table 5-8. SPISTAT Register Bits (Cont'd)

Bit	Name	Function	Type	Default
6	TXCOL	**Transmission Collision Error.** The TXCOL flag is set in the SPISTAT register when a write to the TXSPI register coincides with the load of the shift register. See "Transmit Collision Error Bit (TXCOL)" on page 5-55	W1C	0
31–7	Reserved			

The transmit buffer is full after data is written to it and is empty when a transfer begins and the transmit value loads into the Shift register. The receive buffer is full at the end of a transfer when the Shift register value is loaded into the receive buffer. It is empty when the receive buffer is read.

The SPI status also depends on the PACKEN bit in the SPICTL register. If packing is enabled, then the receive buffer status is set to full only after two transfers from the Shift register.

Buffering and Transmit/Receive Registers

The TXSPI and RXSPI registers are 32-bit memory-mapped registers that hold SPI data for transmit and receive operations.

Check the buffer status before reading from or writing to these registers because the core does not hang when it attempts to read from an empty buffer or write to a full buffer. When the core writes to a full buffer, the data in that buffer is overwritten and the SPI begins transmitting the new data. Invalid data is obtained when the core reads from an empty buffer.

SPI Transmit Data Buffer Register (TXSPI)

The Transmit Data Buffer register (TXSPI) is a 32-bit read-write (RW) register. 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 Transmit Shift Data (SFDR) register. A normal core read of TXSPI can be done at any time and does not interfere with, or initiate, SPI transfers.

With DMA enabled for transmit operations, the IOP loads data into this register. Core writes to TXSPI should not be made to prevent corrupting the DMA data to be transmitted.

With DMA enabled for receive operations, the contents of the TXSPI register are repeatedly transmitted. A normal core write to TXSPI is permitted in this mode, and this data is transmitted. If the Send Zeroes Control bit (SENDZ) is set, TXSPI resets once the data is transferred from TX to TXSR.

If multiple writes to TXSPI occur while a transfer is already in progress, only the last data written is transmitted. None of the intermediate values written to TXSPI are transmitted. Multiple writes to TXSPI are possible but not recommended. To avoid overwriting data, be sure to poll the TXS bit before writing to TXSPI.

To prevent transmit collision errors, ensure that the program writes to the TXSPI register before the load to the Shift register occurs by writing to TXSPI whenever TXS is cleared. Programs should refrain from writing to TXSPI when TXS is set. For slave mode, data should exist in TXSPI before the first SPI clock edge (or negative edge of device select) occurs.

The TXCOL bit can be set when there is a TUNF condition and there are attempts to write to the TXSPI register. In this case, TXS is not set and the program wants to send new data. To ensure that TXSPI is written into before the next load to a Shift register occurs, write to the TXSPI register as soon as the SPIF bit (bit 0 in the SPISTAT register) goes from one to zero.

SPI Receive Data Buffer Register (RXSPI)

The Receive Data Buffer register (RXSPI) is a 32-bit read-only (RO) register that is accessible by both the software and DMA. At the end of a data transfer, the data in the Receive Shift register (RXSR) loads into the RXSPI register. During a DMA receive operation, the data in the RXSPI register is automatically read by the DMA. A shadow register for the receive data buffer, RXSPI, supports software debugging functions. See "SPI Receive Data Buffer Shadow Register (RXSPI_SHADOW)" on page 5-53.

DMA Registers

The following registers configure and manage SPI DMA functions:

- SPIDMAC
- IISPI
- IMSPI
- CSPI
- CPSPI

SPI DMA Configuration (SPIDMAC) Register

The SPI DMA Configuration Register contains the control bits for SPI DMA transfers. Table 5-9 provides the bit descriptions for the SPIDMAC register.

The SPIMME, SPIUNF, and SPIOVF bits are sticky; these bits remain set even if the corresponding SPISTAT bits (MME, TUNF, and ROVF) are cleared. To clear these bits, clear corresponding bits in the SPISTAT, register then configure a new DMA.

Table 5-9. SPIDMAC Register Bits

Bit(s)	Name	Function	Type	Default
0	SPIDEN	**DMA Enable.** Enables if set (= 1) or disables if cleared (= 0) DMA for the SPI port.	Control	0
1	SPIRCV	**DMA Direction.** When set, the IOP empties the RXSPI buffer, when cleared, the IOP fills the TXSPI buffer. 0 = SPI Transmit DMA (Memory read) 1 = SPI Receive DMA (Memory write)	Control	0
2	INTEN	**Enable DMA Interrupt**. Enables if set (= 1) or disables if cleared (= 0) an interrupt upon completion of the DMA transfer.	Control	0
3	Reserved			
4	SPICHEN	**SPI DMA Chaining Enable**. Enables if set (=1) or disables if cleared (= 0) DMA chaining.	Control	0
6:5	Reserved			
7	FIFOFLSH	**DMA FIFO Flush.** Clears the four-deep FIFO and FIFO status bits if set (= 1).	Control	0
8	INTERR	**Enable Interrupt on Error.** Enables if set (= 1) or disables if cleared (= 0) an interrupt when an error in the transmission occurs.	Control	0
9	SPIOVF	**Receive Overflow Error (SPIRCV=1).** Set when SPIRCV = 1 and data is received with the receive buffer full (1 = error data received with receive data buffer RXSPI full in receive mode DMA).	Status	0
10	SPIUNF	**SPI Transmit Underrun Error.** Set when SPIRCV = 0 and the SPI transmits without any new data in the transmit buffer TXSPI.	Status	0

Table 5-9. SPIDMAC Register Bits (Cont'd)

Bit(s)	Name	Function	Type	Default
11	SPIMME	**SPI Multimaster Error**. Set when MME is set in the SPISTAT register and DMA is enabled.	Status	0
13:12	SPISx	**DMA FIFO Status 0.** Indicates the status of the DMA FIFO as follows: 00 = FIFO empty 11 = FIFO full 10 = FIFO partially full 01 = Reserved	Status	
14	SPIERRS	**DMA Error Status**. Set if any of the following error bits get set: SPIOVF, SPIUNF, or SPIMME	Status	0
15	SPIDMAS	**DMA Transfer Status.** Indicates the status of the DMA transfer as follows: 1 = DMA in progress, 0 = DMA idle	Status	0
16	SPICHS	**DMA Chain Loading Status.** Indicates the status of the DMA chain loading as follows: 1 = DMA chain pointer loading in progress, 0 = Chain idle	Status	0
31:17	Reserved			

SPI DMA Internal Index Register (IISPI)

This 19-bit register contains the address where the IOP transfers data to or from. Initially, this register holds the first address of the source or destination buffer, and then as the DMA progresses, this register is modified by the value in `IMSPI`.

SPI DMA Address Modifier Register (IMSPI)

This 16-bit register contains the DMA address modifier. After the IOP transfers each word between memory and the `TXSPI`/`RXSPI` register, this value is used to increment the internal index, `IISPI`.

SPI DMA Word Count Register (CSPI)

This 16-bit register contains the number of DMA words to be transferred. When this register decrements from one to zero, the DMA is complete, and an interrupt may be triggered.

To prematurely end a DMA transfer, software should write the value one to the Count register so that it will decrement to zero. Writing a value of zero causes the count to decrement to a negative number, and this is not advised.

SPI DMA Chain Pointer Register (CPSPI)

This register contains the address of the Transfer Control Block (TCB) in memory when DMA chaining is enabled. This register's address is 0x1083 and its reset value is undefined. For more information, see "Transfer Control Block Chain Loading (TCB)" on page 2-12.

Table 5-10 provides the bit descriptions for the `CPSPI` register.

Table 5-10. CPSPI Register Bits

Bits	Function	Default
18:0	**Next Chain Pointer Address.** The address of the next transfer control block (TCB) in memory.	0
19	**PCI – Program Controlled Interrupt.** Affects interrupt functionality when DMA chaining is enabled. Setting this bit (= 1) causes an interrupt to occur after each DMA in the chain completes. Clearing this bit (= 0) causes an interrupt to occur only after the final DMA transfer in the chain is completed.	0

Shift Registers

The processor core contains two separate 32-bit shift registers—one for reception (RXSR) and one for transmission (TXSR).

Receive Shift Register (RXSR)

The RXSR register is clocked on the sampling edge of the SPICLK clock. The active edge is the opposite edge from the sampling edge. The RXSR register behaves the same way whether the device is in Slave or Master mode.

The register is configured by the MSBF and WL bits of the SPICTL register. The MSBF bit indicates the data format (LSB-first or MSB-first) and selects the direction of the shift. The WL bit indicates the length of the transfer—8 bits if WL = 00, 16 bits if WL = 01, and 32 bits if WL = 10. For more information, see "SPI Control Register (SPICTL)" on page A-44.

Transmit Shift Register (TXSR)

The TXSR register is clocked on the active or shifting edge. The active edge is the opposite edge from the sampling edge. The TXSR register can be shifted right or left, depending on the direction of the data flow. This register can also be loaded from the TXSPI register with data that is to be transmitted.

This register behaves the same way whether the device is in Slave or Master mode. The TXSR register contains 32 shift cells.

Each Shift register is configured by the MSBF and WL bits of the SPICTL register. The MSBF bit indicates the data format (LSB-first or MSB-first) and selects the direction of the shift. The WL bit indicates the length of the transfer—8 bits if WL = 00, 16 bits if WL = 01, and 32 bits if WL = 10. When WL = 00, the upper 24 MSBs of the register loads with zeroes after a write to the TXSPI register. For more information, see "SPI Control Register (SPICTL)" on page A-44.

SPI Receive Data Buffer Shadow Register (RXSPI_SHADOW)

Use the RXSPI_SHADOW register for the receive data buffer (RXSPI) to debug software. The RXSPI_SHADOW register resides at a different address from RXSPI, but its contents are identical to the RXSPI. When RXSPI is read via the software, the RXS bit clears and an SPI transfer may be initiated (if TIMOD = 00). No such hardware action occurs when the shadow register is read. The RXSPI_SHADOW register is a read-only (RO) register accessible only by the software and not the DMA. For more information, see "SPI Receive Buffer Register (RXSPI)" on page A-45.

Error Signals and Flags

This section describes the error signals and flags that determine the cause of transmission errors for an SPI port. The bits MME, TUNF and ROVF are set in the SPISTAT register when a transmission error occurs. Corresponding bits (SPIMME, SPIUNF and SPIOVF) in the SPIDMAC register are set when an error occurs during a DMA transfer. These sticky bits generate an SPI interrupt when any one of them are set.

- See "SPI Status Register (SPISTAT)" on page 5-44 for more information about the SPISTAT register bits.
- See "SPI DMA Configuration (SPIDMAC) Register" on page 5-48 for more information about the SPIDMAC register bits.

Mode Fault Error (MME)

The MME bit is set in the SPISTAT register when the $\overline{\text{SPIDS}}$ 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.

To enable this feature, set the ISSEN bit in the SPICTL register. As soon as this 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 write 1-to-clear (W1C-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 $\overline{\text{SPIDS}}$ 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 flag I/O module registers. This may cause contention on the slave-select lines if these lines are still being driven by the ADSP-2126x processor. In order to ensure that the slave-select output drivers are disabled once a MME error occurs, the program must configure these pins as inputs by clearing (= 0) the FLG0O, FLG1O, FLG2O, and FLG3O bits in the FLAGS register prior to configuring the SPI port. See also the FLAGs value register description in the *ADSP-2126x SHARC Processor Core Manual*.

Transmission Error Bit (TUNF)

The TUNF bit is set in the SPISTAT register 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. The TUNF bit is cleared by a W1C-type software operation.

Reception Error Bit (ROVF)

The ROVF flag is set in the SPISTAT register 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 flag is cleared by a W1C-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.

Transmit Collision Error Bit (TXCOL)

The TXCOL flag is set in the SPISTAT register 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 W1C-type software operation.

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.

SPI Programming Examples

The following three programming examples are for the ADSP-21262 processor. The example shown in Listing 5-1 transmits a buffer of data from the SPI port in Master mode using DMA. In this example, the I/O Processor (IOP) automatically moves data from internal memory to the SPI's four-deep DMA-FIFO.

The second example, shown in Listing 5-2, also transmits a buffer, but the transfer is core-driven using interrupts. In this example, only the SPI's one-deep transmit buffer (`TXSPI`) is serviced by the core and the four-deep DMA-FIFO is not used. The core supplies the SPI port with data in a short loop which causes the core to hang at each write to the transmit buffer until the SPI is ready for new data.

The third example, shown in Listing 5-3, receives multiple buffers using DMA chaining. DMA chaining on the ADSP-21262 processor SPI is initialized differently than on other SHARC processors, as described in Chapter 2, I/O Processor.

Listing 5-1. SPI Master Mode Transmit DMA

```
/* SPI Control Registers */
#define SPICTL  (0x1000)
#define SPIFLG  (0x1001)
#define SPIBAUD (0x1005)
#define TXSPI   (0x1003)

/*SPICTL bits*/
#define TIMOD1  (0x0001)  /* Use DMA for transfers */
#define DMISO   (0x0020)  /* Disable MISO pin */
#define WL32    (0x0100)  /* SPI Word Length = 32 */
#define SPIMS      (0x1000)  /* SPI Master if 1, Slave if 0 */
#define SPIEN   (0x4000)  /* SPI Port Enable */
```

```
/*SPIFLG bits */
#define DS0EN   (0x0001)  /* use FLG0 as SPI device-select*/

/* Default Buffer Length */
#define BUFSIZE 10

.SECTION/DM seg_dmda;
/* Transmit Buffer */
.var tx_buf[BUFSIZE] =   0x11111111,
                         0x22222222,
                         0x33333333,
                         0x44444444,
                         0x55555555,
                         0x66666666,
                         0x77777777,
                         0x88888888,
                         0x99999999,
                         0xAAAAAAAA;

/* Main code section */
.global _main;
.SECTION/PM seg_pmco;
_main:
/* Init SPI MASTER TX */
r0=0;
dm(SPICTL) = r0;
dm(SPIFLG) = r0;


/* set the SPI baud rate to CCLK/4x64 (781.25KHz @ 200MHz)*/
ustat3 = 0x64;
dm(SPIBAUD) = ustat3;

/* Set up DAG registers */
```

```
i4 = tx_buf;
m4 = 1;

ustat3 = DMISO|  /* Disable MISO on transfers */
         WL32|   /* 32-bit words */
         SPIMS|  /* Master mode (internal SPICLK) */
         SPIEN|  /* Enable SPI port */
         TIMOD1; /* Initialize SPI port to begin
                    transmitting when DMA is enabled */
dm(SPICTL) = ustat3;


/* Set up loop to transmit data */
lcntr = LENGTH(tx_buf), do (pc,2) until lce;
/* Retrieve data using DAG1 and send TX via SPI */
r0 = dm(i4,m4);
dm(TXSPI) = r0;
_main.end:  jump (pc,0);
```

Listing 5-2. Core Driven Interrupt SPI Transfer

```
/* SPI Control Registers                                  */
#define SPICTL  (0x1000)  /* SPI Control Register         */
#define SPIFLG  (0x1001)  /* SPI Flag register            */
#define SPIBAUD (0x1005)  /* SPI baud setup register      */

/* SPI DMA Registers                                      */
#define IISPI   (0x1080)  /* Internal DMA address         */
#define IMSPI   (0x1081)  /* Internal DMA access modifier */
#define CSPI    (0x1082)  /* Number of words to transfers */
#define CPSPI   (0x1083)  /* Points to next DMA parameters*/
#define SPIDMAC (0x1084)  /* SPI DMA control register     */

/*SPICTL bits                                             */
```

```
#define TIMOD2  (0x0002)  /* Use DMA for transfers        */
#define DMISO   (0x0020)  /* Disable MISO pin             */
#define WL32    (0x0100)  /* SPI Word Length = 32         */
#define SPIMS   (0x1000)  /* SPI Master if 1, Slave if 0  */
#define SPIEN   (0x4000)  /* SPI Port Enable              */

/*SPIFLG bits                                             */
#define DS0EN   (0x0001)  /* use FLG0 as SPI device-select*/

/*SPIDMAC bits                                            */
#define SPIDEN  (0x0001)  /* enable DMA on the SPI port   */

/* Default buffer size */
#define BUFSIZE 0x100
/*=========================================================*/
/* Source data to be transmitted via SPI DMA              */
.section/dm seg_dmda;
.var src_buf[BUFSIZE] = "source.dat";

/* Application code */
.global _main;
.segment/pm seg_pmco;
_main:

/* set the SPI baud rate to CCLK/4x64 (781.25KHz @ 200MHz)*/
ustat3 = 0x64;
dm(SPIBAUD) = ustat3;

/* Init SPI MASTER TX DMA */
r0 = 0;
dm(SPICTL) = r0;
dm(SPIFLG) = r0;
dm(SPIDMAC) = r0;
```

```
r0 = DSOEN;
dm(SPIFLG) = r0;   /*use flag0 as spi device select */

ustat3 = src_buf;   dm(IISPI) = ustat3;  /* point to 'src_buf'*/
ustat3 = LENGTH(src_buf); dm(CSPI) = ustat3; /* count = 256  */
ustat3 = 1;         dm(IMSPI) = ustat3;  /* step size = 1 */

ustat3 = DMISO|  /* Disable MISO on transfers */
         WL32|   /* 32-bit words */
         SPIMS|     /* Master mode (internal SPICLK) */
         SPIEN|  /* Enable SPI port */
         TIMOD2; /* Initialize SPI port to begin
                    transmitting when DMA is enabled */
dm(SPICTL) = ustat3;

ustat3 = SPIDEN;  dm(SPIDMAC) = ustat3;  /* begin DMA         */
/*=========================================================*/

_main.end: jump (pc,0);
```

Listing 5-3. SPI DMA Chaining Example

```
/* SPI Control Registers                                  */
#define SPICTL  (0x1000)  /* SPI Control Register         */
#define SPIFLG  (0x1001)  /* SPI Flag register            */
#define SPIBAUD (0x1005)  /* SPI baud setup register      */

/* SPI DMA Registers                                      */
#define IISPI   (0x1080)  /* Internal DMA address         */
#define IMSPI   (0x1081)  /* Internal DMA access modifier */
#define CSPI    (0x1082)  /* Number of words to transfers */
#define CPSPI   (0x1083)  /* Points to next DMA parameters*/
#define SPIDMAC (0x1084)  /* SPI DMA control register     */
```

```
/*SPIFLG bits                                                */
#define DSOEN   (0x0001) /* enable SPI device select 0    */
#define SPIFLG0 (0x0100) /* manually set SPIFLG0 state    */
#define SPIFLG1 (0x0200) /* manually set SPIFLG1 state    */
#define SPIFLG2 (0x0400) /* manually set SPIFLG2 state    */
#define SPIFLG3 (0x0800) /* manually set SPIFLG3 state    */

/*SPIDMAC bits                                               */
#define SPIDEN  (0x0001) /* enable DMA on the SPI port    */
#define SPIRCV  (0x0002) /* set to have DMA receive       */
#define SPICHEN (0x0010) /* set to enable DMA chaining    */

/*SPICTL bits                                                */
#define TIMOD2  (0x0002) /* Use DMA for transfers         */
#define SENDZ   (0x0004) /* when TXSPI empty, MOSI sends 0 */
#define WL32    (0x0100) /* SPI Word Length = 32          */
#define SPIMS      (0x1000) /* SPI Master if 1, Slave if 0   */
#define SPIEN   (0x4000) /* SPI Port Enable               */
#define CLKPL   (0x0800) /* if 1, rising edge samples data */
#define CPHASE  (0x0400) /* if 1, data's sampled on second */
                         /*  (middle) edge of SPICLK cycle*/

/*============================================================*/
.section/dm seg_dmda;

/* Destinations for incoming data                            */
.var dest_bufC[8];
.var dest_bufB[8];
.var dest_bufA[8];

/* Transfer Control Blocks (TCB's)                           */
.var first_tcb[] =
        (0x7FFFF&second_tcb + 3), /* for CPSPI (next tcb) */
```

```
                LENGTH(dest_bufB), /* for CSPI (next count) */
                    1,           /* for IMSPI (next modify) */
                    dest_bufB;    /* for IISPI (next index) */

.var second_tcb[] = 0,          /* null CPSPI ends chain    */
                LENGTH(dest_bufC), /* count for final DMA   */
                     1,         /* IM for final DMA         */
                     dest_bufC; /* II for final DMA         */
/* NOTE: Chain Pointer registers must point to the LAST   */
/*        location in the TCB, "tcb + 3".    */

/*Main code section */
.global _main;
.section/pm seg_pmco;
_main:
/* clear SPI settings */
r0 = 0;
dm(SPICTL) = r0;
dm(SPIFLG) = r0;
dm(SPIDMAC) = r0;
/* setup first DMA in chain */
ustat3 = 8;          dm(CSPI) = ustat3; /* count = 8 words  */
ustat3 = 1;          dm(IMSPI) = ustat3; /* step size = 1   */
ustat3 = dest_bufA; dm(IISPI) = ustat3; /* point to dest_bufA */

/* set the SPI baud rate to CCLK/4x64 (781.25KHz @ 200MHz)*/
ustat3 = 0x64;
dm(SPIBAUD) = ustat3;

/* configure processor's SPI slave-select signals     */
ustat3 = DSOEN|  /*enable SPI slave device select zero */
        SPIFLG3|SPIFLG2|SPIFLG1;/* Set SPIFLG0 low to  */
dm(SPIFLG) = ustat3;   /*select SPI slave on FLG0 pin  */
```

```
/* configure SPI port to power-on settings */
ustat3 = CPHASE| /* sample MISO on second edge of SPICLK */
         CLKPL|  /* sampling edge of SPICLK is rising  */
         WL32|   /* 32-bit words  */
         SPIMS|  /* Master mode (internal SPICLK) */
         SPIEN|  /* Enable SPI port  */
         SENDZ|  /* when TXSPI empty, MOSI sends zeros  */
         TIMOD2; /* Start SPICLK when DMA is enabled    */
dm(SPICTL) = ustat3;

/*configure SPI for chained receive DMA operation         */
ustat3 = SPIRCV|  /* DMA direction = receive               */
        SPICHEN| /* enable DMA chaining                   */
        SPIDEN;  /* enabling DMA initiates the transfer   */
dm(SPIDMAC) = ustat3;

/* 1st DMA starts when a valid address is written to CPSPI*/
ustat3 = (0x7FFFF&(first_tcb+3));
dm(CPSPI) = ustat3; /* point to tcb_A   */
_main.end: jump(pc,0);
/*=============================================================*/
```

6 INPUT DATA PORT

The signal routing unit (SRU) provides paths among both on-chip and off-chip peripherals. To make this feature effective in a real-world system, a low overhead method of making data from various serial formats parallel and routing them back to the main core memory is needed. The Input Data Port (IDP) provides this mechanism for a large number of asynchronous channels.

This chapter describes how data is routed into the core's memory space. Figure 6-1 provides a graphical overview of the Input Data Port architecture. Notice that each channel is independent and each contains a separate clock and frame sync input.

Channels 0 through 7 can accept serial data in audio format. Channel 0 can also be configured to accept parallel data. The parallel input bypasses the serial-to-parallel converter and latches up to 20 bits per clock cycle.

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 or packed together (up to four clock cycles of data). Figure 6-2 illustrates the data flow for the IDP channel 0, where either the PDAP or serial input can be selected via control bit `IDP_PDAP_EN` (bit 31 of the `IDP_PDAP_CTL` register).

The following sections describe each of the Input Data Port functions.

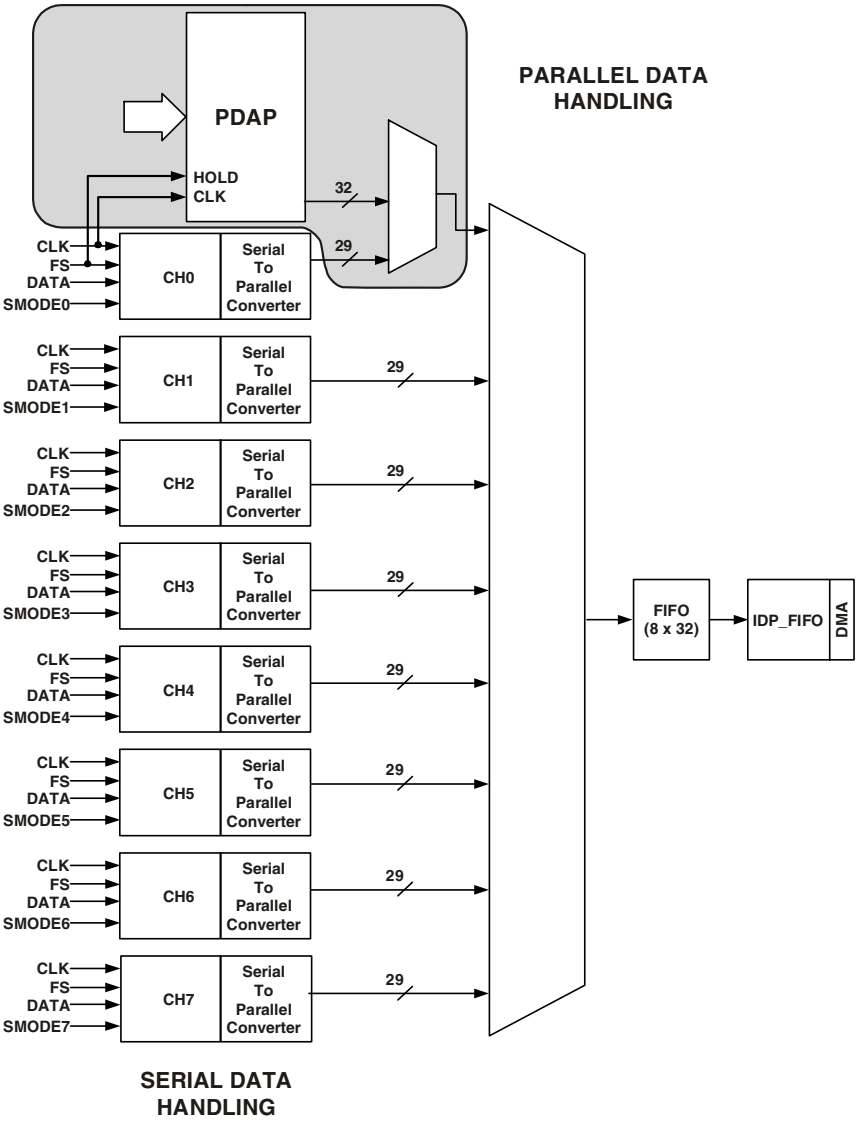


Figure 6-1. The Input Data Port

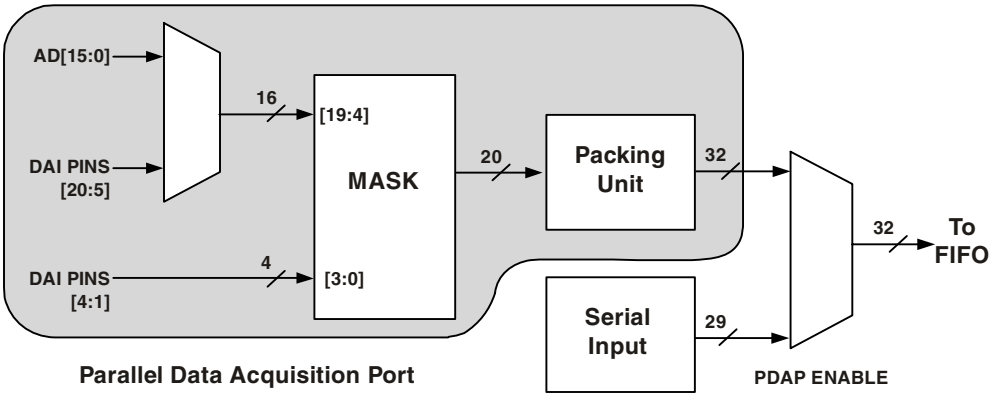


Figure 6-2. Detail of IDP Channel 0

Serial Inputs

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 Sample Pair, 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 a frame at a time).

Contained within the 32-bit word is an audio signal that is normally 24 bits wide. An additional 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. Regardless of mode, bit 3 always specifies if the data is received in the first half (left channel), or the second half (right channel) of the same frame, as shown in Figure 6-3. 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.

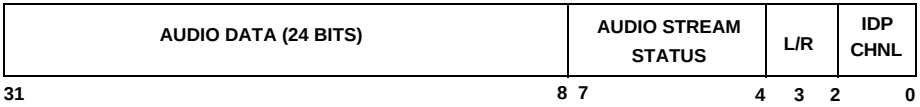


Figure 6-3. Word Format

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 assigned to `LOW` to avoid unintentional acquisition.

The framing format is selected by using `IDP_SMODEx` bits (three bits per channel) in the `IDP_CTL` register. The bits `[31:8]` of the `IDP_CTL` 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, as shown in Table 6-1.

Table 6-1. Serial Modes

Bit Field Values IDP_SMODEx	Mode
000	Left-justified Sample Pair
001	I^2S
010	Reserved
011	Reserved
100	Right-justified Sample Pair 24 bits
101	Right-justified Sample Pair 20 bits

Table 6-1. Serial Modes (Cont'd)

Bit Field Values IDP_SMODEx	Mode
110	Right-justified Sample Pair 18 bits
111	Right-justified Sample Pair 16 bits

The polarity of left-right encoding is independent of the serial mode frame sync polarity selected in IDP_SMODE for that channel (Table 6-1). Note that I^{2}S mode uses a LOW frame sync (left-right) signal to dictate the first (left) channel, and Left-justified Sample Pair 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).

Figure 6-4 shows the relationship between frame sync, serial clock, and Left-justified Sample Pair data.

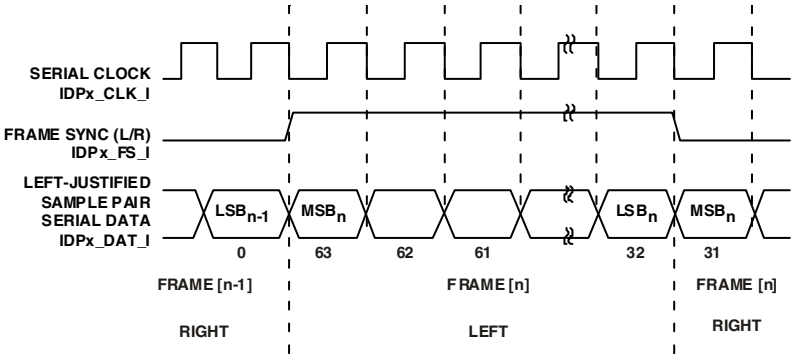


Figure 6-4. Timing in Left-justified Sample Pair Mode

Figure 6-5 shows the relationship between frame sync, serial clock, and I^{2}S data.

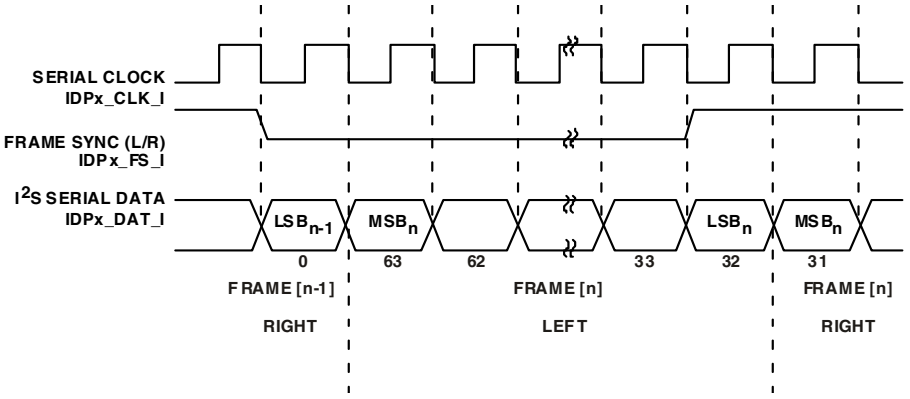


Figure 6-5. Timing in I^{2}S Mode

Parallel Data Acquisition Port (PDAP)

The input to channel 0 of the IDP is multiplexed, and may be used either in the serial mode, described in "Serial Inputs" on page 6-3, or in a direct Parallel Input mode. Serial or parallel input is selected by setting IDP_PDAP_EN bit 31 in the IDP_PDAP_CTL register. When used in parallel mode, the clock input for channel 0 is used to latch parallel sub words. 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 6-6, 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_CTL register also provides a reset bit that zeros any data that is waiting in the packing unit to be latched into the FIFO. When asserted, the IDP_PDAP_RESET bit (bit 30 in the IDP_PDAP_CTL register) causes the reset circuit to strobe, then automatically clear itself. Therefore, this bit always returns a value of zero when read. The IDP_PORT_SELECT bit (bit 26

in the IDP_PDAP_CTL register) selects between the two sets of pins that may be used as the parallel input port. When IDP_PORT_SELECT is set (= 1), the upper 16 bits are read from the AD[15:0]. When IDP_PORT_SELECT is cleared (= 0), the upper 16 bits are read from DAI_P[20:5]. Note that the four least significant bits (LSB's) of the parallel port input are not multiplexed. These input bits are always read from Digital Audio Interface (DAI) pins 4–1, as shown in Figure 6-6. The DAI_P[4:1] pins are always connected as bits 3 through 0. A sample PDAP program is located at the end of this chapter. See Listing 6-2.

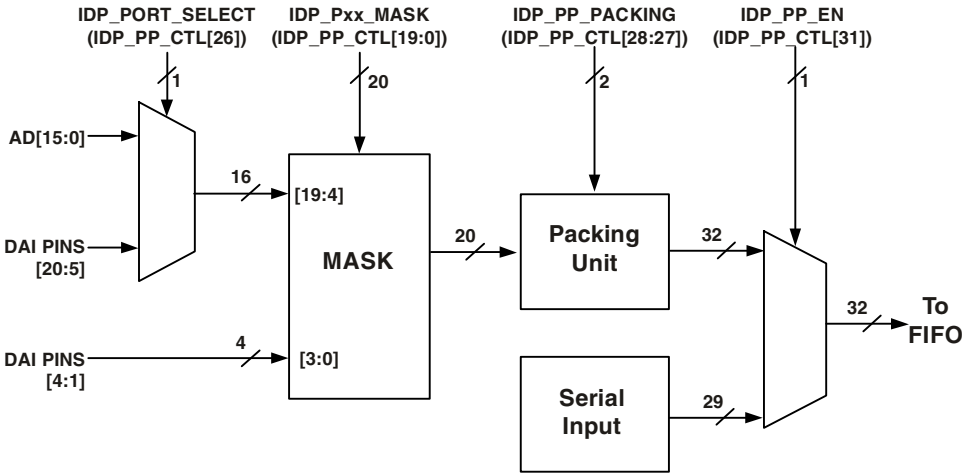


Figure 6-6. Parallel Data Acquisition Port (PDAP) Functions

Masking

The IDP_PDAP_CTL register provides 20 mask bits that allow the input from any of the 20 pins to be ignored. The mask is specified by setting the IDP_Pxx_PDAPMASK bits (bits 19–0 of the IDP_PDAP_CTL register) for the 20 parallel input signals. For each of the parallel inputs, a bit is set (= 1) to

indicate the bit is unmasked and therefore its data can be passed on to be read, or masked (= 0) so its data will not be read. After this masking process, data gets passed along to the packing unit.

Packing Unit

The Parallel Data Acquisition Port (PDAP) packing unit receives masked parallel sub words from the 20 parallel input signals and packs them into a 32-bit word. The `IDP_PDAP_PACKING` bit field (bits 28–27 of the `IDP_PDAP_CTL` register), indicates how data is to be packed. Data can be packed in any of four modes. Selection of Packing mode is made based on the application.

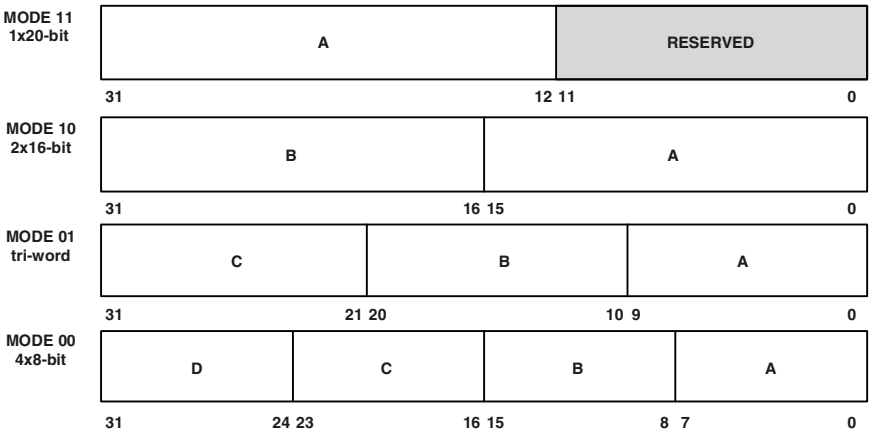


Figure 6-7. Packing Modes in IDP_PDAP_CTL

Packing Mode 11

Mode 11 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, as shown in Figure 6-7 on page 6-8.

This mode sends one 32-bit word to FIFO for each input clock cycle—the DMA transfer rate will match the PDAP input clock rate.

Packing Mode 10

On the first clock edge (cycle A), the packing unit latches parallel data up to 16 bits wide (bits 19–4 of the parallel input) and places it in bits 15–0 (the lower half of the word), then waits for the second clock edge (cycle B). On the second clock edge (cycle B), the packing unit takes the same set of inputs and places the word into bits 31–16 (the upper half of the word).

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.

Packing Mode 01

Mode 01 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 Mode 00

Mode 00 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.

Clocking Edge Selection

Notice that in all four packing modes described, data is read on a clock edge, but the specific edge used (rising or falling) is not indicated. Clock edge selection is configurable using the IDP_PDAP_CLKEDGE bit (bit 29 of the IDP_PDAP_CTL register). Setting this bit (= 1) causes the data to be latched on the falling edge. Clearing this bit (= 0) causes data to be latched on the rising edge (default).

Hold Input

A synchronous clock enable can be passed from any DAI pin to the PDAP packing unit. This signal is called PDAP_HOLD.

The PDAP_HOLD signal is actually the same physical internal signal as the frame sync for IDP channel 0. Its functionality is determined by the PDAP Enable bit (IDP_PDAP_EN).

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 6-8 shows the affect of the hold input (B) for four 8-bit words in Packing Mode 00, and Figure 6-9 shows the affect of the hold input (B) for two 16-bit words in Packing Mode 10.

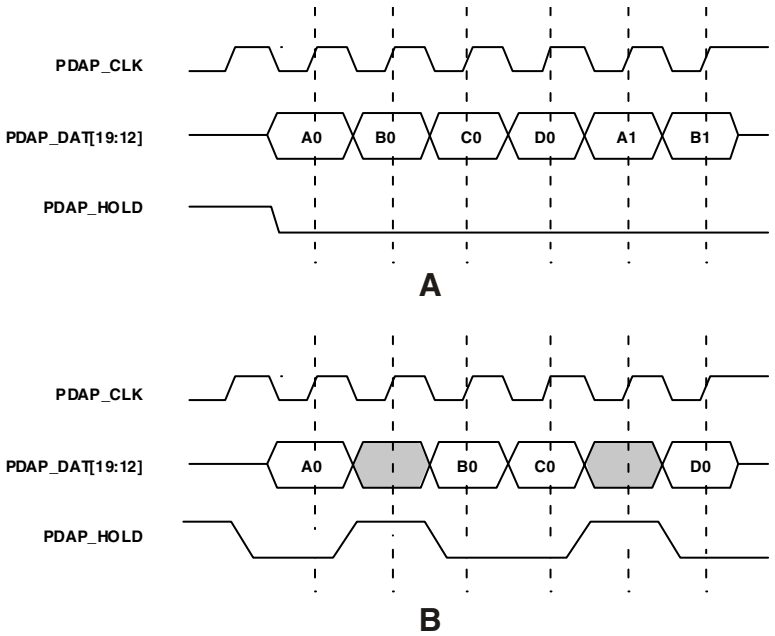


Figure 6-8. Hold Timing for Four 8-bit Words to 32 bits (Mode 00)

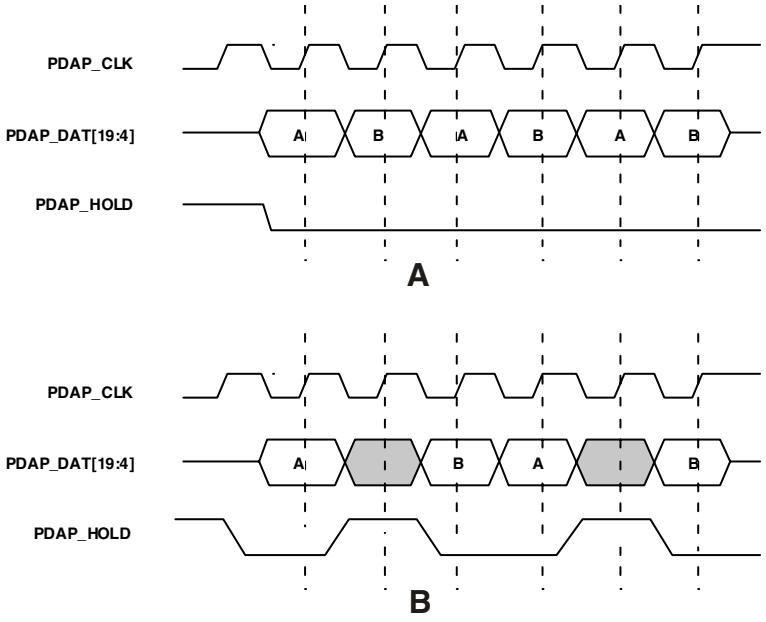


Figure 6-9. Hold Timing for Two 16-bit Words to 32 bits (Mode 10)

PDAP Strobe

Whenever the PDAP packing unit receives the number of sub words corresponding to its select mode, it asserts the PDAP Output Strobe signal. This signal can be routed through the SRU using the `MISC` unit to any of the DAI pins. See "SRU Connection Groups" on page 7-15 for more information.

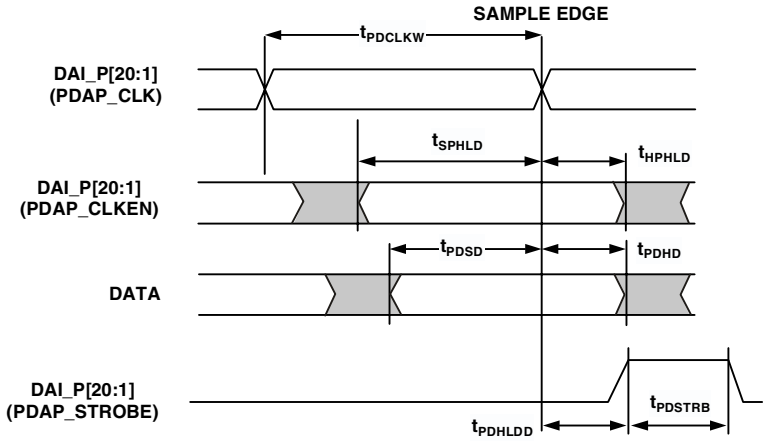


Figure 6-10. PDAP Timing

FIFO Control and Status

Several bits can be used to control and monitor FIFO operations:

- **IDP Enable.** The IDP_ENABLE bit (bit 7 of the IDP_CTL register) enables the IDP.
- **IDP Buffer Hang Disable.** The IDP_BHD bit (bit 4 in the IDP_CTL register) determines whether or not the core hangs on reads when the FIFO is empty.
- **Number of Samples in FIFO.** The IDP_FIFOSZ bits (bits 31–28 in the DAI_STAT register) monitors the number of valid data words in the FIFO.
- **FIFO Overflow Status.** The IDP_FIFO_OVER bit (bit 25 in the DAI_STAT register) monitors overflow error conditions in the FIFO.
- **FIFO Overflow Clear bit.** The IDP_CLROVR bit (bit 6 of the IDP_CTL register) clears an indicated FIFO overflow error.

The IDP is enabled through the IDP_ENABLE bit. When this bit is set (= 1), the IDP is enabled. When this bit is cleared (= 0), the IDP is disabled, and data can not come to the IDP_FIFO register from the IDP channels. When this bit transitions from 1 to 0, all data in the IDP FIFO is cleared.

The IDP_BHD bit 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.

The IDP_FIFOSZ bits track the number of words in the FIFO. This four-bit field identifies the number of valid data samples in the IDP FIFO.

The IDP_FIFO_OVER bit provides IDP FIFO overflow status information. This bit is set (= 1), whenever an overflow occurs. When this bit is cleared (= 0), it indicates there is no overflow condition. This read-only bit is a *sticky* bit, which does not automatically reset to 0 when it is no longer in overflow condition. This bit must be reset manually, using the IDP_CLROVR bit in the IDP_CTL register. Writing one to this bit clears the overflow condition in the DAI_STAT register. Since IDP_CLROVR is a write-only bit, it always returns LOW when read.

FIFO to Memory 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. When more than one channel has data ready, the channels access the FIFO with fixed priority, from low to high channel number (that is, channel 0 is the highest priority and channel 7 is the lowest priority).

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 by reading the memory-mapped register, IDP_FIFO. The output of the FIFO is held in the (read-only) IDP_FIFO register. 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.

  This method of moving data from the IDP FIFO is described in "Interrupt-Driven Transfers" on page 6-15.

- 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.

  This method of moving data from the IDP FIFO is described in "DMA Transfers" on page 6-18.

Interrupt-Driven Transfers

The output of the FIFO can be directly fetched by reading from the IDP_FIFO register. The IDP_FIFO register is used only to read and remove the top sample from the FIFO, which is eight locations deep.

As data is read from the IDP_FIFO register, it is removed from the FIFO and new data is copied into the IDP_FIFO register. The contents of the IDP_NSET bits (bits 3–0 in the IDP_CTL register) represent a threshold number of entries (*N*) in the FIFO. When the FIFO fills to a point where it has more than *N* words (data in FIFO exceeds the value set in the IDP_NSET bit field, bits 3–0 of IDP_CTL register), a DAI interrupt is gener-

ated. This DAI interrupt corresponds to the IDP_FIFO_GTN_INT bit, the eighth interrupt in DAI_IRPTL_L or DAI_IRPTL_H. The core can use this interrupt to detect when data needs to be read.

Starting an Interrupt-Driven Transfer

To start an interrupt-driven transfer:

1. Clear and halt FIFO by setting (= 1) and clearing (= 0) the IDP_ENABLE bit (bit 7 in the IDP_CTL register).
2. Set the required values for:
   - IDP_SMODEx bits in the IDP_CTL register to specify the frame sync format for the serial inputs (I^2S, Left-justified Sample Pair, or Right-justified Sample Pair Mode).
   - IDP_Pxx_PDAPMASK bits in the IDP_PDAP_CTL register to specify the input mask, if the PDAP is used.
   - IDP_PORT_SELECT bits in the IDP_PDAP_CTL register to specify input from the DAI pins or the Parallel Port pins, if the PDAP is used.
   - IDP_PDAP_CLKEDGE bit (bit 29) in the IDP_PDAP_CTL register to specify if data is latched on the rising or falling clock edge, if the PDAP is used.
3. Keep the clock and frame sync inputs of all serial inputs and/or PDAP connected to LOW. Use the SRU_CLK1, SRU_CLK2, SRU_FS1, and SRU_FS2 registers to specify these inputs.
4. Connect all of the inputs to the IDP by writing to the SRU_DAT3, SRU_DAT4, SRU_FS1, SRU_FS2, SRU_CLK1 and SRU_CLK2 registers. Connect the clock and frame sync of any unused ports to LOW.
5. Set the desired value for *N_SET* variable (the IDP_NSET bits, 3–0, in the IDP_CTL register).

6. Set the IDP_FIFO_GTN_INT bit (bit 8 of the DAI_IRPTL_RE register) to HIGH and set the corresponding bit in the DAI_IRPTL_FE register to LOW to unmask the interrupt. Set bit 8 of the DAI_IRPTL_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 in step 5.
7. Enable the PDAP by setting IDP_PDAP_EN (bit 31 in the IDP_PDAP_CTL register), if required.
8. Enable the IDP by setting IDP_ENABLE bit (bit 7 in the IDP_CTL register).

Do *not* set the IDP_DMA_EN bit (bit 5 of the IDP_CTL register).

Interrupt-Driven Transfer Notes

The following items provide general information about interrupt driven transfers.

- 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 number of data samples in the FIFO at any time is reflected in the IDP_FIFOSZ bit field (bits 31–28 in the DAI_STAT register), which tracks the number of samples in FIFO.

  When using the interrupt scheme, the IDP_NSET bits (bits 3–0 of the IDP_CTL register) can be set to *N*, so *N* + 1 data can be read from the FIFO in the interrupt service routine (ISR).
- If the IDP_BHD bit (bit 4 in the IDP_CTL register) is not set, attempts to read more data than is available in the FIFO results in a core hang.

DMA Transfers

DMA access is enabled when the IDP_DMA_EN bit (bit 5 of the IDP_CTL register) is set (= 1).

Starting DMA Transfers

To start a DMA transfer from the FIFO to memory:

1. Clear and halt the FIFO by setting (= 1) and then clearing (= 0) the IDP_ENABLE bit (bit 7 in the IDP_CTL register).
2. While the IDP_DMA_EN and IDP_ENABLE bits are LOW, set the values for the DMA parameter registers that correspond to channels 7–0. If some channels are not going to be used, then the corresponding parameter registers can be left in their default states:
   - Index registers (IDP_DMA_Ix)
   - Modifier registers (IDP_DMA_Mx)
   - Counter registers (IDP_DMA_Cx)

     For each of these registers, "x" is 0 to 7. Refer to "DMA Channel Parameter Registers" on page 6-21.
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_CLK1, SRU_CLK2, SRU_FS1, and SRU_FS2 registers.
4. Set required values for:
   - IDP_SMODEx bits in the IDP_CTL register to specify the frame sync format for the serial inputs (I^2S, Left-justified Sample Pair, or Right-justified Sample Pair modes).
   - IDP_Pxx_PDAPMASK bits in the IDP_PDAP_CTL register to specify the input mask, if the PDAP is used.

- IDP_PORT_SELECT bits in the IDP_PDAP_CTL register to specify input from the DAI pins or the Parallel Port pins, if the PDAP is used.
- IDP_PDAP_CLKEDGE bit (bit 29) in the IDP_PDAP_CTL register to specify if data is latched on the rising or falling clock edge, if the PDAP is used.

5. Connect all of the inputs to the IDP by writing to the SRU_DAT3, SRU_DAT4, SRU_FS1, SRU_FS2, SRU_CLK1, and SRU_CLK2 registers. Keep the clock and frame sync of the ports connected to LOW when data transfer is not intended.
6. Enable DMA, IDP, and PDAP (if required) by setting each of the following bits to one:
   - The IDP_DMA_EN bit (bit 5 of the IDP_CTL register)
   - The IDP_PDAP_EN bit (bit 31 in IDP_PDAP_CTL register)
   - The IDP_ENABLE bit (bit 7 in the IDP_CTL register)

A DAI interrupt is generated at the end of each DMA.

DMA Transfer Notes

The following items provide general information about DMA transfers.

- A DMA can be interrupted by changing the IDP_DMA_EN bit in the IDP_CTL register. None of the other control settings (except for the IDP_ENABLE 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.
- Using DMA transfer overrides the mechanism used for interrupt-driven manual reads from the FIFO. When the IDP_DMA_EN bit is set, the eighth interrupt in the DAI_IRPTL_L or DAI_IRPTL_H

registers (IDP_FIFO_GTN_INT) is *not* generated. This interrupt detects the condition that the number of data available in FIFO is more than the number set in the IDP_NSET bits (bits [3:0]) of the IDP_CTL register).

- 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 have been transferred to memory. These interrupts are mapped to the IDP_DMA7_INT bit (bit 17), to the IDP_DMA0_INT bit (bit 10) in the DAI_IRPTL_L or DAI_IRPTL_H registers and generate interrupts when they are set (= 1). These bits are ORed and reflected in high-level interrupts sent to the core.
- If the combined data rate from the channels is more than the DMA can service, a FIFO overflow occurs. This condition is reflected by the IDP_FIFO_OVER bit (25) in the DAI_STAT register. This is a sticky bit that must be cleared by writing to the IDP_CLROVR bit (bit 6 of the IDP_CTL 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.
- 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, data is transferred as packed 32-bit words.
- 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 IDP_DMA_EN is set, and remain set until the last data from that channel is transferred. Even if IDP_DMA_EN remains set, this bit clears once the required number of data transfers takes place. For more information, see “DAI Pin Status Register (DAI_PIN_STAT)” on page A-109.

Note that when a DMA channel is not used (that is, parameter registers are at their default values), that DMA channel's corresponding IDP_DMAx_STAT bit is set (= 1).

- The three LSBs of data from the serial inputs are channel encoding bits. Since the data is placed into a separate buffer for each channel, these bits are not required and are set to LOW when transferring data to internal memory through the DMA. Bit 3 will still contain the left/right status information.

DMA Channel Parameter Registers

The eight DMA channels each have an I-register (pointer, 19 bits), an M-register (modifier/stride, 6 bits), and a C-register (count, 16 bits). For example, IDP_DMA_I0, IDP_DMA_M0 and IDP_DMA_C0 are the registers that control the DMA for Channel 0. For a detailed description of addressing using the I-register, see "Addressing" on page 2-26.

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.

For a descriptions of these registers see "Input Data Port DMA Control Registers" on page A-99.

IDP (DAI) Interrupt Service Routines for DMAs

The IDP can trigger either the high priority DAI core interrupt reflected in the DAI_IRPTL_H register or the low priority DAI core interrupt reflected in the DAI_IRPTL_L register. The ISR must read the corresponding DAI_IRPTL_H or DAI_IRPTL_L register to find all the interrupts currently latched. The DAI_IRPTL_H register reflects the high priority interrupts and the DAI_IRPTL_L register reflects the low priority interrupts. When these registers are read, it clears the latched interrupt bits. This is a destructive read.

The following steps describe how an IDP ISR should be handled.

1. When the DMA for a channel completes, an interrupt is generated and program control jumps to the ISR.
2. The program should clear the IDP_DMA_EN bit in the IDP_CTL register (= 0).
3. The program should read the DAI_IRPTL_L or DAI_IRPTL_H registers to determine which DMA channels have completed.

   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.

   As each DMA channel completes, a corresponding bit in either the DAI_IRPTL_L or DAI_IRPTL_H registers for each DMA channel is set (IDP_DMAx_INT). Refer to Figure A-59 on page A-112 and Figure A-60 on page A-113 for more information on the DAI_IRPTL_L or DAI_IRPTL_H registers.

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 is not used, then its clock and frame sync must 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_CTL 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 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 will *not* have been reprogrammed and its buffer will overrun.

Input Data Port Programming Example

Listing 6-1 shows a data transfer using an interrupt service routine (ISR). The transfer takes place through the Digital Audio Interface (DAI). This code implements the algorithm outlined in "FIFO to Memory Data Transfer" on page 6-14.

Listing 6-1. Interrupt-Driven Data Transfer

```
/* Using Interrupt-Driven Transfers from the IDP FIFO */

#define IDP_ENABLE   (8)       /* IDP_ENABLE = IDP_CTL[7] */
#define IDP_CTL      (0x24B0)  /* Memory-mapped register */
#define IDP_FIFO_GTN_INT (8)   /* Bit 8 in interrupt regs */
#define IDP_FIFO    (0x24D0)   /* IDP FIFO packing mode */
#define DAI_IRPTL_FE (0x2480)  /* Falling edge int latch */
#define DAI_IRPTL_RE (0x2481)  /* Rising edge int latch */
#define DAI_IRPTL_PRI (0x2484) /* Interrupt priority */

.section/dm seg_dmda;
.var OutBuffer[6];

.section/pm seg_pmco;

initIDP:

  r0 = dm(IDP_CTL);          /* Reset the IDP */
  r0 = BSET r0 BY IDP_ENABLE;
  dm(IDP_CTL) = r0;
  r0 = BCLR r0 BY IDP_ENABLE;

  r0 = BCLR r0 BY 10;    /* Set IDP serial input channel 0 */
  r0 = BCLR r0 BY 9;     /*   to receive in I2S format     */
  r0 = BCLR r0 BY 8;
  dm(IDP_CTL) = r0;
  /*************************************************/
  /* Connect the clock, data and frame sync of IDP */
  /*   channel 0 to DAI pin buffers 10, 11 and 12. */
  /*************************************************/

  /*  Connect IDP0_CLK_I to DAI_PB10_O  */
```

```
/*        (SRU_CLK1[19:15] = 01001)        */

/*  Connect IDP0_DAT_I to DAI_PB11_O  */
/*        (SRU_DAT3[11:6] = 001010)        */

/*  Connect IDP0_FS_I to DAI_PB12_O   */
/*        (SRU_FS1[19:15] = 01011)    */

/*******************************************************/
/* Pin buffers 10, 11 and 12 are always being used as */
/*   inputs.  Tie their enables to LOW (never driven). */
/*******************************************************/

/* Connect PBEN10_I to LOW  */
/* (SRU_PIN1[29:24] = 111110) */

/*  Connect PBEN11_I to LOW  */
/*    (SRU_PIN2[5:0] = 111110)  */

/*  Connect PBEN12_I to LOW  */
/*    (SRU_PIN2[11:6] = 111110)  */


/*********************************************************/
/* Assign a value to N_SET. An interrupt will be raised */
/*   when there are N_SET+1 words in the FIFO.          */
/*********************************************************/

r0 = dm(IDP_CTL);             /* N_SET = 6 */
r0 = BSET r0 BY 0;
r0 = BSET r0 BY 1;
r0 = BSET r0 BY 2;
r0 = BCLR r0 BY 3;
dm(IDP_CTL) = r0;
```

```
  r0 = dm(DAI_IRPTL_RE);          /* Unmask for rising edge */
  r0 = BSET r0 BY IDP_FIFO_GTN_INT;
  dm(DAI_IRPTL_RE) = r0;

  r0 = dm(DAI_IRPTL_FE);         /* Mask for falling edge */
  r0 = BCLR r0 BY IDP_FIFO_GTN_INT;
  dm(DAI_IRPTL_FE) = r0;

  r0 = dm(DAI_IRPTL_PRI);     /* Map to high priority in core */
  r0 = BSET r0 BY IDP_FIFO_GTN_INT;
  dm(DAI_IRPTL_PRI) = r0;

  r0 = dm(IDP_CTL);             /* Start the IDP */
  r0 = BSET r0 BY IDP_ENABLE;
  dm(IDP_CTL) = r0;
initIDP.end:

IDP_ISR:
  i0 = OutBuffer;
  m0 = 1;
  LCNTR = 5, DO RemovedFromFIFO UNTIL LCE;
  r0 = dm(IDP_FIFO);
  dm(i0,m0) = r0;
RemovedFromFIFO:
  RTI;
IDP_ISR.end:
```

Listing 6-2. PDAP Example

```
main:
  IRPTL=0x0;               /* clear all latched interrupts */
  bit set IMASK DAIHI;     /* enable hi-priority DAI interrupt
                              in core interrupt register */
  bit set MODE1 CBUFEN;    /* enable circular buffering */
```

```
  r0 = 0x000FFFFF;
  dm(DAI_PIN_PULLUP) = r0; /* pullup un-used DAI pins */

  ustat2 = dm(IDP_CTL);    /* Reset the IDP by enabling... */
  bit set ustat2 IDP_EN;
  dm(IDP_CTL) = ustat2;
  bit clr ustat2 IDP_EN;   /* ...and then disabling it */
  dm(IDP_CTL) = ustat2;

/*setup for DMA-driven data handling FIFO-->Internal memory */
  r9=INTERNAL_MEM_ADDRESS;
  dm(IDP_DMA_I0)=r9;    /* initialize the index register with the
                           normal-word alias of data buffer to
                           store the data*/
  r0= 1;
  dm(IDP_DMA_M0)=r0;    /* initialize the modify register with
                            a stride of 1  */
  r0= 8;
  dm(IDP_DMA_C0)=r0;    /* FIFO is 8-deep x32, so initialize the
                           count register to 8  */
  ustat2=
  IDP_PDAP_PACKING2|    /* two 16-bit words per 32-bit location
                           in fifo  */
  DP_PP_SELECT|         /* Use AD[15-0] if set, if cleared use
                          DAI_P[20-5]  */
  IDP_P20_PDAPMASK|      /* Bits in the data buffer can be
                            masked out */
                IDP_P19_PDAPMASK|     /* cclr=masked*/
                IDP_P18_PDAPMASK|     /* set=unmasked*/ */
                IDP_P17_PDAPMASK|
                IDP_P16_PDAPMASK|

                IDP_P15_PDAPMASK|
                IDP_P14_PDAPMASK|
```

```
                IDP_P13_PDAPMASK|
                IDP_P12_PDAPMASK|
                IDP_P11_PDAPMASK|
                IDP_P10_PDAPMASK|
                IDP_P09_PDAPMASK|
                IDP_P08_PDAPMASK|
                IDP_P07_PDAPMASK|
                IDP_PDAP_CLKEDGE;    /* latch data in falling
                                        edge of the clock that is
                                        provided to the PDAP  */
     dm(IDP_PP_CTL) = ustat2;
  ustat2 = IDP_DMA0_INT;
     dm(DAI_IRPTL_PRI)=ustat2;   /* unmask individual interrupt
                                    for DMA_INT (PDAP)  in RIC */
     dm(DAI_IRPTL_RE)=ustat2;    /* PDAP interrupt latches on
                                     the rising edge only */
/* Following are two macros that setup the Signal Routing Unit
(SRU) to configure the two pins we'll be using there, PDAP_CLK &
PDAP_HOLD. The data pins in this case are routed through the par-
allel ports AD15-0 pins, but could alternatively be routed via
the SRU */


/* Hold */
SRU(LOW,DAI_PB01_I);
SRU(DAI_PB01_O, IDP0_FS_I);
SRU(LOW,PBEN01_I);


/* Clk */
SRU(LOW,DAI_PB02_I);
SRU(DAI_PB02_O, IDP0_CLK_I);
SRU(LOW,PBEN02_I);


ustat2 = dm(IDP_PP_CTL);
```

```
bit set ustat2 IDP_PDAP_EN;    /* PDAP if set, IDP channel 0
                                  if cleared  */
ustat2 = dm(IDP_CTL);          /* Start the IDP */
bit set ustat2 IDP_EN;
dm(IDP_CTL) = ustat2;

/* in packing mode 2, the data is stored in the buffer like this:
1|OOOOPPPP|
2|MMMMNNNN|
3|KKKKLLLL|
4|IIIIJJJJ|
5|GGGGHHHH|
6|EEEEFFFF|
7|CCCCDDDD|
8|BBBBAAAA|
where AAAA is Sample 1 and BBBB is Sample 2, etc. */

IDP_ISR:        /* This interrupt indicates that the current DMA
                   has completed */

/* test for IDP_DMA0_INT (Read of DAI_IRPTL clears latched
   interrupt) */
      r0=dm(DAI_IRPTL_H);
      btst r0 by 10;
      if not SZ call dma_again;  /* SZ flag cleared if tested
                                    bit = 1 */
      rti;
dma_again:
   ustat2 = dm(IDP_CTL);
   bit clr ustat2 IDP_DMA_EN;    /* disable DMA */
   dm(IDP_CTL) = ustat2;

    rts;
    IDP_ISR.end:
```

7 DIGITAL AUDIO INTERFACE

The Digital Audio Interface (DAI) is comprised of a group of peripherals and the signal routing unit (SRU). The inputs and outputs of the peripherals are not directly connected to external pins. Rather, the SRU connects 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 ADSP-2126x processor to include an arbitrary number and variety of peripherals while retaining high levels of compatibility without increasing pin count.

Structure of the DAI

The DAI incorporates a set of peripherals and a very flexible routing (connection) system permitting a large combination of signal flows. A set of DAI-specific registers make such design, connectivity, and functionality variations possible. All routing related to peripheral states for the DAI interface is specified using DAI registers. For more information on pin states, refer to Figure 7-5 on page 7-7.

The function of the DAI in the ADSP-2126x processor can be compared with the SPORTs' communication with the core. SPORTs communicate with the core directly, just as the DAI communicates directly with the core. The DAI, however, makes use of the SRU to communicate with the core.

The DAI may be used to connect any combination of inputs to any combination of outputs. This function is performed by the SRU via memory-mapped registers.

This *virtual connectivity* design offers a number of distinct advantages:

- Flexibility
- Increased numbers and kinds of configurations
- Connections can be made via software—no hard-wiring is required

Inputs may only be connected to outputs.

DAI System Design

Figure 7-1 and Figure 7-2 show how the DAI pin buffers are connected via the SRU. The SRU allows for very flexible data routing. In its design, the DAI makes use of several types of data from a large variety of sources, including:

- Timers, which are shown in Figure 7-1.
- Six serial ports (SPORTS). Serial ports offer Left-justified Sample Pair and I^2S mode support via 12 programmable and simultaneous receive or transmit pins. These pins support up to 24 transmit or 24 receive I^2S channels of audio when all six SPORTs are enabled, or six full-duplex TDM streams of up to 128 channels per frame. For more information, see “Serial Ports” on page 4-1.
- Precision Clock Generators (PCG). The PCG consists of two units, each of which generates a pair of signals derived from a clock input signal. See “Precision Clock Generator” on page 8-1 for more information.
- Input Data Port (IDP). The IDP provides an additional mechanism for peripherals to communicate with memory. Part of the IDP’s function is to convert information from serial format to parallel format so that it can be moved into memory using a parallel FIFO. IDP is described in “Input Data Port” on page 6-1.

- Digital Audio Interface Pins. These pins provide the physical interface to the SRU. The DAI pins are described in "Pins Interface" on page 7-7.
- Signal Routing Unit. The SRU provides the connection between the serial ports, IDP, and PCG and `DAI_P20-1` pins. The SRU is described in "Signal Routing Unit" on page 7-3.

For a sample of a DAI system configuration, refer to "Using the SRU() Macro" on page 7-30.

Signal Routing Unit

This section describes how to use the signal routing unit (SRU) to connect inputs to outputs.

Connecting Peripherals

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, there is a set of permissible output options. Outputs can feed 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.

The SRU contains six groups that are named sequentially A through F. Each group routes a unique set of signals with a specific purpose. For example, Group A routes clock signals, Group B routes serial data signals, and Group C routes frame sync signals. Together, the SRU's six 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.

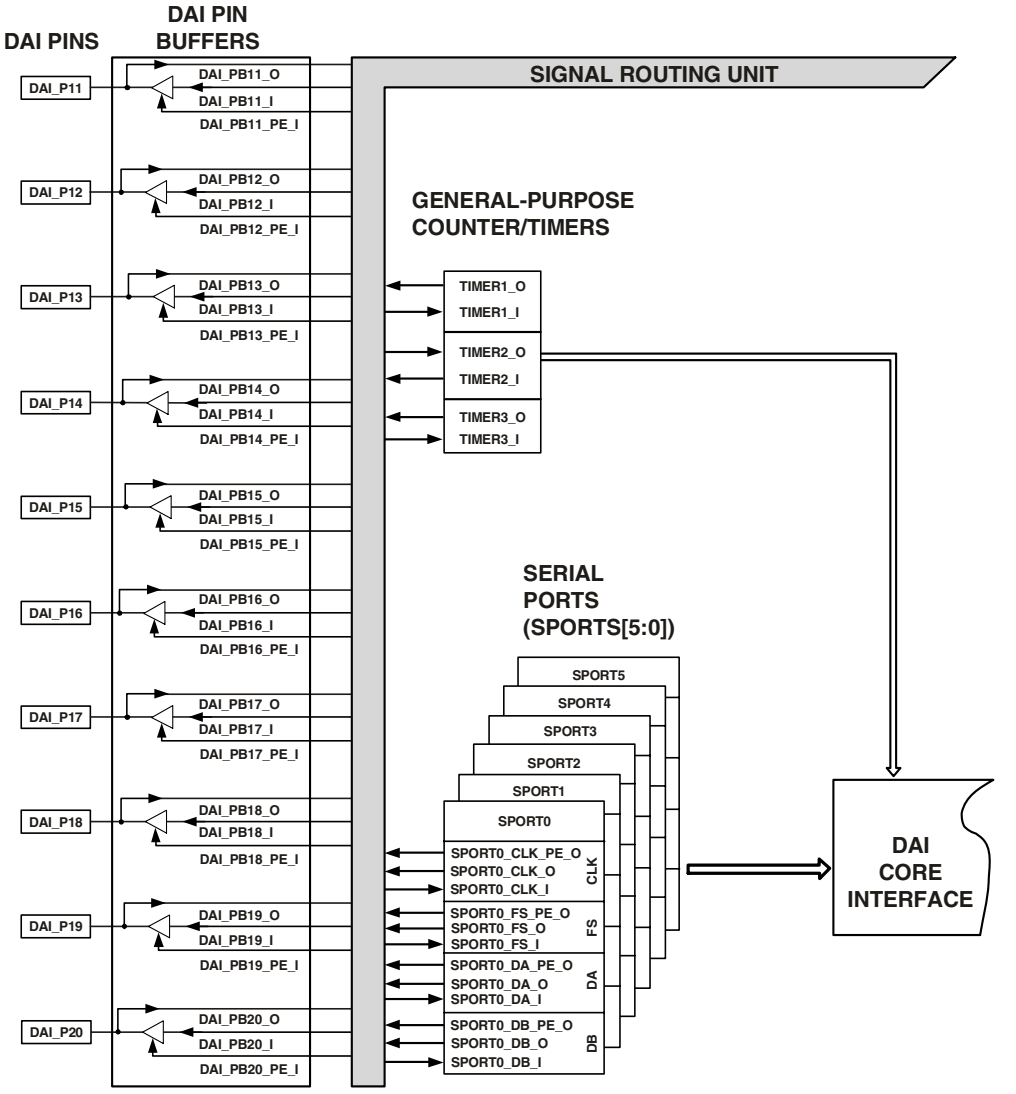


Figure 7-1. DAI System Design

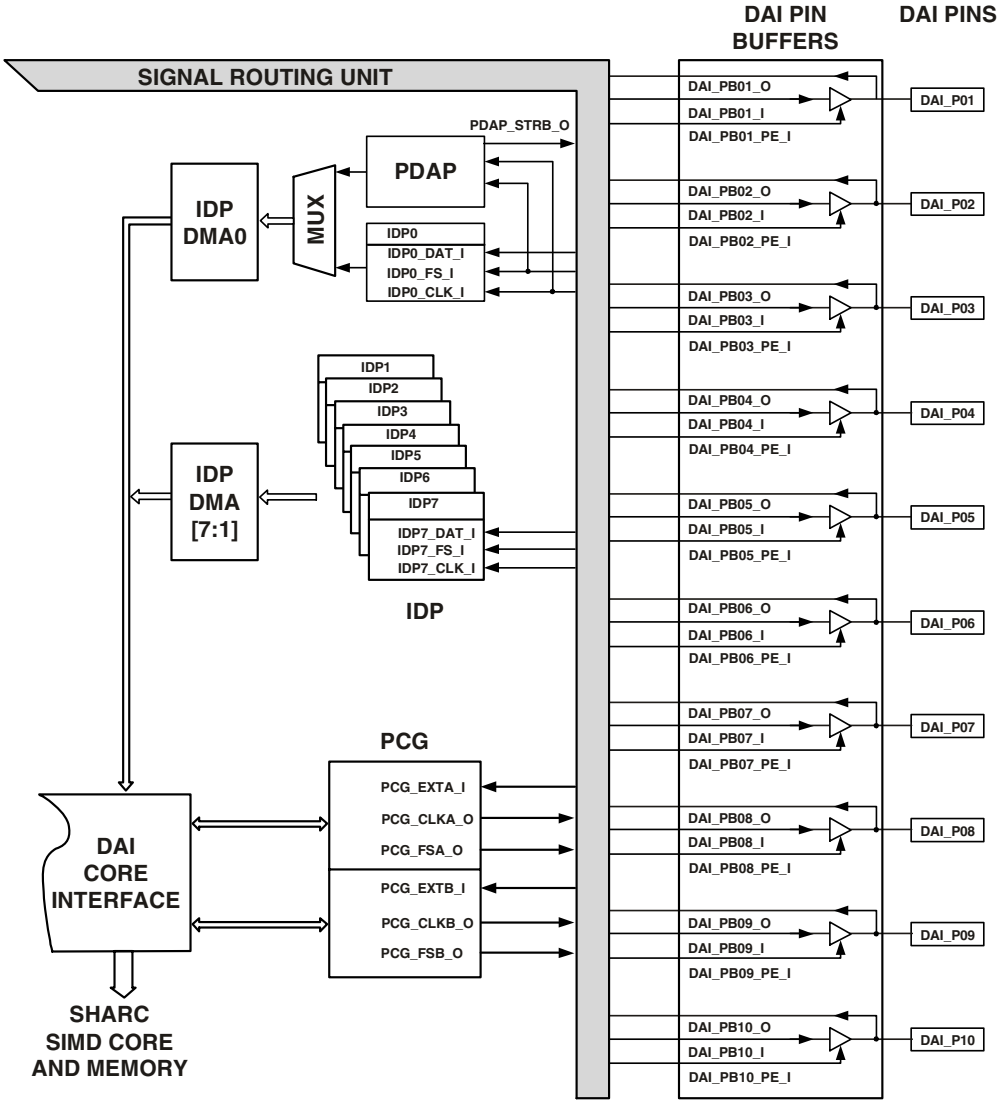


Figure 7-2. DAI System Design (continued)

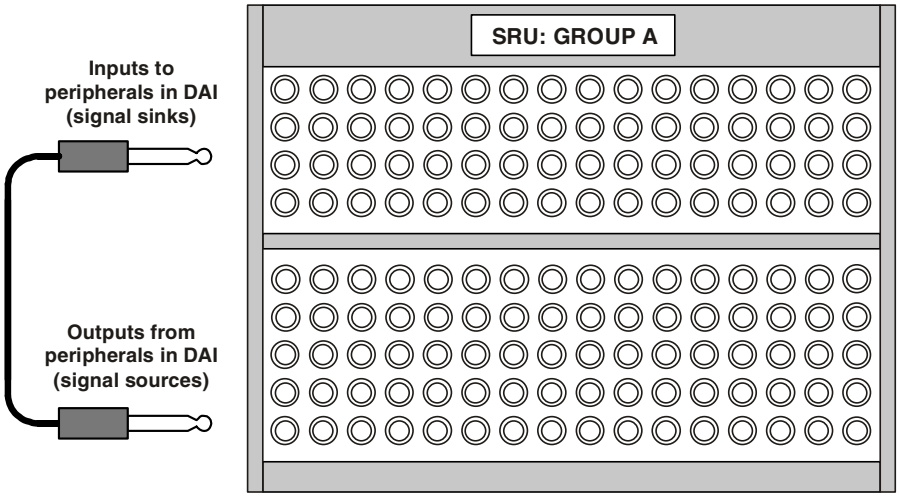


Figure 7-3. Group A as a Patch Bay

Each input and output in each group is given a unique mnemonic. In the few cases where a signal appears in more than one group, the mnemonic is slightly different to distinguish between the connections. 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.

Note that it is not possible to connect a signal in one group directly to a signal in a different group (analogous to wiring from one patch bay to another). However, Group D is largely devoted to routing in this vein.

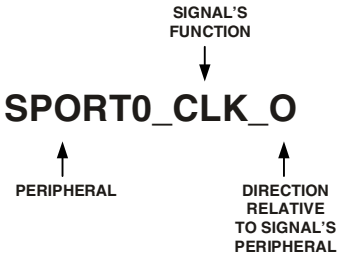


Figure 7-4. Example SRU Mnemonic

Pins Interface

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 a pin input, output, and enable as shown in Figure 7-5. 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.

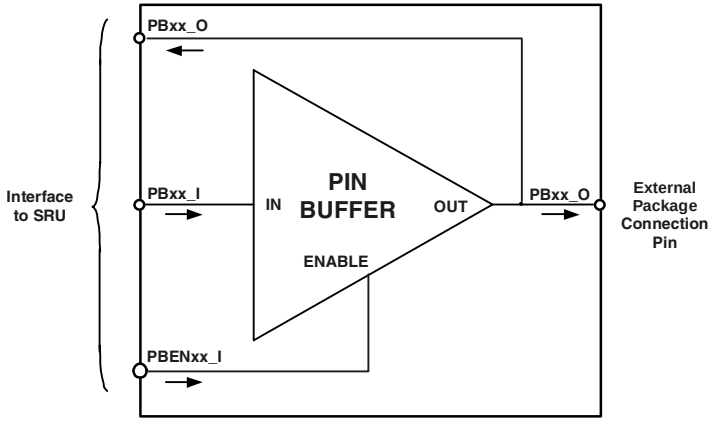


Figure 7-5. Pin Buffer Example

The notation for pin input and output connections can be quite confusing at first because, in a typical system, a pin is simply a wire that connects to a device. The manner in which pins are connected within the SRU requires additional nomenclature. The pin interface's input may be thought of as the input to a buffer amplifier that can drive a load on the physical external lead. The pin interface enable is the input signal that enables the output of the buffer by turning it on when its value is logic high, and turning it off when its value is logic low.

When the pin enable is asserted, the pin output is logically equal to pin input, and the pin is driven. When the pin enable is deasserted, the output of the buffer amplifier becomes high impedance. In this situation, an external device may drive a level onto the line, and the pin is used as an input to the ADSP-2126x processor.

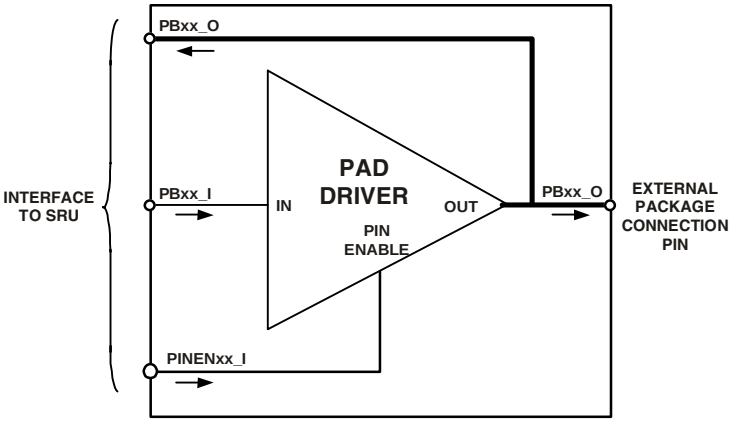


Figure 7-6. Input Signal from Off-chip Drives Pin Output when Pin is not Enabled

While the pin is high impedance and another device is driving a logic level onto the external pin, this value is sent to the SRU as the pin interface output. Even though the signal is an input to the processor, it is an output from the pin interface (as a three-terminal device) and may be patched to

the signal inputs of peripherals within the SRU. Pin output is equal to pin input when the pin enable is asserted, but pin output is equal to the external (input) signal when the pin enable is deasserted.

If a DAI pin is not being used, the pin enable (DAI_PBxx_I) for its pin buffer should be connected to LOW and its associated bit in the DAI_PIN_PULLUP register should be set (= 1) to enable a 22.5 kΩ pull-up resistor for that pin.

Pin Buffers as Signal Output Pins

In a typical embedded system, most pins are designated as either inputs or outputs when the circuit is designed, even if 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 is 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 7-7. 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) will be the same signal as the pin buffer input (PBxx_I), and this signal will be driven as an output.

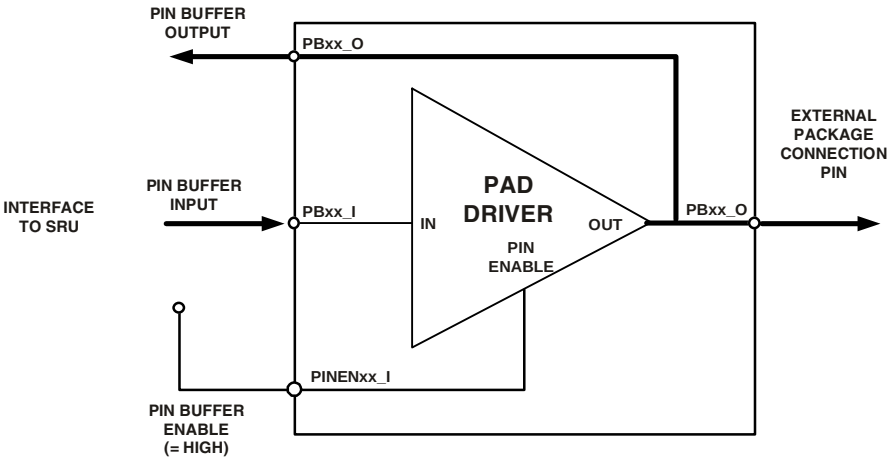


Figure 7-7. Pin Buffer as Output

Pin Buffers as Signal Input Pins

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 7-8. 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) will be the signal driven onto the DAI pin by an external source, and the pin buffer input (PBxx_I) is not used.

Although not strictly necessary, it is recommended programming practice to tie the pin buffer input to logic low whenever the pin buffer enable is tied to logic low. By default, the pin buffer enables are connected to SPORT pin enable signals that may change value. Tying the pin buffer input low decouples the line from irrelevant signals and can make code simpler to debug. It also ensures that no voltage is driven by the pin if a bug in your code accidentally asserts the pin enable.

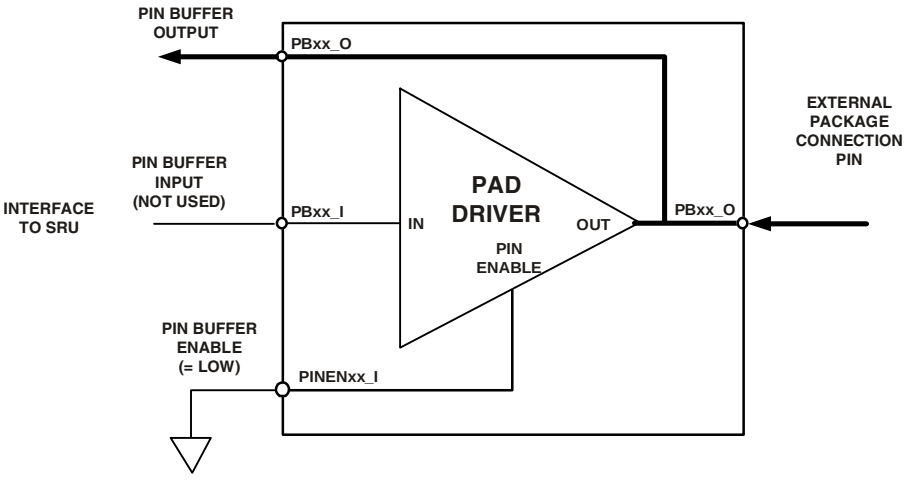


Figure 7-8. Pin Buffer as Input

Bidirectional Pin Buffers

All peripherals within the DAI 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 is driven accordingly. Both the peripherals Controls registers and the configuration of the SRU can effect the direction of signal flow in a pin buffer.

For example, from an external perspective, when a serial port (SPORT) is completely routed off-chip, it uses four pins—clock, frame sync, data channel A, and data channel B. Because all four of these pins comprise the interface that the serial port presents to the SRU, there is a total of 12 connections as shown in Figure 7-9.

For each bidirectional line, the serial port provides three separate signals. For example, a SPORT clock has three separate SRU connections—an input clock to the SPORT (SPORTx_CLK_I), an output clock from the SPORT (SPORTx_CLK_O), and an output enable from the SPORT

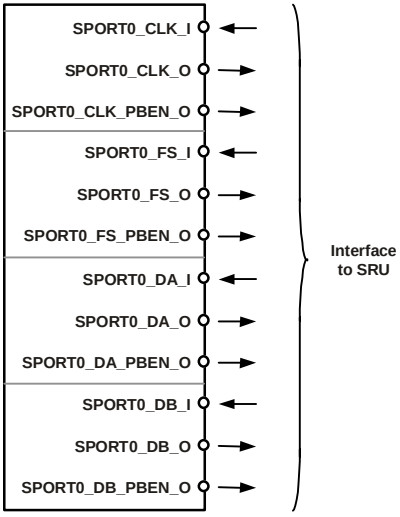


Figure 7-9. SRU Connections for SPORTx

(SPORTx_CLK_PE_O). Note that the input and output signal pair are never used simultaneously. The pin enable signal dictates which of the two SPORT lines appears at the DAI pin at any given time. By connecting all three signals through the SRU, the standard SPORT configuration registers behave as documented in Chapter 4, Serial Ports. The SRU then becomes transparent to the peripheral. Figure 7-10 demonstrates SPORT0 properly routed to DAI pins one through four; although it can be equally well routed to any of the 20 DAI pins.

Though SPORT signals are capable of operating in this bidirectional manner, it is not required that they be connected to the pin buffer this way. As mentioned above, if the system design only uses a SPORT signal in one direction, it is simpler and safer to connect the pin buffer enable pin directly high or low as appropriate. Furthermore, signals in the SRU other than the pin buffer enable signal (which is generated by the peripheral) may be routed to the pin buffer enable input. For example, an outside source may be used to ‘gate’ a pin buffer output by controlling the corresponding pin buffer enable.

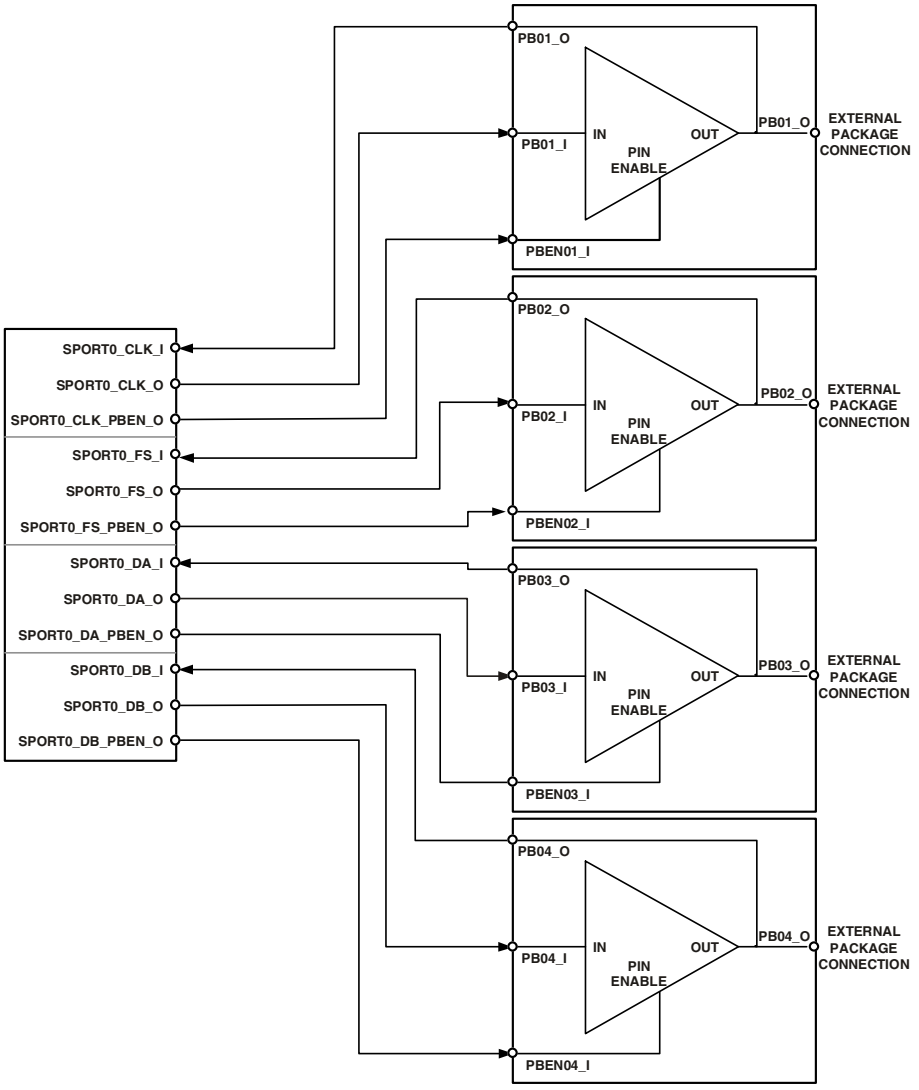


Figure 7-10. SRU Connection to Four Bidirectional SPORT Pins

Making Connections in the SRU

As described previously, 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 as item 1 in Figure 7-11) is written to a bit field corresponding to a signal input (shown as item 2 in Figure 7-11).

The memory-mapped SRU registers are arranged by groups, referred to as Group A through Group F and described in "Signal Routing Unit Registers" on page A-60. Each group has unique encodings for its associated output signals and a set of Configuration registers. For example, Group A is used to route clock signals. Four memory-mapped registers, `SRU_CLK[3:0]`, contain 5-bit wide 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 7-11 diagrams the input signals that are controlled by the Group A register, `SRU_CLK0`. 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.

Note that the lower portion of the patch bay in Figure 7-11 is shown with a large number of ports to reinforce the point that one output can be connected to many inputs. 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.

Just as Group A routes clock signals, each of the other groups route a collection of compatible signals. Group B routes serial data streams. Group C routes frame sync signals. Group D routes signals to pins so that they may

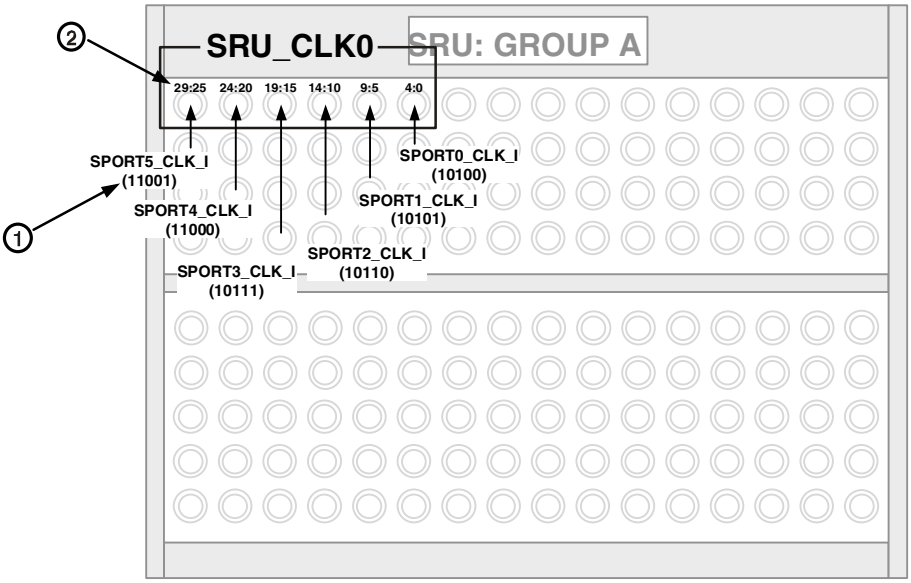


Figure 7-11. Patching to the Group A Register SRU_CLK0

be driven off-chip. Note that all of the groups have encodings that allow a signal to flow from a pin output to the input being specified by the bit field, but Group D is 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. These groups are described in more detail in the following sections.

SRU Connection Groups

There are five separate groups of connections that are used in the SRU. The following sections summarize each.

Group A Connections – Clock Signals

Group A is used to route signals to clock inputs. The SPORTs clock inputs (when the SPORTs are in clock slave mode), the clock inputs to the eight IDP channels and the two Precision Clock Generators (PCGs) external sources are selected from the list of Group A sources and set in the Group A registers. When channel 0 of the IDP is configured for PDAP input, the clock source set here is used as the parallel word latch instead of the serial bit clock.

All unused clock inputs should be set to logic LOW. Any IDP channels that receive clock signals set here will send data to the FIFO. When a SPORT is used as a clock master, setting the unused SPORT clock input to logic LOW improves signal integrity. The registers, and input and output signals for group A are shown in Table 7-5.

Table 7-1. Group A Sources – Serial Clock

Signal Inputs		Signal Sources
Clock Register	**Bit field**	
SRU_CLK0	SPORT0_CLK_I SPORT1_CLK_I SPORT2_CLK_I SPORT3_CLK_I SPORT4_CLK_I SPORT5_CLK_I	• 20 External Pins (DAI_PBxx_O) • 6 Serial Port x Clock Outs (SPORTx_CLK_O) • 2 Precision Clock Generators (A/B) (PCG_CLKx_O) • 2 Logic Level (HIGH/LOW) Options
SRU_CLK1	IDP0_CLK_I IDP1_CLK_I IDP2_CLK_I	
SRU_CLK2	IDP3_CLK_I IDP4_CLK_I IDP5_CLK_I IDP6_CLK_I IDP7_CLK_I	
SRU_CLK3	PCG_EXTA_I PCG_EXTB_I	

Group B Connections – Data Signals

Group B connections, shown in Table 7-2, are used to route signals to serial data inputs. The serial data inputs to both the A and B channels of the SPORTs and to each of the eight IDP channels are selected from the list of Group B sources and set in the Group B registers. When a SPORT is not configured to receive, the data source set here is ignored. Likewise, when channel 0 of the IDP is used for the PDAP, the serial data source set here is ignored.

Table 7-2. Group B Sources – Serial Data

Signal Inputs		Signal Sources
Serial Data Register	**Bit field**	
SRU_DAT0	SPORT0_DA_I SPORT0_DB_I SPORT1_DA_I SPORT1_DB_I SPORT2_DA_I	• 20 External Pins (DAI_PBxx_O) • 12 Serial Port x Data Outs (SPORTx_DB_O) • 2 Logic Level (High/Low) Options
SRU_DAT1	SPORT2_DB_I SPORT3_DA_I SPORT3_DB_I SPORT4_DA_I SPORT4_DB_I	
SRU_DAT2	SPORT5_DA_I SPORT5_DB_I	
SRU_DAT3	IDP0_DAT_I IDP1_DAT_I IDP2_DAT_I IDP3_DAT_I	
SRU_DAT4	IDP4_DAT_I IDP5_DAT_I IDP6_DAT_I IDP7_DAT_I	

Group C Connections – Frame Sync Signals

Group C connections are used to route signals to frame sync inputs. The SPORT frame sync inputs (when the SPORT is in slave mode) and the frame sync inputs to the eight IDP channels are selected from the list of Group C sources and set in the Group C registers.

Each of the frame sync inputs specified is connected to a frame sync source based on the 5-bit values described in the Group C frame sync sources, listed in Table 7-3. Thirty-two possible frame sync sources can be connected using the registers `SRU_FS0-2` described in Figure A-31 on page A-70 through Figure A-33 on page A-71.

Table 7-3. Group C Sources – Frame Sync

Signal Inputs		Signal Sources
Frame Sync Register	**Bit field**	
SRU_FS0	SPORT0_FS_I SPORT1_FS_I SPORT2_FS_I SPORT3_FS_I SPORT4_FS_I SPORT5_FS_I	• 20 External Pins (DAI_PBxx_O) • 6 Serial Port FS Output Options (SPORTx_FS_O) • 2 Precision Frame Sync (A/B) Outputs (PCG_FSx_O) • 2 Frame Sync Logic Level (High/Low) Options
SRU_FS1	IDP0_FS_I IDP1_FS_I IDP2_FS_I	
SRU_FS2	IDP3_FS_I IDP4_FS_I IDP5_FS_I IDP6_FS_I IDP7_FS_I	

Group D Connections – Pin Signal Assignments

Group D is used to specify any signals that will be driven off-chip by the pin buffers. A pin buffer input (DAI_PBxx_I) is driven as an output from the processor when the pin buffer enable is set (= 1). Note that DAI pins 19 and 20 may be configured as either active high or active low by setting the corresponding invert bit.

Each physical pin (connected to a bonded pad) may be routed via the SRU to any of the outputs of the DAI audio peripherals, based on the 6-bit values listed in Table 7-4. The SRU also may be used to route signals that control the pins in other ways. These signals may be configured for use as flags, timers, precision clock generators, or miscellaneous control signals.

Group D registers are SRU_PIN0-3, described in Figure A-34 on page A-74 through Figure A-37 on page A-75.

Table 7-4. Group D Sources – Pin Signal Assignments

Signal Inputs		Signal Sources
DAI Pin Register	**Bit field**	
SRU_PIN0	DAI_PB01_I DAI_PB02_I DAI_PB03_I DAI_PB04_I DAI_PB05_I	• 20 External Pins (DAI_PBxx_O) • 12 Serial Port Data Channel Output Options (two for each SPORT, and one for each Channel A/B) (SPORTx_DB_O) • 6 Serial Port Clock Output Options (one for each SPORT) (SPORTx_CLK_O) • 6 Serial Port FS Output Options (one for each SPORT) (SPORTx_FS_O) • 3 Timers (TIMERx_O) • 6 Flags (FLGxx_O) • 4 Miscellaneous Control B Options (MISCBx_O) • 2 PCG Clock (A/B) Outputs • 2 PCG Frame Sync (A/B) Outputs • 2 Pin Logic Level (High/Low) Designations
SRU_PIN1	DAI_PB06_I DAI_PB07_I DAI_PB08_I DAI_PB09_I DAI_PB10_I	
SRU_PIN2	DAI_PB11_I DAI_PB12_I DAI_PB13_I DAI_PB14_I DAI_PB15_I	
SRU_PIN3	DAI_PB16_I DAI_PB17_I DAI_PB18_I DAI_PB19_I DAI_PB20_I DAI_PB19_INVERT DAI_PB20_INVERT	

Group E Connections – Miscellaneous Signals

Group E connections, shown in Table 7-5, are slightly different from the others in that the inputs and outputs being routed vary considerably in function. This group routes control signals (flags, timers, and so on) and provides a means of connecting signals between groups. Signals with names such as `MISCxy` appear as inputs in Group E, but do not directly feed any peripheral. Rather, they reappear as outputs in Group D and Group F.

Additional connections among Groups D, E, and F provide a surprising amount of utility. Since Group D routes signals off-chip and Group F dictates pin direction, these few signal paths enable an enormous number of possible uses and connections for DAI pins. A few examples include:

- One pin's input can be patched to another pin's output, allowing board-level routing under software control.
- 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.
- Both input and output signals of the timers can be routed to DAI pins. These peripherals are capable of counting in up, down, or elapsed time modes.
- Many types of bidirectional signaling may be created by routing an output of the PCG to a pin enable.

The SRU enables many possible functional changes, both within the processor as well as externally. Used creatively, it allows system designers to radically change functionality at run time, and potentially to reuse circuit boards across many products.

Table 7-5. Group E Sources – Misc. Assignment

Signal Inputs		Signal Sources
DAI Pin Register	**Bit field**	
SRU_EXT_MISCA	MISCA0_I DAI_INT_28 FLG13_I MISCA1_I DAI_INT_29 MISCA2_I DAI_INT_30 FLG14_I MISCA_3_I DAI_INT_31 MISCA_4_I MISCA_5_I INV_MISCA4_I INV_MISCA5_I MISCB_0_I DAI_INT_22 TIMER0_I MISCB_1_I DAI_INT_23 TIMER1_I	• 20 External Pins (DAI_PBxx_O) • 3 Timers (TIMERx_O) • 1 IDP Parallel Input Strobe Output (PDAP_STRB_O) • 2 Clock A/B Outputs (PCG_CLKx_O) • 2 PCG Frame Sync A/B Outputs (PCG_FSx_O) • 2 Logic Level (High/Low) Options
SRU_EXT_MISCB	MISCB_2_I DAI_INT_24 TIMER2_I MISCB_3_I DAI_INT_25 FLG10_I MISCB_4_I DAI_INT26 FLG11_I MISCB_5_I DAI_INT_27 FLG12_I	

Group F – Pin Enable Signals

Group F signals, shown in Table 7-6, are used to specify whether each DAI pin is used as an output or an input by setting the source for the pin buffer enables. When a pin buffer enable (DAI_PBENxx_I) is set (= 1) the signal present at the corresponding pin buffer input (DAI_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 20 DAI pins. When the pins are not enabled (driven), they can be used as inputs.

Table 7-6. Group F Sources – Pin Output Enable

Signal Inputs		Signal Sources
DAI Pin Register	Bit field	
SRU_PBEN0	DAI_PB01_I DAI_PB02_I DAI_PB03_I DAI_PB04_I DAI_PB05_I	• 2 Pin Enable Logic Level (High/Low) Options • 6 Miscellaneous A Control Pins (MISCAx_O) • 24 Pin Enable Options for 6 Serial Ports (one each for FS, Data Channel A/B, and Clock) (SPORTx_CLK_PBEN_O), (SPORTx_FS_PBEN_O), (SPORTx_DA_PBEN_O), (SPORTx_DB_PBEN_O) • 3 Timer Pin Enables (TIMERx_PBEN_O) • 6 Flags Pin Enables (FLGxx_PBEN_O)
SRU_PBEN1	DAI_PB06_I DAI_PB07_I DAI_PB08_I DAI_PB09_I DAI_PB10_I	
SRU_PBEN2	DAI_PB11_I DAI_PB12_I DAI_PB13_I DAI_PB14_I DAI_PB15_I	
SRU_PBEN3	DAI_PB16_I DAI_PB17_I DAI_PB18_I DAI_PB19_I DAI_PB20_I	

General-Purpose (GPIO) and Flags

Any of the DAI pins may also be considered general-purpose input/output (GPIO) pins. Each of the DAI pins can also be set to drive a high or low logic level signal to assert signals. They can also be connected to miscellaneous signals and used as interrupt sources or as control inputs to other blocks. Other than these, out of the 16 flags available, six (10:15) can use 20 DAI pins.

Miscellaneous Signals

In a standard SHARC processor, a clock out connects to a clock in. Likewise, a frame sync out is connected to a frame sync in, and a data out is connected to a data in, and so on. In the ADSP-2126x processor there are exceptions to these standard connection practices. Signals:

- May also be configured as interrupt sources
- Can be configured as invert signals (forcing a signal to active low)
- Can connect one pin to another
- Can be configured as pin enables

DAI Interrupt Controller

The DAI contains a dedicated Interrupt Controller that signals the core when DAI-peripheral events have occurred.

Relationship to the Core

Generally, interrupts are classified as catastrophic or normal. Catastrophic events include any hardware interrupts (for example, resets) and emulation interrupts (under the control of the PC), math exceptions, and

"reads" of memory that do not exist. Catastrophic events are treated as high priority events. In comparison, normal interrupts are "deterministic"—specific events emanating from a source (the causes), the result of which is the generation of an interrupt. The expiration of a timer can generate an interrupt, a signal that a serial port has received data that must be processed, a signal that an SPI has either transmitted or received data, and other software interrupts like the insertion of a trap that causes a breakpoint—all are conditions which identify to the core that an event has occurred.

Since DAI-specific events generally occur infrequently, the DAI IC classifies such interrupts as either high or low priority interrupts. Within these broad categories, users can indicate which interrupts are high and which are classified as low.

Any interrupt causes a two-cycle stall, since it forces the core to stop processing an instruction in process, then vector to the Interrupt Service routine (ISR), (which is basically an Interrupt Vector Table (IVT) lookup), then proceed to implement the instruction referenced in the IVT. For more information, see the appendix "Interrupt Vector Table" in the *ADSP-2126x SHARC DSP Core Manual*.

When an interrupt from the DAI must be serviced, one of the two core ISRs must query the DAI's Interrupt Controller to determine the source(s). Sources can be any one or more of the Interrupt Controller's 32-configurable channels (`DAI_INT[31:0]`). For more information, see "DAI Interrupt Controller Registers" on page A-110.

DAI events trigger two interrupts in the primary 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).

Reading the DAI's interrupt latches clears them. Therefore, the ISR must service *all* the interrupt sources it discovers.

DAI Interrupts

There are several registers in the DAI Interrupt Controller that can be configured to control how the DAI interrupts are reported to and serviced by the core's Interrupt Controller. Among other options, each DAI interrupt can be mapped either as a high or low priority interrupt in the primary interrupt controller, certain DAI interrupts can be triggered on either the rising or falling edge of signals, and each DAI interrupt can also be independently masked.

Just as the core has its own interrupt latch registers (IRPTL and LIRPTL), the DAI has its own latch registers (DAI_IRPTL_L and DAI_IRPTL_H). When a DAI interrupt is configured to be high priority, it is latched in the DAI_IRPTL_H register. When any bit in the DAI_IRPTL_H register is set (= 1), bit 11 in the IRPTL register is also set and the core services that interrupt with high priority. When a DAI interrupt is configured to be low priority, it is latched in the DAI_IRPTL_L register. Similarly, when any bit in the DAI_IRPTL_L register is set (= 1), bit 6 in the LIRPTL register is also set and the core services that interrupt with low priority. Regardless of the priority, when a DAI interrupt is latched and promoted to the core interrupt latch, the ISR must query the DAI's Interrupt Controller to determine the source(s). Sources can be any one or more of the Interrupt Controller's 32-configurable channels (DAI_INT[31:0]). For more information, see "DAI Interrupt Controller Registers" on page A-110.

Reading the DAI's interrupt latches clears them. Therefore, the ISR must service all the interrupt sources it discovers. That is, if multiple interrupts are latched in one of the DAI_IRPTL_x registers, all of them must be serviced before executing an RTI instruction.

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 of the interrupt goes away.

High and Low Priority Latches

In the ADSP-2126x processor, a pair of registers (DAI_IRPTL_H and DAI_IRPTL_L) replace functions normally performed by the IRPTL register. A single register (DAI_IRPTL_PRI) specifies the latch to which each of these interrupts are mapped.

Two registers (DAI_IRPTL_RE and DAI_IRPTL_FE) replace the DAI peripheral's version of the IMASK register. As with the IMASK register, these DAI registers provide a way to specify which interrupts to notice and handle, and which interrupts to ignore. These dual registers function like IMASK does, but with a higher degree of granularity.

Signals from the SRU can be used to generate interrupts. For example, when SRU_EXTMISCA2_INT (bit 30) or DAI_IRPTL_H is set to one, any signal from the external miscellaneous Channel 2 generates an interrupt. If set to one, DAI interrupts trigger an interrupt in the core and the interrupt latch is set. A read of this bit does not reset it to zero. The bit is only set to zero when the cause of the interrupt is cleared. A DAI interrupt indicates the source (in this case, external miscellaneous A, Channel 2), and checks the IVT for an instruction (next operation) to perform.

The 32 interrupt signals within the Interrupt Controller are mapped to two interrupt signals in the primary Interrupt Controller of the SHARC core. The DAI_IRPT_PRI register specifies if the Interrupt Controller interrupt is mapped to the high or low core interrupt (1 = high priority and 0 = low priority).

The DAI_IRPTL_H register is a read-only register with bits set for every DAI interrupt latched for the high priority core interrupt. The DAI_IRPTL_L register is a read-only register with bits set for every DAI interrupt latched for the low priority core interrupt. When a DAI interrupt occurs, the low or high priority core ISR should interrogate its corresponding register to determine which of the 32 interrupt sources to service. When the

DAI_IRPTL_H register is read, the high priority latched interrupts are all cleared. When the DAI_IRPTL_L register is read, the low priority latched interrupts are all cleared.

Rising and Falling Edge Masks

For interrupt sources that correspond to waveforms (as opposed to DAI event signals such as DMA complete or buffer full), the edge of a waveform may be used as an interrupt source as well. Just as interrupts can be generated by a source, interrupts can also be generated latched on the rising (or falling) edges of a signal. This concept does not exist in the main Interrupt Controller, only in the DAI Interrupt Controller.

When a signal comes in, the system needs to determine what kind of signal it is and what kind of protocol, as a result, to service. The preamble indicates the signal type. When the protocol changes, output (signal) type is noted.

For audio applications, the ADSP-2126x processor 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. Programs may elect to use any of four conditions:

- Latch on the rising edge
- Latch on the falling edge
- Latch on *both* the rising and falling edge
- Latch on *neither* the rising *nor* falling edge

The DAI Interrupt Controller may be configured using three registers. Each of the 32 interrupt lines can be independently configured to trigger in response to the incoming signal's rising edge, its falling edge, both the rising edge and the falling edge, or neither the rising edge nor the falling edge. Setting a bit in either the DAI_IRPTL_RE or DAI_IRPTL_FE registers

enables the interrupt level on the rising and falling edges, respectively. For more information on these registers, see Figure A-62 on page A-115 and Figure A-62 on page A-115.

Programs can manage responses to signals by configuring registers. In a sample audio application, for example, upon detection of a change of protocol, the output can be muted. This change of output and the resulting behavior (causing the sound to be muted) results in an alert signal (an interrupt) being introduced in response (if the detection of a protocol change is a high priority interrupt).

The `DAI_IRPTL_FE` register can only be used for latching interrupts on the falling edge.

Use of the `DAI_IRPT_RE` or `DAI_IRPT_FE` registers allows programs to notice and respond to rising edges, falling edges, both rising and falling edges, or neither rising nor falling edges so they can be masked separately.

Enabling responses to changes in condition signals (including changes in DMA state, introduction of error conditions, and so on) can only be enabled using the `DAI_IRPT_RE` register.

Using the SRU() Macro

As discussed above, 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 include file `sru2126x.h`, is included with the 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:

```
SRU(OutputSignal,InputSignal);
```

The names passed to the macro are the names given in Table 7-1 through Table 7-6 and in the DAI registers section in "Signal Routing Unit Registers" on page A-60. Note that each processor has its own specific version of the macro that implements the bit field encodings appropriate to that part. For example, in code for the ADSP-21262, add the following line in your source code:

```
#include <sru21262.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_PBO5_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,DAI_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`.

8 PRECISION CLOCK GENERATOR

The Precision Clock Generator (PCG) consists of two units, each of which generates a pair of signals derived from a clock input signal. The pair of units, A and B, are identical in functionality and operate independently of each other. Each unit generates two signals that are normally used as a clock frame sync pair. The unit that generates the clock is relatively simple, since digital clock signals are usually regular and symmetrical. The unit that generates the frame sync output, however, is designed to be extremely flexible and capable of generating the wide variety of framing signals needed by the many types of peripherals that can be connected to the signal routing unit (SRU). For more information, see "Signal Routing Unit" on page 7-3.

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.

The PCG can accept its clock input either directly from the external oscillator (or discrete crystal) connected to the CLKIN/XTAL pins or from any of the 20 DAI pins. This allows a design to contain an external clock with performance specifications appropriate for the application target.

Note that any clock and frame sync signals generated by the serial ports are also subject to these jitter problems because the SPORT clock is generated from the core clock. However, a SPORT can produce data output while being a clock and frame sync slave. The clock generated by the SPORT is sufficient for most serial communications, but it is suboptimal

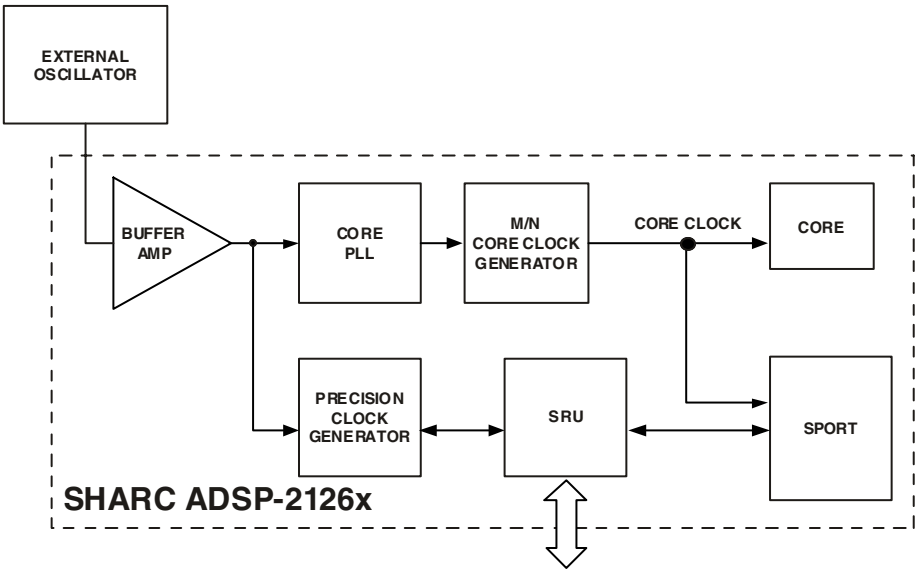


Figure 8-1. Clock Inputs

for analog conversion. Therefore, all precision data converters should be synchronized to a clock generated by the PCG or to a clean (low jitter) clock that is fed into the SRU off-chip via a pin.

Any clock or frame sync unit should be disabled (have its enable bit cleared) before changing any of the associated parameters.

Clock Outputs

As stated in the overview, each of the two units (A and B) produces a clock output and a frame sync output. The clock output is derived from the input to the PCG with a 20-bit divisor.

$$\text{Frequency of Clock Output} = \frac{\text{Frequency of Clock Input}}{\text{Clock Divisor}}$$

If the divisor is either zero or one, the PCG's clock generation unit is bypassed, and the clock input is connected directly to the clock output. Otherwise, the PCG unit clock output frequency is equal to the input clock frequency, divided by a 20-bit integer. The integer is specified in the CLKADIV bit field (bits 19–0 of the PCG_CTLA_1 register) for unit A and a corresponding bit field in the PCG_CTLB_1 register for unit B. See also Figure A-45 on page A-90 and Figure A-47 on page A-92.

The clock outputs have two other control bits that enable the A and B units, ENCLKA and ENCLKB, respectively (bits 31 of the PCG_CTLA_0 and PCG_CTLB_0 registers). These bits enable (= 1) and disable (= 0) the clock output signal for units A and B, respectively. When disabled, clock output is held at logic low.

The CLKASOURCE bit (bit 31 in the PCG_CTLA_1 register) specifies the input source for the clock of unit A. When this bit is cleared (= 0), the input is sourced from the external oscillator, as shown in Figure 8-1. When set (= 1), the input is sourced from the SRU, as specified in the SRU_CLK3 register in PCG_EXTA_I (bits 4–0). See Table A-22 on page A-90.

The PCG unit B functions identically, except that the PCG_CTLB_1 bit (bit 31) indicates that the external source for unit B is specified in PCG_EXTB_I (bits 9–5 of the SRU_CLK3 register). See Table A-25 on page A-93.

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.

A PCG clock output cannot be fed to its own input. Setting SRU_CLK3[4:0] = 28 connects PCG_EXTA_I to logic low, not to PCG_CLKA_O. Setting SRU_CLK3[9:5] = 29 connects PCG_EXTB_I to logic low, not to PCG_CLKB_O.

Frame Sync Outputs

Each of the two units (A and B) 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.

There are two modes of operation for the PCG frame sync. The divisor field determines if the frame sync will operate in Normal mode (divisor > 1) or Bypass mode (divisor = 0 or 1).

Frame Sync

For a given frame sync, the output is determined by the following:

- **Divisor.** A 20-bit divisor of the input clock that determines the period of the frame sync. When set to zero or one, the frame sync operates in Bypass mode, otherwise it operates in Normal mode.
- **Phase.** A 20-bit value that determines the phase relationship between the clock output and the frame sync output. Settings for phase can be anywhere between zero to `DIV - 1`.
- **Pulse width.** A 16-bit value that determines the width of the framing pulse. Settings for pulse width can be zero to `DIV-1`. If the pulse width is equal to zero, then the actual pulse width of the output frame sync is:

$$\text{For even divisors: } \frac{\text{Frame Sync Divisor}}{2}$$

$$\text{For odd divisors: } \frac{\text{Frame Sync Divisor} - 1}{2}$$

The frequency of the frame sync output is determined by:

$$\text{Frequency of Frame Sync Output} = \frac{\text{Frequency of Clock Input}}{\text{Frame Sync Divisor}}$$

When the divisor is set to any value *other* than zero or one, the ADSP-2126x processor operates in Normal mode.

The frame sync A divisor is specified in bits 19–0 of the PCG_CTLA_O register and the frame sync B divisor is specified in bits 19–0 of the PCG_CTLB_O register. The pulse width of frame sync output is equal to the number of input clock periods specified in the 16-bit field of the PCG_PW register. Bits 15–0 specify the pulse width of frame sync A, and bits 31–16 specify the pulse width of frame sync B.

Frame Sync Output Synchronization with External Clock

The frame sync output may be synchronized with an external clock by programming the SRU_EXT_MISCA, SRU_CLK2 and SRU_CLK3 registers appropriately. In this mode, the PCG frame sync output is synchronized with the rising edge of the external clock (shown in Figure 8-2). The external clock is routed to the PCG block from any of the SRU group E sources through the MISCA4_I (for PCGA) and MISCA5_I (for PCGB) signals of the SRU_EXT_MISCA register. For more information, see “Miscellaneous SRU Registers (SRU_EXT_MISCx, Group E)” on page A-79.

Synchronization with the external clock is enabled by setting bit 25 of the SRU_CLK2 register for PCGA frame sync output and bit 10 of the SRU_CLK3 register for PCGB frame sync output. For more information, see “Clock Routing Control Registers (Group A)” on page A-61. The phase must be programmed to three, so that the rising edge of the external clock is in sync with the frame sync. Programming should occur in the following order:

1. Program PCG control registers SRU_EXT_MISCA, SRU_CLK2 and SRU_CLK3 as mentioned above.
2. Enable the clock, frame sync, or both. In other words, program all the values before enabling the PCG (clock and frame sync).

   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.

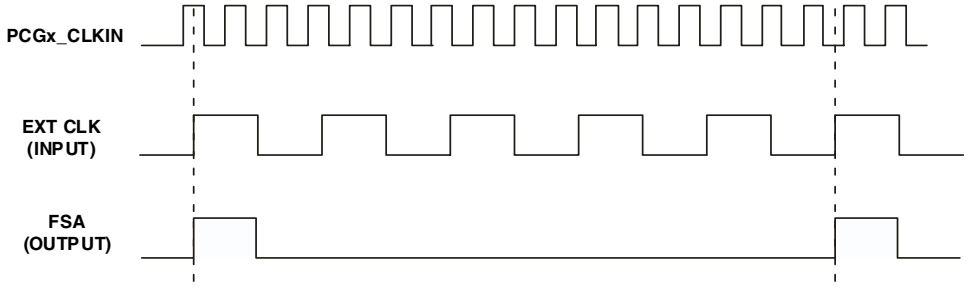


Figure 8-2. Clock Output Synchronization with External Clock

The clock output cannot be aligned with the rising edge of the external clock as there is no phase programmability. Once CLKA and CLKB have been enabled, by programming bit 31 of PCG_CTLA_0 and PCG_CTLB_0 registers respectively, these outputs are activated when a low to high transition is sensed in the external clock (MISCA4_I, MISCA5_I).

Phase Shift

Another PCG frame sync parameter provides for phase shifting with respect to the clock of the same unit. This feature allows shifting in time relative to clock signals. Frame sync phase shifting is often required by peripherals that need to lead or lag a clock signal. For example, the I^2S protocol specifies that the frame sync should transition from high to low

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.

The amount of phase shifting is specified as a 20-bit value in the FSAPHASE_HI bit field (bits 29–20) of the PCG_CTLA_0 register and in the FSAPHASE_LO bit field (bits 29–20) of the PCG_CTLA_1 register for unit A. A single 20-bit value spans these two bit fields. The upper half of the word [19:10] is in the PCG_CTLA_0 register, and the lower half [9:0] is in the PCG_CTLA_1 register.

Similarly, the phase shift for frame sync B is specified in the PCG_CTLB_0 and PCG_CTLB_1 registers.

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.

Phase Shift Settings

The phase shift between clock and frame sync outputs may be programmed under these conditions:

- The input clock source for the clock generator output and the frame sync generator output is the same.
- Clock and frame sync are enabled at the same time using a single atomic instruction.
- Frame sync divisor is an integral multiple of the clock divisor.

If the phase shift is zero, the clock and frame sync outputs rise at the same time. If the phase shift is one, 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. This translates to the input clock period after the clock transition, which further translates to one input clock period after the clock transition.

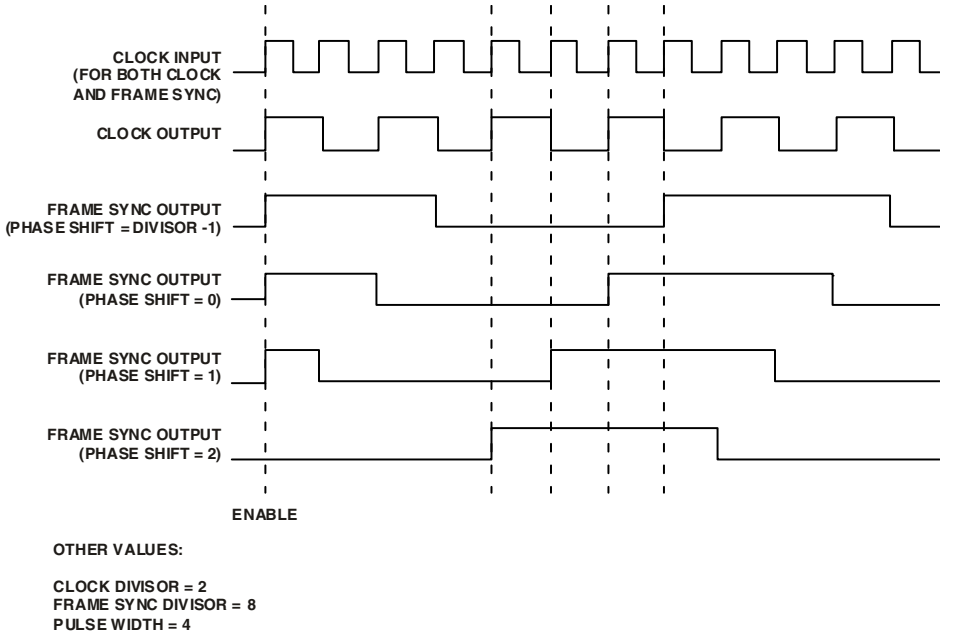


Figure 8-3. Adjusting Frame Sync Phase Shift

Phase shifting is represented as a full 20-bit value so that even when 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.

Pulse Width

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 of frame sync A is specified in bits 15–0 of the PCG_PW register and the pulse width of frame sync B is specified in bits 31–16 of the PCG_PW register.

If the pulse width is equal to zero, then the actual pulse width of the frame sync output is equal to:

$$\frac{\text{DIVISOR}}{2}$$

if the divisor is even, or

$$\frac{\text{DIVISOR} - 1}{2}$$

if the divisor is odd.

Bypass Mode

When the divisor for the frame sync has a value of zero or one, the frame sync is in Bypass mode, and the PCG_PW register has different functionality than in Normal mode. Two bit fields determine the operation in this mode. The One Shot Frame Sync A or B (STROBEx) bit (bits 0 and 16, respectively) determines if the frame sync has the same width as the input, or of a single strobe. These bits also determine whether the Active Low Frame Sync Select for the Frame Sync A or B (INVFSx) bit (bits 1 and 17, respectively) inverts the input. For additional information about the PCG_PW register, see Figure A-48 on page A-93.

In Bypass mode, bits 15–2 and bits 31–18 of the PCG_PW register are ignored.

Bypass as a Pass Through

When the STROBEA bit in the PCG_PW register for unit A or the STROBEB bit in the PCG_PW register for unit B equals zero, the unit is bypassed and the output equals the input. If INVFSA (bit 1) for unit A or INVFSB (bit 17) for unit B is set, then the signal is inverted.

Bypass mode also enables the generation of a strobe pulse ("one shot"). Strobe usage ignores the counter and looks to the SRU to provide the input signal.

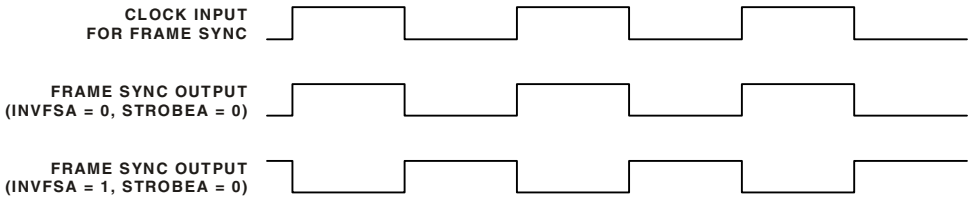


Figure 8-4. Frame Sync Bypass

Bypass as a One Shot

When the STROBEA bit (bit 0 of the PCG_PW register) or STROBEB bit (bit 16 of the PCG_PW register) is set (= 1), the One Shot option is used. When the STROBEx bit is set (= 1), the frame sync is a pulse with a duration equal to one period, or one full cycle, of MISCA2_I for unit A and MISCA3_I for unit B that repeats at the beginning of every clock input period. This pulse is generated during the high period when the INVFSA/B bits (bits 1 or 17, respectively = 0), are cleared or low period when invert bit (INVFSA/B = 1) of the input clock.

A *strobe period* is equal to the period of the normal clock input signal specified by FSASOURCE (bit 30 in the PCG_CTLA_1 register for unit A) and FSBSOURCE (bit 30 in the PCG_CTLB_1 register for unit B).

The output pulse width is equal to the period of the SRU source signal (MISCA2_I for frame sync A and MISCB3_I for frame sync B). The pulse begins at the second rising edge of MISCxx_I following a rising edge of the clock input. When the INVFSA/B bit is set, the pulse begins at the second rising edge of MISCxx_I coincident or following a falling edge of the clock input.

For more information, see "Group E Connections – Miscellaneous Signals" on page 7-22.

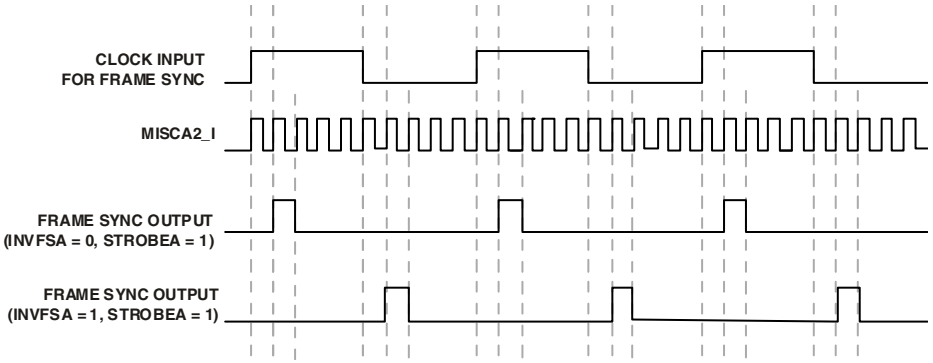


Figure 8-5. One Shot (Synchronous Clock Input and MISCA2_I)

The second INVFSA bit (bit 1) of the Pulse Width Control (PCG_PW) register determines whether the falling or rising edge is used. When set (= 1), this bit selects an active low frame sync, and the pulse comes during the low period of clock input. When cleared (= 0) this bit is set to active high frame sync and the pulse comes during the high period of clock input. For more information on the PCG_PW register, refer to Table A-25 on page A-93.

PCG Programming Examples

This section provides two programming examples written for the ADSP-21262 processor. The first listing, Listing 8-1, uses PCG channel B to output a clock on DAI pin 1 and frame sync on DAI pin 2. The input used to generate the clock and frame sync is CLKIN. This example demonstrates the clock and frame sync divisors, as well as the pulse width and phase shift capabilities of the PCG.

The second listing, Listing 8-2, uses both PCG channels. Channel A is set up to only generate a clock signal. This clock signal is used as the input to channel B via the SRU. The clock and frame sync are routed to DAI pins 1 and 2, respectively, in the same manner as the first example. This frame sync generated in this example is set for a 50% duty cycle, with no phase shift.

Listing 8-1. PCG Channel B Output Example

```
/* Register definitions */
#define SRU_CLK3     0x2434
#define SRU_PIN0     0x2460
#define SRU_PBEN0    0x2478
#define PCG_CTLB0    0x24C2
#define PCG_CTLB1    0x24C3
#define PCG_PW       0x24C4

/* SRU definitions */
#define PCG_CLKB_P       0x39
#define PCG_FSB_P        0x3B
#define PBEN_HIGH_Of     0x01

//Bit Positions
#define DAI_PB02         6
#define PCG_PWB          16
```

```
/* Bit definitions */
#define ENFSB       0x40000000
#define ENCLKB      0x80000000

/* Main code section */
.global _main;
.section/pm seg_pmco;
_main:
/* Route PCG Channel B clock to DAI Pin 1 via SRU */
/* Route PCG Channel B frame sync to DAI Pin 2 via SRU */
r0 = PCG_CLKB_P|(PCG_FSB_P<<DAI_PB02);
dm(SRU_PIN0) = r0;

/* Enable DAI Pins 1 & 2 as outputs */
r0 = PBEN_HIGH_Of|(PBEN_HIGH_Of<<DAI_PB02);
dm(SRU_PBEN0) = r0;

r0 = (100<<PCG_PWB); /* PCG Channel B FS Pulse width = 100 */
dm(PCG_PW) = r0;

r2 = 1000; /* Define 20-bit Phase Shift */
r0 = (ENFSB|ENCLKB| /*Enable PCG Channel B Clock and FS*/
        1000000); /* FS Divisor = 1000000 */
r1 = lshift r2 by -10;
/* Deposit the upper 10-bits of the Phase Shift in the */
/* correct position in PCG_CTLB0 (Bits 20-29) */
r1 = fdep r1 by 20:10;
r0 = r0 or r1; /* Phase Shift 10-19 = 0 */
dm(PCG_CTLB0) = r0;


r0 = (100000);  /* Clk Divisor = 100000 */
            /* Use CLKIN as clock source */
```

```
/* Deposit the lower 10-bits of the Phase Shift in the */
/* correct position in PCG_CTLB1 (Bits 20-29) */
r1 = fdep r2 by 20:10;
r0 = r0 or r1; /* Phase Shift 10-19 = 0x3E8 */
dm(PCG_CTLB1) = r0;
//---------------------------------------
_main.end: jump(pc,0);
```

Listing 8-2. PCG Channel A and B Output Example

```
/* Register Definitions */
#define SRU_CLK3    0x2434
#define SRU_PIN0    0x2460
#define SRU_PBEN0   0x2478
#define PCG_CTLA0   0x24C0
#define PCG_CTLA1   0x24C1
#define PCG_CTLB0   0x24C2
#define PCG_CTLB1   0x24C3
#define PCG_PW      0x24C4

/* SRU Definitions */
#define PCG_CLKA_O      0x1c
#define PCG_CLKB_P      0x39
#define PCG_FSB_P       0x3B
#define PBEN_HIGH_Of    0x01

//Bit Positions
#define PCG_EXTB_I      5
#define DAI_PB02        6
#define PCG_PWB         16

/* Bit Definitions */
#define ENCLKA      0x80000000
```

```
#define ENFSB       0x40000000
#define ENCLKB      0x80000000
#define CLKBSOURCE  0x80000000
#define FSBSOURCE   0x40000000

/* Main code section */
.global _main; /* Make main global to be accessed by ISR */
.section/pm seg_pmco;
_main:
/*Route PCG Channel A clock to PCG Channel B Input via SRU*/
r0 = (PCG_CLKA_O<<PCG_EXTB_I);
dm(SRU_CLK3) = r0;

/* Route PCG Channel B clock to DAI Pin 1 via SRU */
/* Route PCG Channel B frame sync to DAI Pin 2 via SRU */
r0 = (PCG_CLKB_P|(PCG_FSB_P<<DAI_PBO2));
dm(SRU_PIN0) = r0;

/* Enable DAI Pins 1 & 2 as outputs */
r0 = (PBEN_HIGH_Of|(PBEN_HIGH_Of<<DAI_PBO2));
dm(SRU_PBEN0) = r0;

r0 = ENCLKA; /* Enable PCG Channel A Clock, No Channel A FS */
           /* FS Divisor = 0, FS Phase 10-19 = 0 */
dm(PCG_CTLA0) = r0;

r1 = 0xfffff; /* Clk Divisor = 0xfffff, FS Phase 0-9 = 0 */
            /* Use CLKIN as clock source */
dm(PCG_CTLA1) = r1;

r0 = (5<<PCG_PWB); /* PCG Channel B FS Pulse width = 1 */
dm(PCG_PW) = r0;

r0 = (ENFSB|ENCLKB|10); /*Enable PCG Channel B Clock and FS*/
```

```
            /* FS Divisor = 10, FS Phase 10-19 = 0 */
dm(PCG_CTLB0) = r0;

r0 = (CLKBSOURCE|FSBSOURCE|10); /* Clk Divisor = 10 */
     /* FS Phase 0-9 = 0, Use SRU_MISC4 as clock source */
dm(PCG_CTLB1) = r0;

_main.end: jump(pc,0);
```

9 SYSTEM DESIGN

The processor supports many system design options. The options implemented in a system are influenced by cost, performance, and system requirements. This chapter provides the following system design information:

Other chapters also discuss system design issues. Some other locations for system design information include:

By following the guidelines described in this chapter, you can ease the design process for your ADSP-2126x processor product. Development and testing of your application code and hardware can begin without debugging the debug port.

Before proceeding with this chapter it is recommended that you become familiar with the ADSP-2126x core architecture. This information is presented in the *ADSP-2126x SHARC Processor Core Manual*.

Pin Descriptions

This section describes the pins and shows how these signals can be used in a processor system. Table 9-1 illustrates how the pins are used in a typical system.

Pin definitions are listed in Table 9-1. The following symbols appear in the **Type** column of Table 9-1:

- A = Asynchronous
- G = Ground
- I = Input
- O = Output
- P = Power Supply
- S = Synchronous
- (A/D) = Active Drive
- (O/D) = Open Drain
- T = Three-State

Unlike previous SHARC processors, the ADSP-2126x processor contains internal series resistance equivalent to 360 Ω on the input path of all pins.

Table 9-1 includes only a brief description of pins. For a complete description, refer to the product specific data sheet. The data sheet contains the most current and detailed information about this product.

Table 9-1. Pin Descriptions

Pin	Type	State During and After Reset	Function
AD15–0	I/O/T	Three-state with pull-up enabled	Parallel Port Address/Data
$\overline{RD}$	O	Output only, driven high[1]	Parallel Port Read Enable
$\overline{WR}$	O	Output only, driven high[1]	Parallel Port Write Enable
ALE	O	Output only, driven low[1]	Parallel Port Address Latch Enable
FLG3–0	I/O/A	Three-state	Flag Pins
DAI_P20–1	I/O/T	Three-state with programmable pull-up	Digital Audio Interface Pins
SPICLK	I/O	Three-state with pull-up enabled	Serial Peripheral Interface Clock Signal
$\overline{SPIDS}$	I	Input only	Serial Peripheral Interface Slave Device Select
MOSI	I/O (O/D)	Three-state with pull-up enabled	SPI Master Out Slave In
MISO	I/O (O/D)	Three-state with pull-up enabled	SPI Master In Slave Out
BOOTCFG1–0	I	Input only	Boot Configuration Select
CLKIN	I	Input only	Local Clock In

Table 9-1. Pin Descriptions (Cont'd)

Pin	Type	State During and After Reset	Function
XTAL	O	Output only[2]	Crystal Oscillator Terminal
CLKCFG1–0	I	Input only	Core/CLKIN Ratio Control
CLKOUT	O	Output only	Local Clock Out/ Reset Out
$\overline{\text{RESET}}$	I/A	Input only	Processor Reset
TCK	I	Input only	Test Clock (JTAG)
TMS	I/S	Three-state with pull-up enabled	Test Mode Select (JTAG)
TDI	I/S	Three-state with pull-up enabled	Test Data Input (JTAG)
TDO	O	Three-state	Test Data Output (JTAG)
$\overline{\text{TRST}}$	I/A	Three-state with pull-up enabled	Test Reset (JTAG)
$\overline{\text{EMU}}$	O (O/D)	Three-state with pull-up enabled	Emulation Status
V_{DDINT}	P		Core Power Supply
V_{DDEXT}	P		I/O Power Supply
A_{VDD}	P		Analog Power Supply
A_{VSS}	G		Analog Power Supply Return
GND	G		Power Supply Return

1 $\overline{\text{RD}}$, $\overline{\text{WR}}$, and ALE are continuously driven by the processor and will not be three-stated.
2 Output only is a three-state driver with its output path always enabled.

Inputs identified as synchronous (S) must meet timing requirements with respect to `CLKIN` (or with respect to `TCK` for `TMS`, `TDI`). Inputs identified as asynchronous (A) can be asserted asynchronously to `CLKIN` (or to `TCK` for $\overline{\texttt{TRST}}$).

Tie or pull unused inputs to V_{DDEXT} or GND, except for the following: AD15-0, DAI_Px, SPICLK, MISO, MOSI, $\overline{\text{EMU}}$, TMS, and TDI. These pins have a pull-up resistor and can be left floating. See the pin list in Table 9-1.

The $\overline{\text{TRST}}$ input of the JTAG interface must be asserted (pulsed low) or held low after power-up for proper operation of the processor. Do not leave this pin unconnected.

Pin Multiplexing

The ADSP-2126x processor provides the same functionality as other SHARC processors but with a much lower pin count which helps to reduce total system costs. It does this through extensive use of pin multiplexing. Table 9-3 shows an example multiplexing scheme. The following registers (addresses) and bits are used.

Table 9-2. Register and their Bits Used for Multiplexing

Registers Used (Address)	Bits Used
SYSCTL (0x3024)	PPFLGS, TMREXPEN, IRQxEN
SPIFLG (0x1001)	SPIFLGx (3:0)
SPICTL (0x1000)	SPIMS
IDP_PDAP_CTL (0x24B1)	IDP_PP_SELECT
PMCTL (0x2000)	CLOCKOUTEN

Table 9-3. ADSP-2126x Processor Pin Multiplexing Scheme

External Pin	Function	Type I = input O = output	Control 0 = cleared 1 = set x = do not care
FLGn[1]	FLGn	I/O	PPFLGS = 0 SPIFLG[n] = 0 and SPIMS=0 IRQxEN = 0
	$\overline{\text{IRQn}}$[2]	I	PPFLGS=0 SPIFLG[n] = 0 and SPIMS = 0 IRQxEN = 1
	SPI Device Select[3]	O	PPFLGS=0 SPIFLG[n] = 1 and SPIMS = 1 IRQxEN = 0
AD[15:0]	AD[15:0]	I/O	PPFLGS = 0[4] IDP_PP_SELECT = 0
	PDAP	I	PPFLGS = 0 IDP_PP_SELECT = 1
	FLG[15:0]	I/O	PPFLGS = 1[5] IDP_PP_SELECT = 0
DAI_P[20:1][6]	PDAP	I	IDP_PP_SELECT = 0
	FLG[15:10]	I/O	Note[7]
	Other	I/O	Note [6]
CLKOUT	CLKOUT	O	PMCTL [12] = 1
	$\overline{\text{RESETOUT}}$	O	PMCTL [12] = 0

1 n = 0, 1, 2, 3.
2 For n = 3 function is FLG3 or TIMEXP, not $\overline{\text{IRQ3}}$.
3 These pins are used at boot time as device selects during SPI Master booting.
4 Setting PPFLGS = 1 and IDP_PP_SELECT = 1 at the same time is illegal.
5 When PPFLGS = 1, the FLG pins toggle then alternate functions. For example $\overline{\text{IRQx}}$ and TIMEXP.
6 For complete information on the operation of these pins, see "Digital Audio Interface" on page 7-1.
7 For complete information on the operation of these pins, see "Clock Routing Control Registers (Group A)" on page A-61.

If the system clock to the SPICLK module is shut off in the PMCTL register, FLG0-3 are not usable.

Address/Data Pins as FLAGs

To use these pins as flags (FLGS15-0) set (= 1) bit 20 of the SYSCTL register and disable the Parallel Port.

Table 9-4. AD[15:0] to FLAG Pin Mapping

AD Pin	FLAG PIN	AD Pin	FLAG Pin
AD0	FLAG8	AD8	FLAG0
AD1	FLAG9	AD9	FLAG1
AD2	FLAG10	AD10	FLAG2
AD3	FLAG11	AD11	FLAG3
AD4	FLAG12	AD12	FLAG4
AD5	FLAG13	AD13	FLAG5
AD6	FLAG14	AD14	FLAG6
AD7	FLAG15	AD15	FLAG7

Input Synchronization Delay

The processor has several asynchronous inputs—$\overline{\text{RESET}}$, $\overline{\text{TRST}}$, $\overline{\text{IRQ2-0}}$, and FLG11-0 (when configured as inputs). These inputs can be asserted in arbitrary phase to the processor clock, CLKIN. The processor synchronizes the inputs 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 full processor cycle plus setup and hold time, except for RESET, which must be asserted for at least four processor cycles. The minimum time prior to recognition (the setup and hold time) is specified in the data sheet.

Clock Derivation

The processor uses a PLL on the chip, to provide clocks that switch at higher frequencies than the system clock (CLKIN). The PLL-based clocking methodology used influences the clock frequencies and behavior for the serial, SPI, and parallel ports, in addition to the processor core and internal memory. In each case, the processor PLL provides a non-skewed clock to the port logic and I/O pins.

The PLL provides a clock that switches at the processor core frequency to the serial ports. Each of the serial ports can be programmed to operate at clock frequencies derived from this clock. The six serial ports' transmit and receive clocks are divided down from the processor core clock frequency by setting the DIVx registers appropriately.

On power-up, the CLKCFG1-0 pins are used to select ratios of 16:1, 8:1, and 3:1. After booting, numerous other ratios (slowing or speeding up the clock) can be selected via software control using the Power Management Control register.

Power Management Control Register

The ADSP-2126x processor has a Power Management Control register (PMCTL) that allows programs to determine the amount of power dissipated. This includes the ability to program the PLL dynamically in software. This feature eases design for systems that need to use specific clock frequencies or are sensitive to power consumption.

In addition to changing the clock rate on the fly, The PMCTL register also allows programs to disable the clock source to a particular processor peripheral completely, (for example the serial ports or the timers), to further conserve power. By default, each peripheral block has its internal CLK enabled only after it is initialized. Programs can use the PMCTL register to turn the specific peripheral off after the application no longer needs it. After reset these clocks are not enabled until the peripheral is initialized by the program.

Listing 9-1 and Listing 9-2 are examples that show how to use the Power Management Control register to enable/disable clocking to a peripheral.

Listing 9-1. Power Management Example

```
ustat2 = dm(PMCTL);
bit set ustat2 SPIPDN;  /* disable internal peripheral clock for
                           SPI module. SPIPDN is defined as bit 3
                           of PMCTL*/
dm(PMCTL) = ustat2;
```

Listing 9-2. PMCTL Example Code.

```
ENABLING CLKOUT:
  ustat2 = dm(PMCTL);
  bit set ustat2 CLKOUTEN;  /* switch pin function from Reset
                               Out(RSTOUT) to CLKOUT */
  dm(PMCTL) = ustat2;

PLL Divisor modification:
  ustat2 = dm(PMCTL);
  bit set ustat2 DIVEN|PLLD8;  /* set and enable PLL Divisor for
                                  CoreCLK = CLKIN/8 */
  dm(PMCTL) = ustat2;
```

```
PLL Multiplier modification:
  ustat2 = dm(PMCTL);
  bit set ustat2 PLLM8 | PLLBP;   /* set a multiplier of 8
                                     (default divisor is 2) and put
                                     PLL in Bypass */
  dm(PMCTL) = ustat2;
  waiting loop:
  r0 = 4096;   /* wait for PLL to lock at new rate
                  (requirement for modifying multiplier only) */
  lcntr = r0, do pllwait until lce;
  pllwait:nop;
  ustat2 = dm(PMCTL);
  bit clr ustat2 PLLBP;
 /* take PLL out of Bypass, PLL is now at CLKIN*4 (CoreCLK = CLKIN
* M/N = CLKIN* 8/2) */
  dm(PMCTL) = ustat2;

PLL Input Divider Usage:
ustat2 = dm(PMCTL);
bit set ustat2 INDIV | PLLBP;   /* divide clkin/2, put
                                   PLL in Bypass */
dm(PMCTL) = ustat2;

waiting loop:
r0 = 4096;        /* wait for PLL to lock at new rate
                     (requirement for modifying multiplier
                     and setting INDIV bit only) */
lcntr = r0, do pllwait until lce;
pllwait:nop;
ustat2 = dm(PMCTL);
bit clr ustat2 PLLBP;
/* take PLL out of Bypass */
dm(PMCTL) = ustat2;
```

```
PMCTL register bit definitions:
/* Power Management Control register (PMCTL) */
#define PLLM8   (BIT_3)     // PLL Multiplier 8
#define PLLD8   (BIT_7)     // PLL Divisor 8
#define INDIV   (BIT_8)     // Input Divider
#define DIVEN   (BIT_9)     // Enable PLL Divisor
#define CLKOUTEN (BIT_12)   // Mux select for CLKOUT/RESETOUT
#define PLLBP   (BIT_15)    // PLL Bypass mode indication
#define SPIPDN  (BIT_30)    // Shutdown clock to SPI
```

Timing Specifications

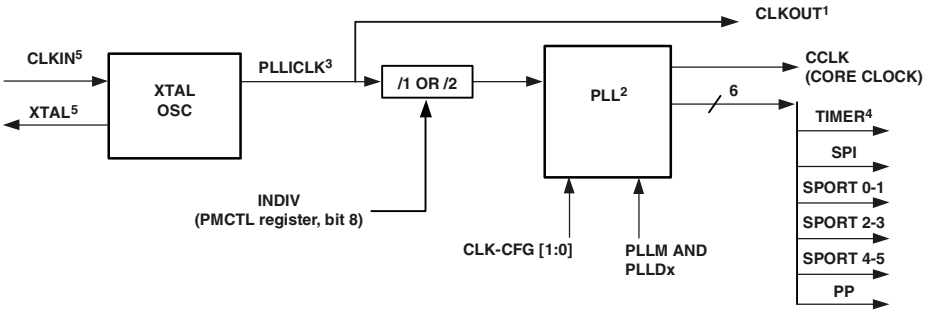
The ADSP-2126x processor's internal clock (a multiple of CLKIN) provides the clock signal for timing internal memory, the processor core, serial ports, the SPI, and the parallel port (as required for read/write strobes). During reset, program the ratio between the processor's internal clock frequency and external (CLKIN) clock frequency with the CLK_CFG1-0 pins.

To determine switching frequencies for the serial ports, divide down the internal clock, using the programmable divider control of each port. For the SPI port, the BAUDR bit in the SPICTL register controls the SPICLK baud rate based on the core clock frequency. For the serial ports, use the appropriate DIVx register.

Use the equation CCLK = PLLICLK x PLL Multiply Ratio to set clock periods. The PLL multiply ratio is determined by the CLK_CFG1-0 pins. See Figure 9-1 and Table 9-9 on page 9-14.

The following tables provide descriptions of the various clock definitions, inputs, outputs and uses in an ADSP-2126x processor system.

Table 9-7 describes clock ratio requirements. Table 9-8 shows an example clock derivation.



Notes
1. CLKOUT is muxed with RSTOUT. After reset, RSTOUT is selected. CLKOUT is selected by setting bit 12 in the PMCTL register.
2. The PLL ratio is controlled by the states of the CLKCFG[1:0] pins at reset and can be modified in software via the PLLM & PLLDx bits in the PMCTL register.
3. To place the PLL in Bypass mode, set bit 15 in the PMCTL register. (CCLK = PLLICLK when set.)
4. Programs can interrupt the internal clock source to each of the following peripherals: Timer, SPI, SPORTs, and Parallel Port. These internal clock sources are disabled at reset and are enabled and left enabled after each peripheral is enabled. A modest power savings can be achieved by disabling these clocks when they are no longer needed. Note that these peripherals DO NOT RUN at the Core Clock frequency. For more information please see the respective chapters in the *ADSP-2126x SHARC DSP Peripherals Manual*.
5. Please refer to one of the ADSP-2126x family datasheets for maximum CLKIN and crystal source specifications.

Figure 9-1. Core Clock and System Clock Relationship to CLKIN

Table 9-5. CLKOUT and CCLK Clock Generation Operation

Timing Requirements		Calculation		Description
CLKIN	=	$1/t_{CKIN}$	=	Input Clock
CLKOUT	=	$1/t_{TCK}$	=	Local Clock Out
PLLICLK	=	$1/t_{PLLIN}$	=	PLL Input Clock
CCLK	=	$1/t_{CCLK}$	=	Core Clock

Table 9-6. Clock Relationships

Timing Requirements		Description[1]
t_{CK}	=	CLKOUT Clock Period
t_{PLLICK}	=	PLL Input Clock

Table 9-6. Clock Relationships (Cont'd)

Timing Requirements		Description[1]
t_{CCLK}	=	Core Clock Period (Processor)
t_{SCLK}	=	Serial Port Clock Period = (t_{CCLK}) x SR
t_{SPICLK}	=	SPI Clock Period = (t_{CCLK}) x SPIR

1 where:
SR = serial port-to-core clock ratio (wide range, determined by the DIVx register)
SPIR = SPI-to-core clock ratio (wide range, determined by the SPICTL register)
SCLK = serial port clock
SPICLK = SPI clock

Table 9-7. Clock Ratios

Timing Requirements		Description
c_{RTO}	=	Core:CLKOUT ratio, (3:1, 8:1, or 16:1, determined by CLK_CFGx pins at reset. Programs can modify this ratio using the PMCTL register.)
s_{RTO}	=	Sport:core clock ratio (wide range determined by xCLKDIV)

Table 9-8. Clock Derivation

Timing Requirements		Description
t_{CCLK}	=	(t_{CK}) x cRTO
t_{SCLK}	=	(t_{CCLK}) x sRTO

RESET and CLKIN

The processor receives its clock input on the `CLKIN` pin. The processor uses an on-chip phase-locked loop (PLL) to generate its internal clock, which is a multiple of the `CLKIN` frequency (Figure 9-1 on page 9-12). Because the PLL requires some time to achieve phase lock, `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, see the specific ADSP-2126x data sheet.

Table 9-9 and Table 9-10 show the internal clock to CLKIN frequency ratios supported by the processor. Note that programs control the PLL through the PMCTL register. This register is described in the *ADSP-2126x SHARC Processor Core Manual*.

When using an external crystal, the maximum crystal frequency cannot exceed 25 MHz. The internal clock generator, when used in conjunction with the XTAL pin and an external crystal, is designed to support up to a maximum of 25 MHz external crystal frequency. For all other external clock sources, the maximum CLKIN frequency is 50 MHz.

Table 9-9. Clock Rate Ratios After Reset (Default)

CLKCFG[1-0]	Core to CLKIN Ratio
00	3:1, PLLD = 2, PLLM = 6
01	16:1, PLLD = 2, PLLM = 32
10	8:1, PLLD = 2, PLLM = 16
11	Reserved

Table 9-10. PLL Multiplier and Divider Settings

PLLD[7:6]	PLL Divider Ratio	PLLM[5:0]	PLL Multiplier Ratio
00 (= reset)	Clock Divider = 2	000000	Clock Multiplier = 64
01	Clock Divider = 4	000001	Clock Multiplier = 1
10	Clock Divider = 8	...	...
11	Clock Divider = 16	111111	Clock Multiplier = 63

Table 9-11 shows the internal core clock switching frequency across a range of CLKIN frequencies. The minimum operational range for any given frequency is constrained by the operating range of the PLL. Note that the goal in selecting a particular clock ratio for the processor application is to provide the highest internal frequency, given a CLKIN frequency.

If an external master clock 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, there should be 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 100 µs depending on the choice of crystal, operating frequency, loop gain and capacitor ratios. For details on timing, refer to the product specific data sheet.

After the external processor $\overline{\text{RESET}}$ signal is deasserted, the PLL starts operating. The rest of the chip will be held in reset for 4096 CLKIN cycles after $\overline{\text{RESET}}$ is deasserted by an internal reset signal. This sequence allows the PLL to lock and stabilize. Add one CLKIN cycle if $\overline{\text{RESET}}$ doesn't meet setup requirements with respect to the CLKIN falling edge.

Table 9-11. Selecting Core to CLKIN Ratio

	Typical Crystal and Clock Oscillators Inputs					
	12.5	16.67	25	33.3	40	50
Clock Ratios	Core CLK (MHz)					
3:1	37.5	50	75	100	120	150
8:1	100	133.36	200	N/A	N/A	N/A
16:1	200	N/A	N/A	N/A	N/A	N/A

Reset Generators

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 supervisory circuits from Analog Devices are:

- ADM69x
- ADM70x
- ADM80x
- ADM1232
- ADM181x
- ADM869x

A simple power-up reset circuit is shown below, 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 µs active reset delay is generated to give the power supplies and oscillators time to stabilize.

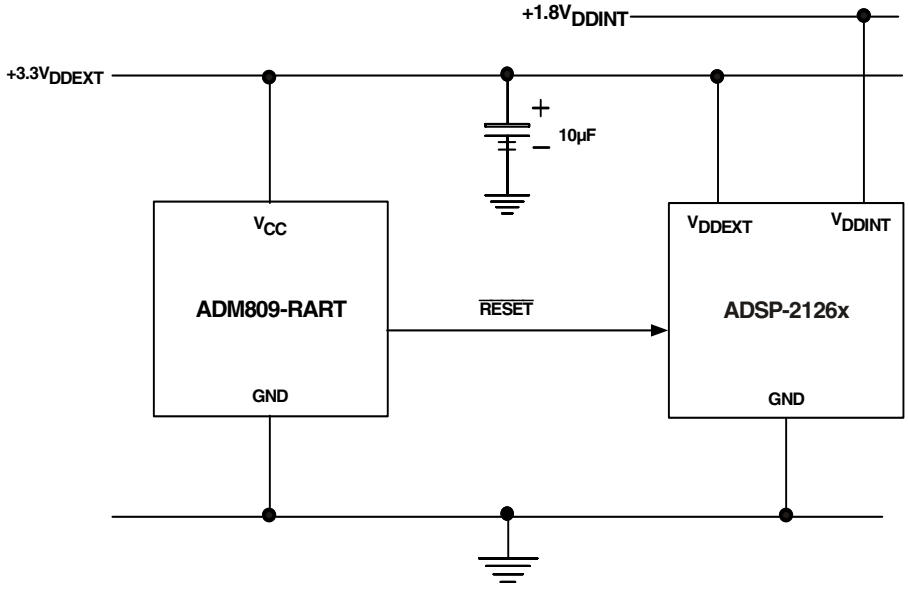


Figure 9-2. Simple Reset Generator

Another part, the ADM706TAR, provides power on $\overline{\texttt{RESET}}$ and optional manual $\overline{\texttt{RESET}}$. 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 eight-lead SOIC package. Figure 9-3 shows a typical application circuit using the ADM706TAR.

Interrupt and Timer Pins

The processor's external interrupt pins, flag pins, and timer pin can be used to send and receive control signals to and from other devices in the system. Hardware interrupt signals $\overline{\texttt{IRQ2-0}}$ are received on the FLG2-0 pins and the TIMEXP pin is mapped on the FLG3 pin. Hardware interrupt signals

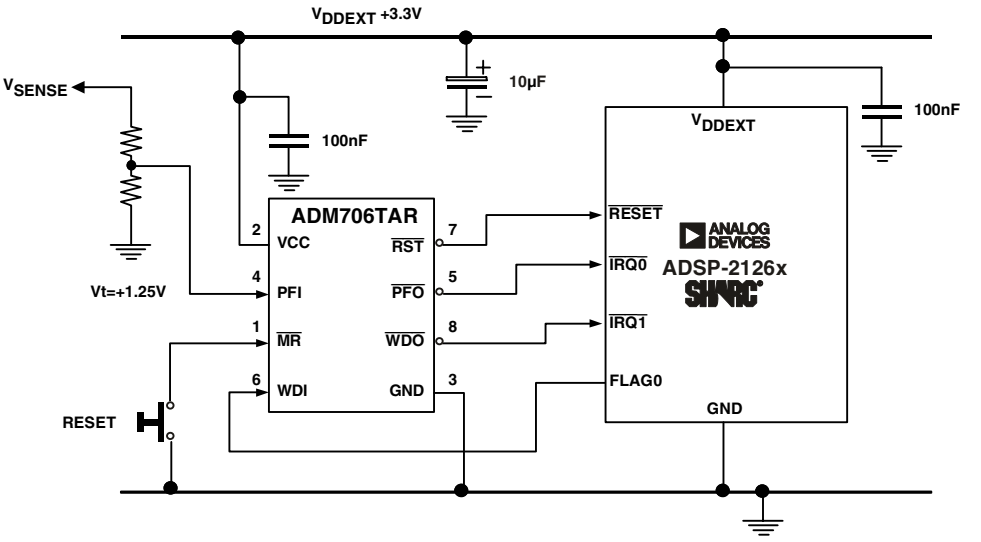


Figure 9-3. Reset Generator and Power Supply Monitor

($\overline{\texttt{IRQ2-0}}$) are received on the FLG2-0 pins. Interrupts can come from devices that require the processor to perform some task on demand. A memory-mapped peripheral, for example, can use an interrupt to alert the processor that it has data available. For more information, see the *ADSP-2126x SHARC DSP Core Manual*.

The TIMEXP output is generated by the on-chip timer. It indicates to other devices that the programmed time period has expired. For more information, see the *ADSP-2126x SHARC DSP Core Manual*.

Core-Based Flag Pins

The FLG3-0 pins allow single bit signalling between the processor and other devices. For example, the processor can raise an output flag to interrupt a host processor. Each flag pin can be programmed to be either an

input or output. In addition, many instructions can be conditioned on a flag's input value, enabling efficient communication and synchronization between multiple processors or other interfaces.

The flags are bidirectional pins and all have the same functionality. The FLGxO bits in the FLAGS register program the direction of each flag pin. For more information, see the *ADSP-2126x SHARC DSP Core Manual*.

When the SPIPDN bit (bit 30 in the PMCTL register) is set (= 1 which shuts down the clock to the SPI), the FLGx pins cannot be used (via the FLGS7-0 register bits) because the FLGx pins are synchronized with the clock.

JTAG Interface Pins

The JTAG Test Access Port (TAP) consists of the TCK, TMS, TDI, TDO, and $\overline{\text{TRST}}$ pins. The JTAG port can be connected to a controller that performs a boundary scan for testing purposes. This port is also used by the Analog Devices Tools product line of JTAG emulator and development software to access on-chip emulation features. To allow the use of the emulator, a connector for its in-circuit probe must be included in the target system.

If the $\overline{\text{TRST}}$ pin is not asserted (or held low) at power-up, the JTAG port is in an undefined state that may cause the processor to drive out on I/O pins that would normally be three-stated at reset. The $\overline{\text{TRST}}$ pin can be held low with a jumper to ground on the target board connector.

A detailed discussion of JTAG and its use can be found in the Engineer-to-Engineer Note (EE-68), *Analog Devices JTAG Emulation Technical Reference*. This document is available on the Analog Devices Web site at www.analog.com.

Phase-Locked Loop Startup

The RESET signal can be held low long enough to guarantee a stable CLKIN source and stable VDDINT/VDDEXT power supplies before the PLL is reset.

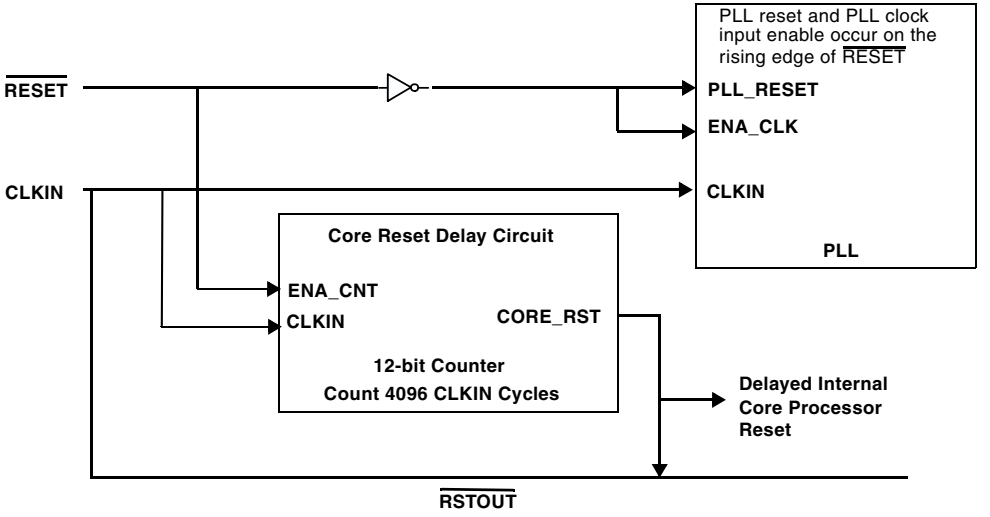


Figure 9-4. Chip Reset Circuit

In order for the PLL to lock to the CLKIN frequency, the PLL needs time to lock before the core can execute or begin the boot process. A delayed core reset has been added via the delay circuit. There is a 12-bit counter that counts up to 4096 CLKIN cycles after RESET is transitioned from low to high. The delay circuit is activated at the same time the PLL is taken out of reset.

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 brown-out situation, the watchdog circuit only has to control the RESET.

Conditioning Input Signals

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.

A typical CMOS input consists of an inverter with specific N and P device sizes that cause a switching point of approximately 1.4 V. This level is selected to be the midpoint of the standard TTL interface specification of V_{IL} = 0.8 V and V_{IH} = 2.0 V. This input inverter, unfortunately, has a fast response to input signals and external glitches wider than 1 ns. Filter circuits and hysteresis are added after the input inverter on some processor inputs, as described in the following sections.

RESET Input Hysteresis

Hysteresis is used only on the $\overline{\text{RESET}}$ input signal. Hysteresis causes the switching point of the input inverter to be slightly above 1.4 V for a rising edge and slightly below 1.4 V for a falling edge. The value of the hysteresis is approximately ± 0.1 V. The hysteresis is intended to prevent multiple triggering of signals which are allowed to rise slowly, as might be expected 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 allowable is due to the restrictions on the tolerance of the V_{IL} and V_{IH} TTL input levels under worst case conditions. Refer to the product specific processor data sheet for exact specifications.

Designing for High Frequency Operation

Because the processor can operate at very fast 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 processor systems.

All synchronous behavior is specified to `CLKIN`. System designers are encouraged to clock synchronous peripherals with this same clock source (or a different low-skew output from the same clock driver).

Clock Specifications and Jitter

The clock 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.

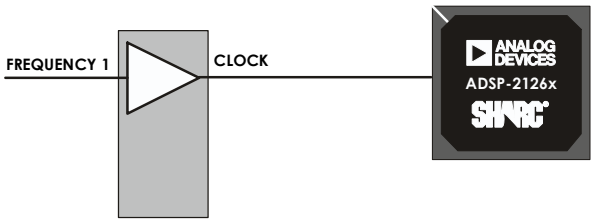


Figure 9-5. Reducing Clock Jitter and Ring

Never share a clock buffer IC with a signal of a different clock frequency. This introduces excessive jitter.

As shown in Figure 9-5, 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 rise time of 3 ns or less and must meet or exceed a high and low voltage of 2 V and 0.4 V, respectively.

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 and away from or layout 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.
- Use 3.3 V peripheral components and power supplies to help reduce transmission line problems, ground bounce and noise coupling (the receiver switching voltage of 1.5 V is close to the middle of the voltage swing).
- 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.

Decoupling Capacitors and Ground Planes

Ground planes must be used for the ground and power supplies. Designs should use a minimum of eight bypass capacitors (six 0.1 µF and two 0.01 µF ceramic). The capacitors should be placed very close to the `VDDEXT` and `VDDINT` pins of the package as shown in Figure 9-6. Use short and fat traces for this. The ground end of the capacitors should be tied directly to the ground plane inside the package footprint of the processor (underneath, on the bottom of the board), not outside the footprint. A

surface-mount capacitor is recommended because of its lower series inductance. Connect the power plane to the power supply pins directly with minimum trace length. The ground planes must not be densely perforated with vias or traces as their effectiveness is reduced. In addition, there should be several large tantalum capacitors on the board.

Designs can use either bypass placement case shown in Figure 9-6, or combinations of the two. Designs should try to minimize signal feedthroughs that perforate the ground plane.

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 3 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

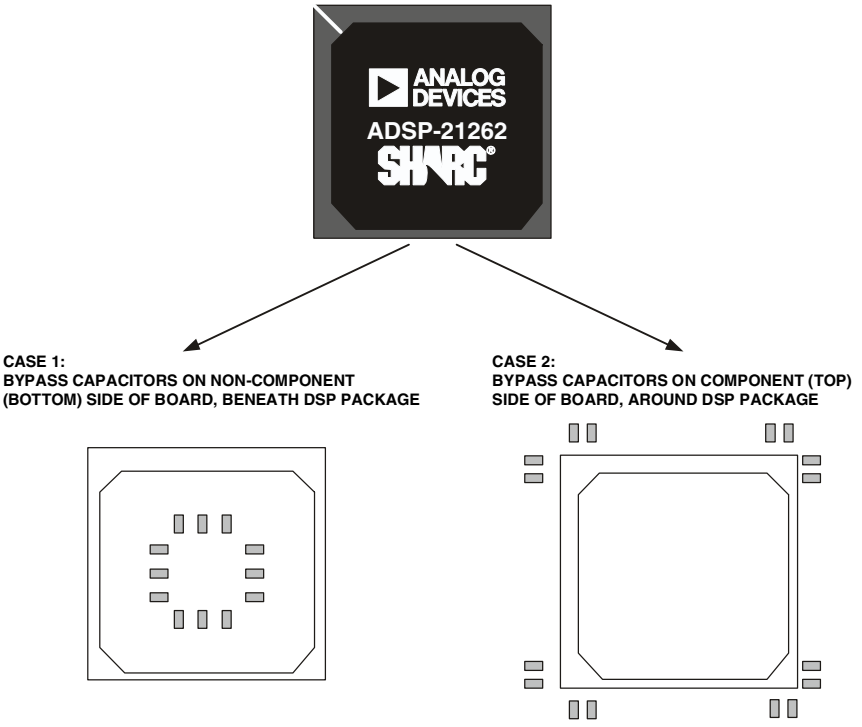


Figure 9-6. Bypass Capacitor Placement

- 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.

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 ADSP-2126x processor processor after power-up or after a software reset.

The ADSP-2126x processor supports three booting modes—EPROM, SPI master and SPI slave. Each of these modes uses the following general procedure:

1. At reset, the ADSP-2126x processor is hardwired to load two hundred fifty-six 32-bit instruction words via a DMA starting at location 0x80000. In this chapter, these instructions are referred to as the *boot kernel* or *loader kernel.*
2. The DMA completes and the interrupt associated with the peripheral that the processor is booting from is activated. The processor jumps to the applicable interrupt vector location (0x80030 for SPI and 0x80050 for the parallel port) and executes the code located there. (Typically, the first instruction at the interrupt vector is a Return From Interrupt (RTI) instruction.)
3. The loader kernel executes a series of Direct Memory Accesses (DMAs) to import the rest of the application, overwriting itself with the applications' Interrupt Vector Table (IVT).
4. After executing the kernel, the processor returns to location 0x80005 where normal program execution begins.

To support this process, a 256-word loader kernel and loader (which converts executables into boot-loader images) are supplied with the VisualDSP++ development tools for both SPI and parallel port booting. For more information on the loader, see the tools documentation in "Processor Product Information" on page -xxv.

The boot source is determined by strapping the two BOOTCFGx pins to either logic low or logic high. These settings are shown in Table 9-12.

Table 9-12. Booting Modes

BOOT_CFG1-0	Description
00	SPI Slave boot
01	SPI Master boot
10	EPROM boot via parallel port
11	Internal Boot mode (not available on all ADSP-2126x processors)

Parallel Port Booting

The ADSP-2126x processor supports an 8-bit boot mode through the parallel port. This mode is used to boot from external 8-bit wide memory devices. The processor is configured for 8-bit boot mode when the BOOT_CFG1-0 pins = 10. When configured for parallel boot loading, the parallel port transfers occur with the default bit settings (shown in Table 9-13) for the PPCTL register.

Table 9-13. Parallel Port Boot Mode Settings in the PPCTL Register

Bit	Setting
PPALEPL	= 0; ALE is active high
PPEN	= 1

Table 9-13. Parallel Port Boot Mode Settings in the PPCTL Register (Cont'd)

Bit	Setting
PPDUR	= 10111; (23 core clock cycles per data transfer cycle)
PPBHC	= 1; insert a bus hold cycle on every access
PP16	= 0; external data width = 8 bits
PPDEN	= 1; use DMA
PPTRAN	= 0; receive (read) DMA
PPBHD	= 0; buffer hang enabled

For a complete description of the Parallel Port Control register, see "Parallel Port Control Register (PPCTL)" on page A-55.

The parallel port 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 9-14.

Unlike previous SHARC processors, the ADSP-2126x processor does not have a Boot Memory Select ($\overline{\text{BMS}}$) pin.

Table 9-14. Parameter Initialization Value

Parameter Register	Initialization Value	Comment
PPCTL	0x0000 016F	See Table 9-13.
IIPP	0	This is the offset from internal memory normal word starting address of 0x80000.
ICPP	0x180 (384)	This is the number of 32-bit words that are equivalent to 256 instructions.
IMPP	0x01	
EIPP	0x00	

Table 9-14. Parameter Initialization Value (Cont'd)

Parameter Register	Initialization Value	Comment
ECPP	0x600	This is the number of bytes in 0x100 48-bit instructions.
EMPP	0x01	

SPI Port Booting

The ADSP-2126x processor supports booting from a host processor via SPI Slave mode (BOOT_CFG1-0 = 00), and booting from an SPI Flash, SPI PROM, or a host processor via SPI Master mode (BOOT_CFG1-0 = 01).

Both SPI boot modes support booting from 8-, 16-, or 32-bit SPI devices. In all SPI boot mode, 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 Figure 9-7.

For more information about 32- and 48-bit internal memory addressing, see the "Memory" chapter in the *ADSP-2126x SHARC Processor Core Manual*.

For 16-bit SPI devices, two words shift into the 32-bit receive Shift register (RXSR) before a DMA transfer to internal memory occurs. For 8-bit SPI devices, four words shift into the 32-bit receive shift register before a DMA transfer to internal memory occurs.

When booting, the ADSP-2126x processor expects to receive words into the RXSPI register seamlessly. This means that bits are received continuously without breaks. For more information, see "Core Transmit and Receive Operations" on page 5-12. For different SPI host sizes, the processor expects to receive instructions and data packed in a least significant word (LSW) format.

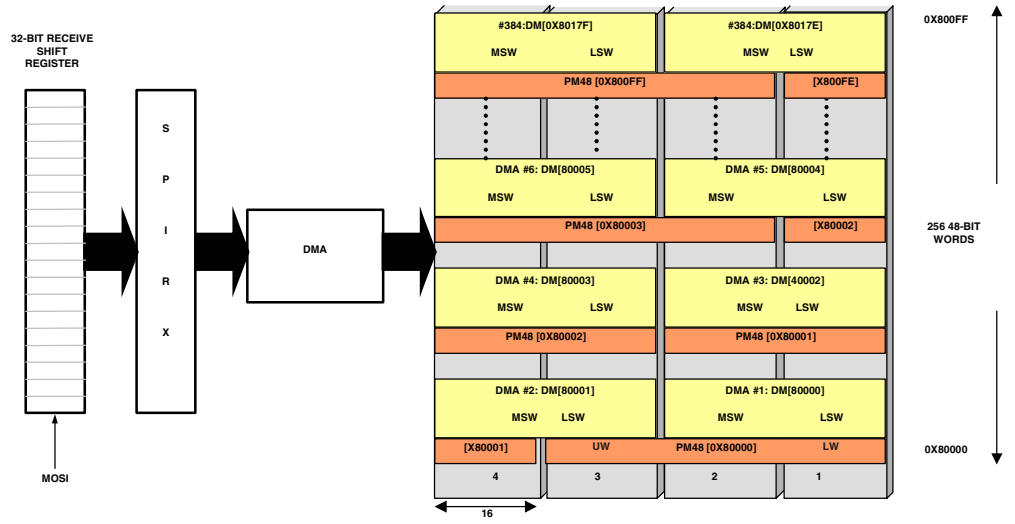


Figure 9-7. SPI Data Packing

Figure 9-8 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 as illustrated in Figure 9-7.

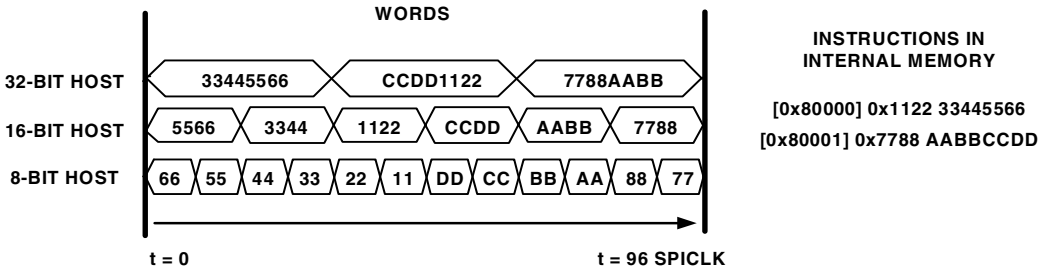


Figure 9-8. Instruction Packing for Different Hosts

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 Host Boot

Figure 9-9 shows 32-bit SPI host packing of 48-bit instructions executed at PM addresses 0x80000 and 0x80001. The 32-bit word is shifted to internal program memory during the 256-word kernel load.

The following example shows a 48-bit instructions executed:

```
[0x80000] 0x112233445566

[0x80001] 0x7788AABBCCDD
```

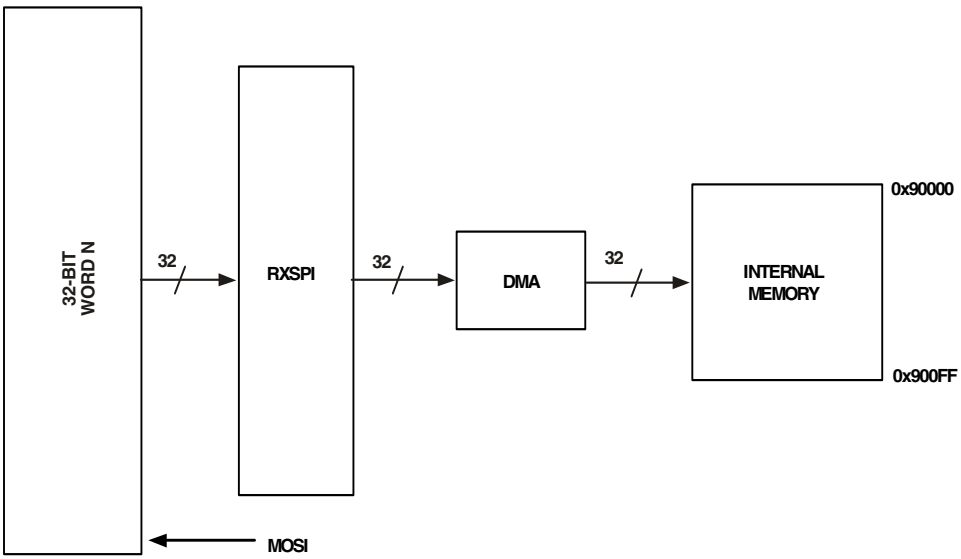


Figure 9-9. 32-Bit SPI Host Packing

The 32-bit SPI host packs or prearranges the data as:

SPI word 1= 0x33445566

SPI word 2 = 0xCCDD1122

SPI word 3 = 0x7788AABB

16-bit SPI Host Boot

Figure 9-10 shows how a 16-bit SPI host packs 48-bit instructions at PM addresses 0x80000 and 0x80001. 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 example shows a 48-bit instructions executed.

```
[0x80000] 0x112233445566

[0x80001] 0x7788AABBCCDD
```

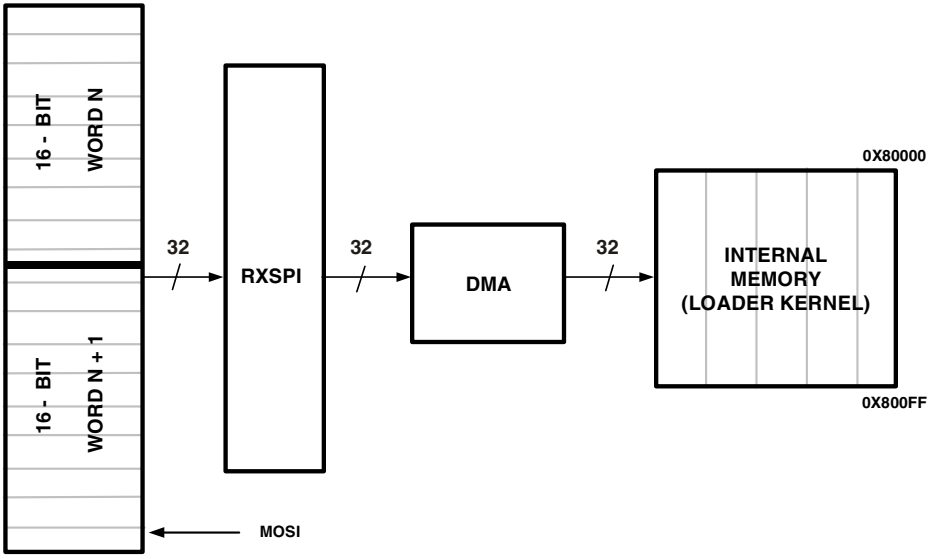


Figure 9-10. 16-Bit SPI Host Packing

The 16-bit SPI host packs or prearranges the data as:

SPI word 1 = 0x5566SPI word 2 = 0x3344

SPI word 3 = 0x1122SPI word 4 = 0xCCDD

SPI word 5 = 0xAABBSPI 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 Host Boot

Figure 9-11 shows 8-bit SPI host packing of 48-bit instructions executed at PM addresses 0x80000 and 0x80001. 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 example shows a 48-bit instructions executed.

```
[0x80000] 0x112233445566

[0x80001] 0x7788AABBCCDD
```

The 8-bit SPI host packs or prearranges the data as:

SPI word 1 = 0x66SPI word 2 = 0x55

SPI word 3 = 0x44SPI word 4 = 0x33

SPI word 5 = 0x22SPI word 6 = 0x11

SPI word 7 = 0xDDSPI word 8 = 0xCC

SPI word 9 = 0xBBSPI word 10 = 0xAA

SPI word 11 = 0x88SPI word 12 = 0x77

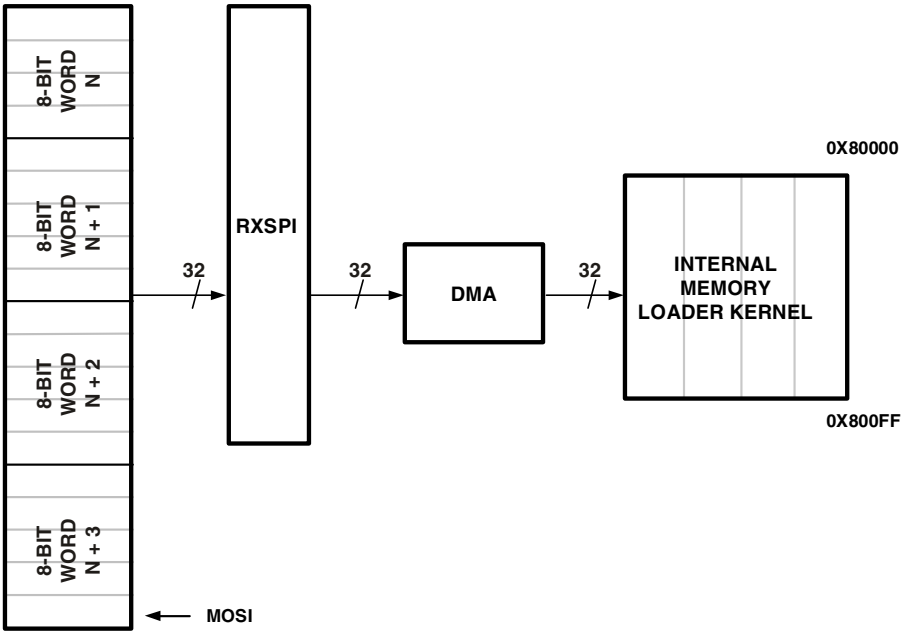


Figure 9-11. 8-Bit SPI Host Packing

The initial boot of the 256-word loader kernel requires an 8-bit host to transmit fifteen hundred thirty-six 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.

For all boot modes, the VisualDSP++ loader automatically outputs the correct word width and count based on the project settings. For more information, see the VisualDSP++ tools documentation.

Slave Boot Mode

In Slave boot mode, the host processor initiates the booting operation by activating the SPICLK signal and asserting the $\overline{\text{SPIDS}}$ signal to the active low state. The 256-word kernel is loaded 32 bits at a time, via 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.

The processor's $\overline{\text{SPIDS}}$ pin should not be tied low. When in SPI Slave mode, including booting, the $\overline{\text{SPIDS}}$ signal is required to transition from high to low. SPI slave booting uses the default bit settings shown in Table 9-15.

Table 9-15. SPI Slave Boot Bit Settings

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 9-16.

Table 9-16. Parameter Initialization Value for Slave Boot

Parameter Register	Initialization Value	Comment
SPICTL	0x0000 4D22	
SPIDMAC	0x0000 0007	Enabled, RX, initialized on completion
IISPI	0x0008 0000	Start of Block 0 NW memory
IMSPI	0x0000 0001	32-bit data transfers
CSPI	0x0000 0180	

Master Boot

In Master Boot mode, the ADSP-2126x processor initiates the booting operation by:

1. Activating the SPICLK signal and asserting the FLG0 signal to the active low state.
2. Writing the read command 0x03 and address 0x00 to the slave device as shown in Figure 9-8.

Master Boot mode is used when the processor is booting from an SPI compatible serial PROM, serial FLASH, or slave host processor. The specifics of booting from these devices are discussed individually. On reset, the interface starts up in Master mode performing a three hundred eighty-four 32-bit word DMA transfer.

SPI master booting uses the default bit settings shown in Table 9-17.

Table 9-17. SPI Master Boot Mode Bit Settings

Bit	Setting	Comment
SPIEN	Set (= 1)	SPI Enabled
SPIMS	Set (= 1)	Master device
MSBF	Cleared (= 0)	LSB first

Table 9-17. SPI Master Boot Mode Bit Settings (Cont'd)

Bit	Setting	Comment
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 9-18.

Table 9-18. Parameter Initialization Value for Master Boot

Parameter Register	Initialization Value	Comment
SPICTL	0x0000 5D06	
SPIBAUD	0x0064	CCLK/400 =500 KHz@ 200 MHz
SPIFLG	0xfe01	FLG0 used as slave-select
SPIDMAC	0x0000 0007	Enable receive interrupt on completion
IISPI	0x0008 0000	Start of block 0 normal word memory
IMSPI	0x0000 0001	32-bit data transfers
CSPI	0x0000 0180	0x100 instructions = 0x180 32-bit words

From the perspective of the ADSP-2126x processor, there is no difference between booting from the three types of SPI slave devices. Since SPI is a full-duplex protocol, the processor is receiving the same amount of bits that it sends as a read command. The read command comprises a full 32-bit word (which is what the processor is initialized to send) comprised of a 24-bit address with an 8-bit opcode. The 32-bit word that is received

while this read command is transmitted is thrown away in hardware, and can never be recovered by the user. Because of this, special measures must be taken to guarantee that the boot stream is identical in all three cases. The processor boots in Least Significant Bit First (LSBF) format, while most serial memory devices operate in Most Significant Bit First (MSBF) format. Therefore, it is necessary to program the device in a fashion that is compatible with the required LSBF format.

Also, because the processor always transmits 32 bits before it begins reading boot data from the slave device, it is necessary for the VisualDSP++ tools to insert extra data to the boot image (in the loader file) if using memory devices that do not use the LSBF format. VisualDSP++ has built-in support for creating a boot stream compatible with both endian formats, and devices requiring 16-bit and 24-bit addresses, as well as those requiring no read command at all.

Figure 9-12 shows the initial 32-bit word sent out from the processor. As shown in the figure, the processor initiates the SPI-Master boot process by writing an 8-bit opcode (LSB first) to the slave device to specify a read operation. This read opcode is fixed to 0xC0 (0x03 in MSBF format). Following that, a 24-bit address (all zeros) is always driven by the processor. On the following SPICLK cycle (cycle 32), the processor expects the first bit of the first word of the boot stream. This transfer continues until the kernel has finished loading the user program into the processor.

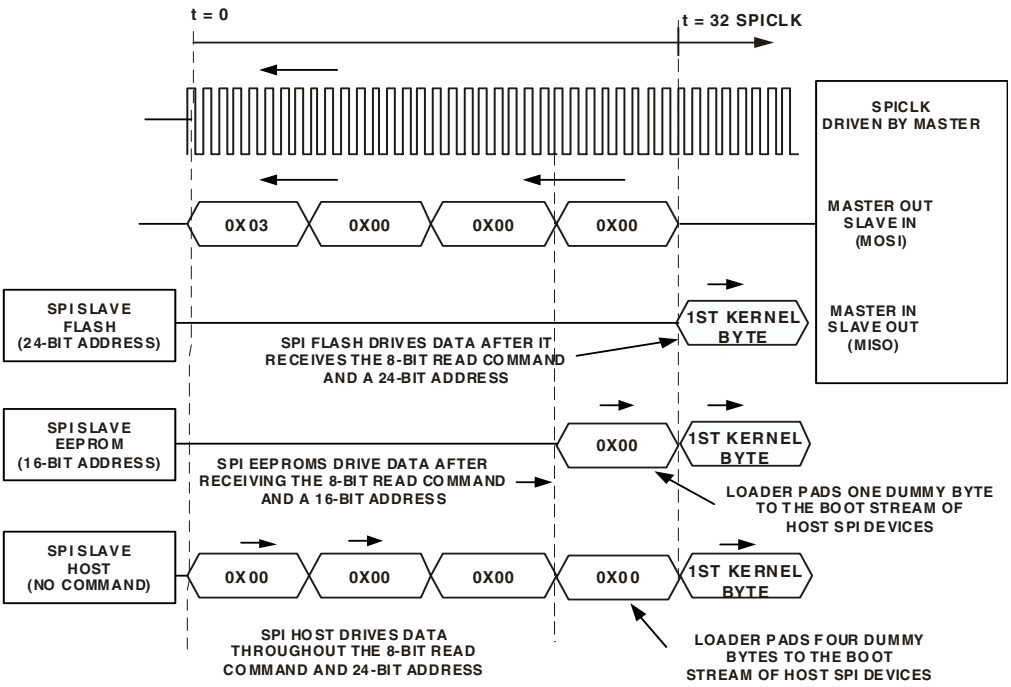


Figure 9-12. SPI Master Mode Booting Using Various Serial Devices

Booting From an SPI Flash

For SPI flash devices, the format of the boot stream will be identical to that used in SPI Slave mode, with the first byte of the boot stream being the first byte of the kernel. This is because SPI flash devices do not drive out data until they receive an 8-bit command and a 24-bit address.

Booting From an SPI PROM (16-bit address)

Figure 9-12 shows the initial 32-bit word sent out from the processor from the perspective of the serial PROM device.

As shown in Figure 9-12, SPI EEPROMS only require an 8-bit opcode and a 16-bit address. These devices begin transmitting on clock cycle 24. However, because the processor is not expecting data until clock cycle 32, it is necessary to pad an extra byte to the beginning of the boot stream when programming the PROM. In other words, the first byte of the kernel will be the second byte of the boot stream. The VisualDSP++ tools will automatically handle this in the loader file generation process for SPI PROM devices.

Booting From an SPI Host Processor

Typically, host processors in SPI Slave mode transmit data on every `SPICLK` cycle. This means that the first four bytes that are sent by the host processor will be part of the first 32-bit word that is thrown away by the processor (see Figure 9-12). Therefore, it is necessary to pad an extra four bytes to the beginning of the boot stream when programming the host, for example, the first byte of the kernel will be the fifth byte of the boot stream. VisualDSP++ will automatically handle this in the loader file generation process.

Data Delays, Latencies, and Throughput

Table 9-19 on page 9-43 specifies latencies and throughput for the ADSP-2126x processor. *Latency* is defined as the number cycles after the first cycle required to complete the operation. A zero wait state memory has a latency of zero. A single wait state memory has a data delay of one. *Throughput* is the maximum rate at which the operation is performed. Data delay and throughput are the same whether the access is from a host processor or from another ADSP-2126x processor.

Execution Stalls

The following events can cause an execution stall for the ADSP-2126x processor:

- One cycle on a Program Memory Data Access with instruction cache miss
- Two cycles on non-delayed branches
- Two cycles on normal interrupts
- One to two cycles on short loops with small iterations
- n cycles on an `IDLE` instruction
- In a sequence of three instructions of the types shown below, the processor may stall for one cycle:

  Instruction 1: Compute instruction affecting flags such as `R2 = R3 - R4;`

  Instruction 2: Conditional instruction involving post-modify addressing such as `IF EQ DM(I1,M1) = R15;`

  Instruction 3: Instruction such as `R0 = DM(I1,M2);` involving post-modify addressing involving same `I` register. This last instruction stalls the processor for one cycle.
- Any read reference to a memory-mapped register located physically within core (registers like `SYSCTL`, which are not situated in the IOP) requires two cycles; therefore, the processor stalls for one cycle.
- Any read reference to a memory-mapped register located within a peripheral such as the SPI, SPORTS, IDP, or parallel port requires a minimum of four cycles; so the minimum stall is three cycles.

- Any reference to a memory-mapped register in a conditional instruction stalls the processor for one extra cycle (with respect to an unconditional instruction).

DAG Stalls

One cycle hold on register conflict.

Memory Stalls

One cycle on PM and DM bus access to the same block of internal memory.

IOP Register Stalls

Read of the IOP registers takes a minimum of four cycles, therefore the processor stalls for at least three cycles.

DMA Stalls

The following events can cause a DMA stall for the ADSP-2126x processor:

- One cycle stall if an access to a DMA Parameter register conflicts with the DMA address generation. For example, writing to or reading from a DMA Parameter register while a register update is taking place conflicts with DMA chaining.
- n cycles if writing (or reading) to a DMA buffer when the buffer is full (or empty).

IOP Buffer Stalls

The following event can cause an IOP buffer stall for the ADSP-2126x processor—n cycles if a program writes to a full buffer or reads from an empty buffer.

Table 9-19. Latencies and Throughput

Operation	Minimum Data Delay (cycles)	Maximum Throughput (cycles/ transfer)
Interrupts ($\overline{\text{IRQ}}$2-0)	3	-
DMA chain initialization	7–11	-
Serial ports[1]	35	32

1 ADSP-2126x processor to ADSP-2126x processor transfers using 32-bit words.

A REGISTERS REFERENCE

The ADSP-2126x processor processor has general-purpose and dedicated registers in each of its functional blocks. The register reference information for each functional block includes bit definitions, initialization values, and memory-mapped addresses (for IOP registers). Information on each type of register is available at:

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 provides a header file that provides these bit and registers definitions. An `#include` file is provided with VisualDSP++ tools and can be found in the `VisualDSP/2126x/include` directory.

Many registers have reserved bits. When writing to a register, programs may only clear (write zero to) the register's reserved bits.

I/O Processor Registers

The I/O Processor (IOP) registers are accessible as part of the processor's memory map. Table A-1 on page A-3 lists the IOP memory-mapped registers and provides a cross-reference to a description of each register. These registers occupy addresses 0x0000 0000 through 0x0003 FFFF of the memory map. The IOP registers control the following DMA operations: Parallel port, Serial port, Serial Peripheral Interface port (SPI), and Input Data Port (IDP).

IOP registers have a one cycle effect latency (changes take effect on the second cycle after the change).

Since the IOP registers are part of the processor's memory map, buses access these registers as locations in memory. While these registers act as memory-mapped locations, they are separate from the processor's internal memory and have different bus access. One bus can access one IOP register from one IOP register group at a time. Table A-1 on page A-3 lists the IOP register groups.

When there is contention among the buses for access to registers in the same IOP register group, the processor arbitrates register access as:

- Data Memory (DM) bus accesses
- Program Memory (PM) bus accesses
- IOP (IO) bus (lowest priority) accesses

The bus with highest priority gets access to the IOP register group, and the other buses are held off from accessing that I/O processor register group until that access been completed.

Table A-1. I/O Processor Register Groups

Register Group	IOP Registers In This Group
System Control (SC) Registers	SYSCTL, REVPID, EEMUIN, EEMUSTAT, EEMUOUT, OSPID, BRKCTL, PSA1S, PSA1E, PSA2S, PSA2E, PSA3S, PSA3E, PSA4S, PSA4E, DMA1S, DMA1E, DMA2S, DMA2E, PMDAS, PMDAE, EMUN, IOAS, IOAE
Parallel Port (PP) Registers	PPCTL, RXPP, TXPP, EIPP, EMPP, ECPP, IIPP, IMPP, ICPP
Serial Peripheral Interface (SPI) Registers	RXSPI, SPIFLG, TXSPI, SPICTL, SPISTAT, SPIBAUD, SPIDMAC, IISPI, IMSPI, RXSPI_SHADOW, CPSPI, CSPI
Timer Registers	TM0STAT, TM0CTL, TM0CNT, TM0PRD, TM0W, TM1STAT, TM1CTL, TM1CNT, TM1PRD, TM1W, TM2STAT, TM2CTL, TM2CNT, TM2PRD, TM2W
Power Management Registers	PMCTL

Table A-1. I/O Processor Register Groups (Cont'd)

Register Group	IOP Registers In This Group
Serial Port (SP) Registers	IISP0A, IISP0B, IMSP0A, IMSP0B, CSP0A, CSP0B, CPSP0A, CPSP0B, IISP1A, IISP1B, IMSP1A, IMSP1B, CSP1A, CSP1B, CPSP1A, CPSP1B, IISP2A, IISP2B, IMSP2A, IMSP2B, CSP2A, CSP2B, CPSP2A, CPSP2B, IISP3A, IISP3B, IMSP3A, IMSP3B, CSP3A, CSP3B, CPSP3A, CPSP3B, IISP4A, IISP4B, IMSP4A, IMSP4B, CSP4A, CSP4B, CPSP4A, CPSP4B, IISP5A, IISP5B, IMSP5A, IMSP5B, CSP5A, CSP5B, CPSP5A, CPSP5B RXSP0A, RXSP0B, TXSP0A, TXSP0B, SPCTL0, DIV0, SPCNT0, MT0CS0, MT0CCS0, MT0CS1, MT0CCS1, MT0CS2, MT0CCS2, MT0CS3, MT0CCS3 RXSP1A, RXSP1B, TXSP1A, TXSP1B, SPCTL1, DIV1, SPCNT1, MR1CS0, MR1CCS0, MR1CS1, MR1CCS1, MR1CS2, MR1CCS2, MR1CS3, MR1CCS3 RXSP2A, RXSP2B, TXSP2A, TXSP2B, SPCTL2, DIV2, SPCNT2, MT2CS0, MT2CCS0, MT2CS1, MT2CCS1, MT2CS2, MT2CCS2, MT2CS3, MT2CCS3 RXSP3A, RXSP3B, TXSP3A, TXSP3B, SPCTL3, DIV3, SPCNT3, MR3CS0, MR3CCS0, MR3CS1, MR3CCS1, MR3CS2, M3CCS2, MR3CCS2, MR3CS3, MR3CCS3 RXSP4A, RXSP4B, TXSP4A, TXSP4B, SPCTL4, DIV4, SPCNT4, MT4CS0, MT4CCS0, MT4CS1, MT4CCS1, MT4CS2, MT4CCS2, MT4CS3, MT4CCS3 RXSP5A, RXSP5B, TXSP5A, TXSP5B, SPCTL5, DIV5, SPCNT5, MR5CS0, MR5CCS0, MR5CS1, MR5CCS1, MR5CS2, MR5CCS2, MR5CS3, MR5CCS3 SPMCTL01, SPMCTL23, SPMCTL45

Table A-1. I/O Processor Register Groups (Cont'd)

Register Group	IOP Registers In This Group
DAI Registers	SRU_CLK0, SRU_CLK1, SRU_CLK2, SRU_CLK3 SRU_DAT0, SRU_DAT1, SRU_DAT2, SRU_DAT3, SRU_DAT4 SRU_FS0, SRU_FS1, SRU_FS2 SRU_PIN0, SRU_PIN1, SRU_PIN2, SRU_PIN3 SRU_EXT_MISCA, SRU_EXT_MISCB SRU_PBEN0, SRU_PBEN1, SRU_PBEN2, SRU_PBEN3 PCG_CTLA_0, PCG_CTLA_1, PCG_CTLB_0, PCG_CTLB_1, PCG_PW IDP_CTL, DAI_STAT, IDP_FIFO, IDP_DMA_I0, IDP_DMA_I1, IDP_DMA_I2, IDP_DMA_I3, IDP_DMA_I4, IDP_DMA_I5, IDP_DMA_I6, IDP_DMA_I7, IDP_DMA_M0, IDP_DMA_M1, IDP_DMA_M2, IDP_DMA_M3, IDP_DMA_M4, IDP_DMA_M5, IDP_DMA_M6, IDP_DMA_M7, IDP_DMA_C0, IDP_DMA_C1, IDP_DMA_C2, IDP_DMA_C3, IDP_DMA_C4, IDP_DMA_C5, IDP_DMA_C6, IDP_DMA_C7, IDP_PP_CTL DAI_PIN_PULLUP, DAI_PIN_STAT, DAI_IRPTL_H, DAI_IRPTL_L, DAI_IRPTL_PRI, DAI_IRPTL_RE, DAI_IRPTL_FE

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 IOP registers, programs must use the processor core registers.

The register names for IOP 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 as described on page A-2.

Flag Value Register (FLAGS)

The FLAGS register is a non-memory-mapped, universal, system register (*Ureg* and *Sreg*). At reset:

- FLG0 bit is FLAG0 pin value
- FLG1 bit is FLAG1 pin value
- FLG2 bit is FLAG2 pin value
- FLG3 bit is FLAG3 pin value
- Other FLGx bit values are unknown
- FLGx0 bits are zero

The FLAGS register indicates the state of the FLGx pins. When a FLGx pin is an output, the processor outputs a high in response to a program setting the bit in FLAGS. The I/O direction (input or output) selection of each bit is controlled by its FLGx0 bit in the FLAGS register. The FLAGS register bit definitions are given in Figure A-1.

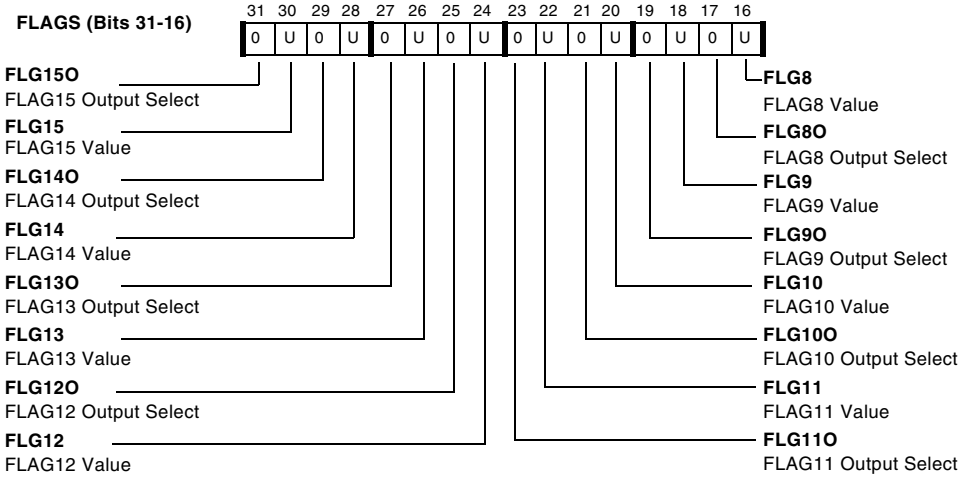
When the flag pins are changed from inputs to outputs, the value that is driven is the value that had been sampled while the pins were inputs.

There are 16 flags in ADSP-2126x processor. All are muxed with other pins. The FLAG[0:3] pins have four dedicated pins. The FLAG[10:15] pins are accessible to the Signal Routing Unit (SRU). All 16 flags are routed to the AD pins when the PPFLGS bit in the SYSCTL register (= 1). While this bit is set, the parallel port is not operational and the four dedicated FLAG[0:3] pins switch to their alternate state—$\overline{\text{IRQ0}}$, $\overline{\text{IRQ1}}$, $\overline{\text{IRQ2}}$, and TMREXP.

When the SPIPDN bit (bit 30 in the PMCTL register) is set (= 1 which shuts down the clock to the SPI), the FLGx pins cannot be used (via the FLGS7-0 register bits) because the FLGx pins are synchronized

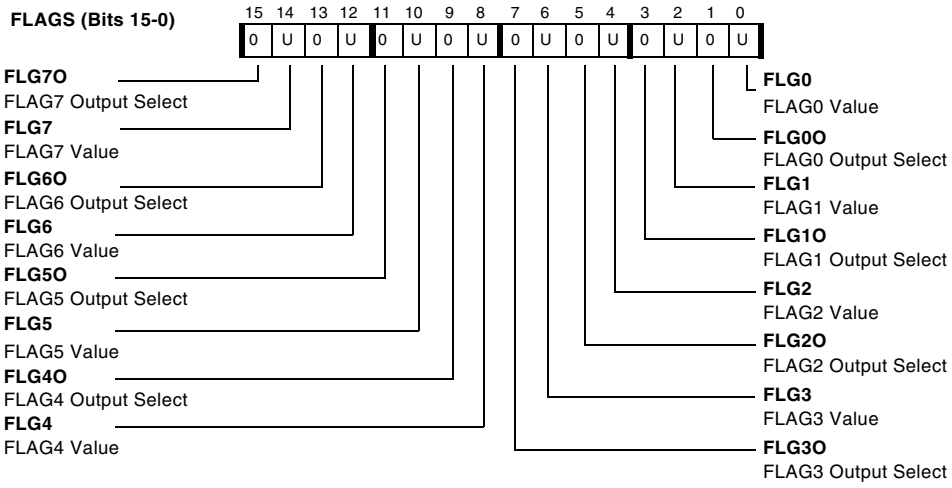
with the clock. For more information, see “Power Management Control Register (PMCTL)” in the *ADSP-2126x SHARC DSP Core Manual*.

Programs cannot change the Output Selects of the FLAGS register and provide a new value in the same instruction. Instead, programs must use two write instructions—the first to change the output select of a particular FLG pin, and the second to provide the new value.



-For all FLGx bits, FLAGx values are as follows: 0=low, 1=high.
-For all FLGxO bits, FLAGx output selects are as follows: 0=FLAGx Input, 1=FLAGx Output.
-U indicates the bit value is unknown at reset.

Figure A-1. FLAGS Register (Upper Bits)



-For all FLGx bits, FLAGx values are as follows: 0=low, 1=high.
-For all FLGxO bits, FLAGx output selects are as follows: 0=FLAGx Input, 1=FLAGx Output.
-U indicates the bit value is unknown at reset.

Figure A-2. FLAGS Register (Lower Bits)

Table A-2. FLAGS Register Bit Descriptions

Bits	Name	Definition
0	FLG0	**FLAG0 Value.** Indicates the state of the FLG0 pin—high if set, (= 1) or low if cleared, (= 0).
1	FLG0O	**FLAG0 Output Select.** Selects the I/O direction for the FLG0 pin, the flag is programmed as an output if set, (= 1) or input if cleared, (= 0).
2	FLG1	**FLAG1 Value.** Indicates the state of the FLG1 pin—high if set, (= 1) or low if cleared, (= 0).
3	FLG1O	**FLAG1 Output Select.** Selects the I/O direction for the FLG1 pin—an output if set, (= 1) or input if cleared, (= 0).
4	FLG2	**FLAG2 Value.** Indicates the state of the FLG2 pin—high if set, (= 1) or low if cleared, (= 0).
5	FLG2O	**FLAG2 Output Select.** Selects the I/O direction for the FLG2 pin—output if set, (= 1) or input if cleared, (= 0).

Table A-2. FLAGS Register Bit Descriptions (Cont'd)

Bits	Name	Definition
6	FLAG3	**FLAG3 Value.** Indicates the state of the FLAG3 pin—high (if set, = = 1) or low if cleared, (= 0).
7	FLG3O	**FLAG3 Output Select.** Selects the I/O direction for the FLAG3 pin—output if set, (= 1) or input if cleared, (= 0).
8	FLG4	**FLAG4 Value.** Indicates the state of the FLAG4 pin—high if set, (= 1) or low if cleared, (= 0).
9	FLG4O	**FLAG4 Output Select.** Selects the I/O direction for the FLAG4 pin—output if set, (= 1) or input if cleared, (= 0).
10	FLG5	**FLAG5 Value.** Indicates the state of the FLAG5 pin—high if set, (= 1) or low if cleared, (= 0).
11	FLG5O	**FLAG5 Output Select.** Selects the I/O direction for the FLAG5 pin—output if set, (= 1) or input if cleared, (= 0).
12	FLG6	**FLAG6 Value.** Indicates the state of the FLAG6 pin—high if set, (= 1) or low if cleared, (= 0).
13	FLG6O	**FLAG6 Output Select.** Selects the I/O direction for the FLAG6 pin—output if set, (= 1) or input if cleared, (= 0).
14	FLG7	**FLAG7 Value.** Indicates the state of the FLAG7 pin—high if set, (= 1) or low if cleared, (= 0).
15	FLG7O	**FLAG7 Output Select.** Selects the I/O direction for the FLAG7 pin—output if set, (= 1) or input if cleared, (= 0).
16	FLG8	**FLAG8 Value.** Indicates the state of the FLAG8 pin—high if set, (= 1) or low if cleared, (= 0).
17	FLG8O	**FLAG8 Output Select.** Selects the I/O direction for FLAG8—output if set, (= 1) or an input if cleared, (= 0).
18	FLG9	**FLAG9 Value.** Indicates the state of the FLAG9 pin—high if set, (= 1) or low if cleared, (= 0).
19	FLG9O	**FLAG9 Output Select.** Selects the I/O direction for FLAG9—output if set, (= 1) or input if cleared, (= 0).
20	FLG10	**FLAG10 Value.** Indicates the state of the FLAG10 pin—high if set, (= 1) or low if cleared, (= 0).

Table A-2. FLAGS Register Bit Descriptions (Cont'd)

Bits	Name	Definition
21	FLG10O	**FLAG10 Output Select.** Selects the I/O direction for FLAG10—output if set, (= 1) or an input if cleared, (= 0).
22	FLG11	**FLAG11 Value.** Indicates the state of the FLAG11 pin—high if set, (= 1) or low if cleared, (= 0).
23	FLG11O	**FLAG11 Output Select.** Selects the I/O direction for the FLAG11—output if set, (= 1) or an input if cleared, (= 0).
24	FLG12	**FLAG12 Value.** Indicates the state of the FLAG12 pin—high if set, (= 1) or low if cleared, (= 0).
25	FLG12O	**FLAG12 Output Select.** Selects the I/O direction for FLAG12—output if set, (= 1) or input if cleared, (= 0).
26	FLG13	**FLAG13 Value.** Indicates the state of the FLAG13 pin—high if set, (= 1) or low if cleared, (= 0).
27	FLG13O	**FLAG13 Output Select.** Selects the I/O direction for FLAG13—output if set, (= 1) or an input if cleared, (= 0).
28	FLG14	**FLAG14 Value.** Indicates the state of the FLAG14 pin—high if set, (= 1) or low if cleared, (= 0).
29	FLG14O	**FLAG14 Output Select.** Selects the I/O direction for FLAG14—output if set, (= 1) or input if cleared, (= 0).
30	FLG15	**FLAG15 Value.** Indicates the state of the FLAG15 pin—high if set, (= 1) or low if cleared, (= 0).
31	FLG15O	**FLAG15 Output Select.** Selects the I/O direction for FLAG15—output if set, (= 1) or input if cleared, (= 0).

System Control Register (SYSCTL)

The SYSCTL register is used to set up system configuration selections. This register's address is 0x30024. The reset value for this register is zero. Bit descriptions for this register are shown in Figure A-3 and described in Table A-3. The reset value has all bits initialized to zero.

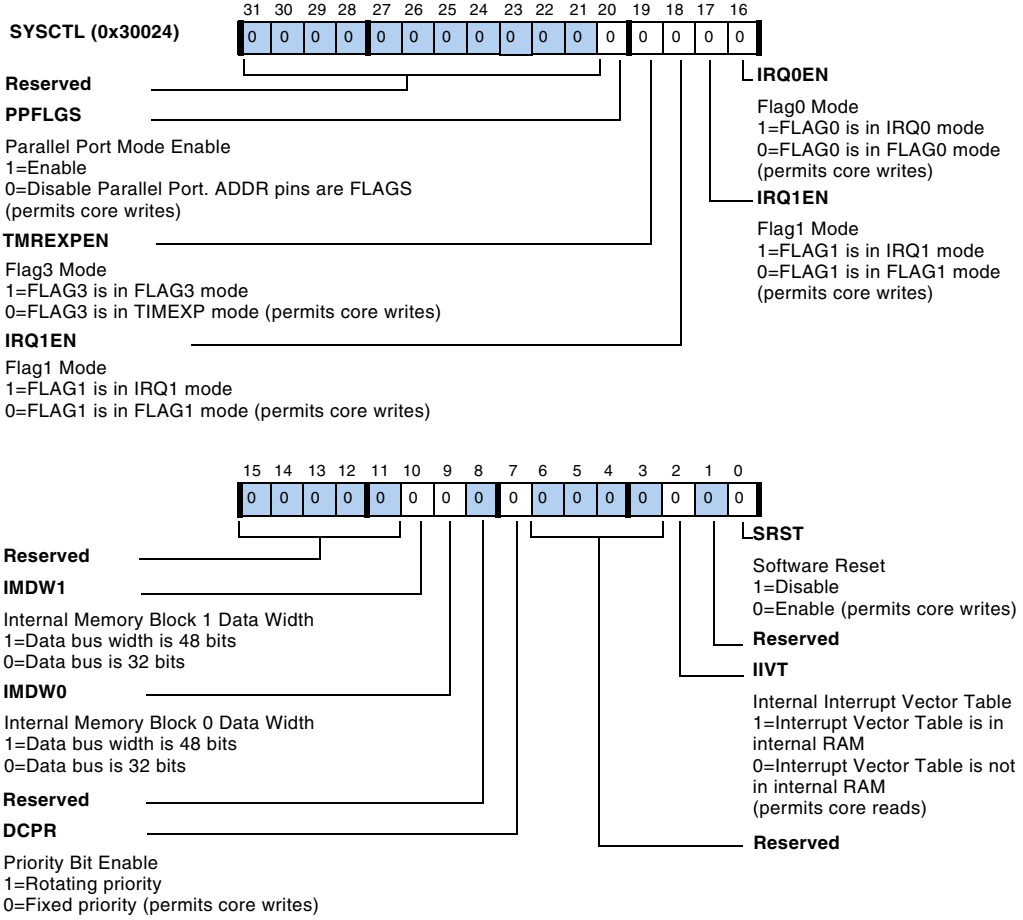


Figure A-3. SYSCTL Register

Table A-3. SYSCTL Register Bit Descriptions

Bits	Name	Definition
0	SRST	**Software Reset.** Resets (when set, = 1) the processor. When a program sets (= 1) SRST, the processor responds to the non-maskable RSTI interrupt and clears (= 0) SRST.
1	Reserved	
2	IIVT	**Internal Interrupt Vector Table.** Forces placement of the interrupt vector table at address 0x0008 0000 regardless of booting mode (if 1) or allows placement of the interrupt vector table as selected by the booting mode (if 0).
6–3	Reserved	
7	DCPR	**DMA Channel Priority Rotation Enable.** Enables (rotates if set, = 1) or disables (fixed if cleared, = 0) priority rotation among DMA channels. Permits core writes.
8	Reserved	
9	IMDW0	**Internal Memory Data Width 0.** Selects the data access size for internal memory as 48-bit data if set, (= 1) or 32-bit data if cleared, (= 0). Permits core writes.
10	IMDW1	**Internal Memory Data Width 1.** Selects the data access size for internal memory as 48-bit data if set, (= 1) or 32-bit data if cleared, (= 0). Permits core writes.
15–11	Reserved	
16	IRQ0EN	**Flag0 Interrupt Mode.** 1 = Flag0 pin is allocated to interrupt request $\overline{\text{IRQ0}}$. 0 = Flag0 pin is a general purpose I/O pin. Permits core writes.
17	IRQ1EN	**Flag1 Interrupt Mode.** 1 = Flag1 pin is allocated to interrupt request $\overline{\text{IRQ1}}$. 0 = Flag1 pin is a general purpose I/O pin. Permits core writes.
18	IRQ2EN	**Flag2 Interrupt Mode.** 1 = Flag2 pin is allocated to interrupt request $\overline{\text{IRQ2}}$. 0 = Flag2 pin is a general purpose I/O pin. Permits core writes.

Table A-3. SYSCTL Register Bit Descriptions (Cont'd)

Bits	Name	Definition
19	TMREXPEN	**Flag Timer Expired Mode.** Read/Write 1 = Flag3 pin outputs are timer-expired signal (TIMEXP). 0 = Flag3 pin is a general purpose I/O pin. Permits core writes.
20	PPFLGS	**Parallel Port Select.** 0 = Parallel port is selected. 1 = Parallel port is not selected. ADDR and DATA pins are in FLAG mode. Permits core writes. Configuring the parallel port pins to function as FLAG0-15 also causes the FLAG[0:3] pins to change to their alternate role, $\overline{\text{IRQ0-2}}$ and TIMEXP.
31–21	Reserved	

Hardware Breakpoint Control Register (BRKCTL)

The BRKCTL register controls how breakpoints are used (if the UMODE bit is set). This user-accessible register in the BRKCTL register is located at address 0x30025.

The BRKCTL register is a 32-bit memory-mapped I/O register. The processor core can write into this register. The bits related to the breakpoint register are same as in the EMUCTL register.

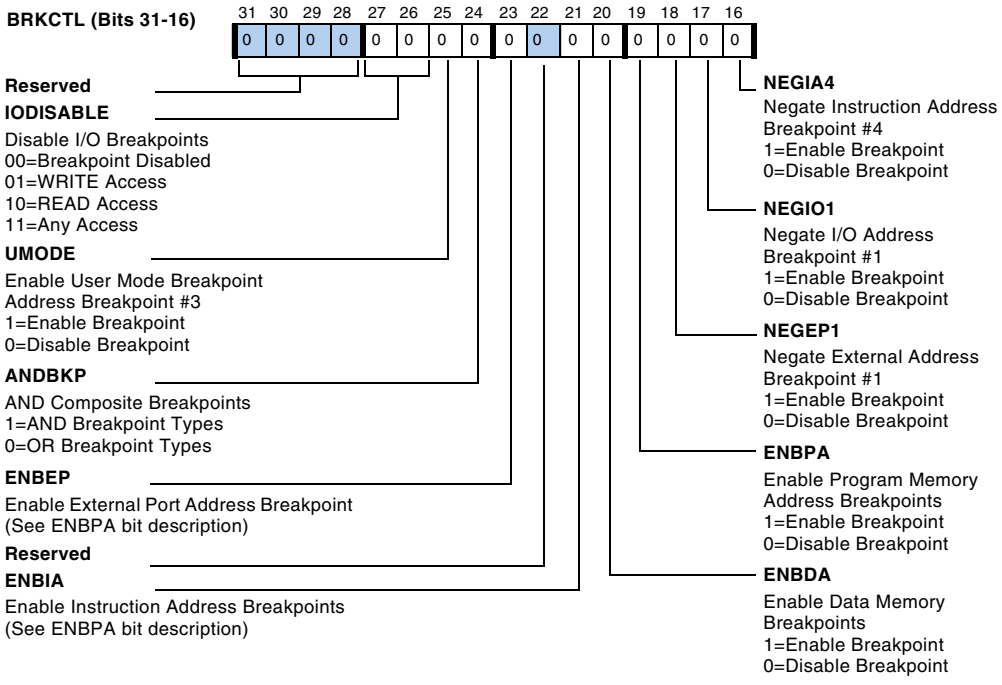


Figure A-4. BRKCTL Register (Upper Bits)

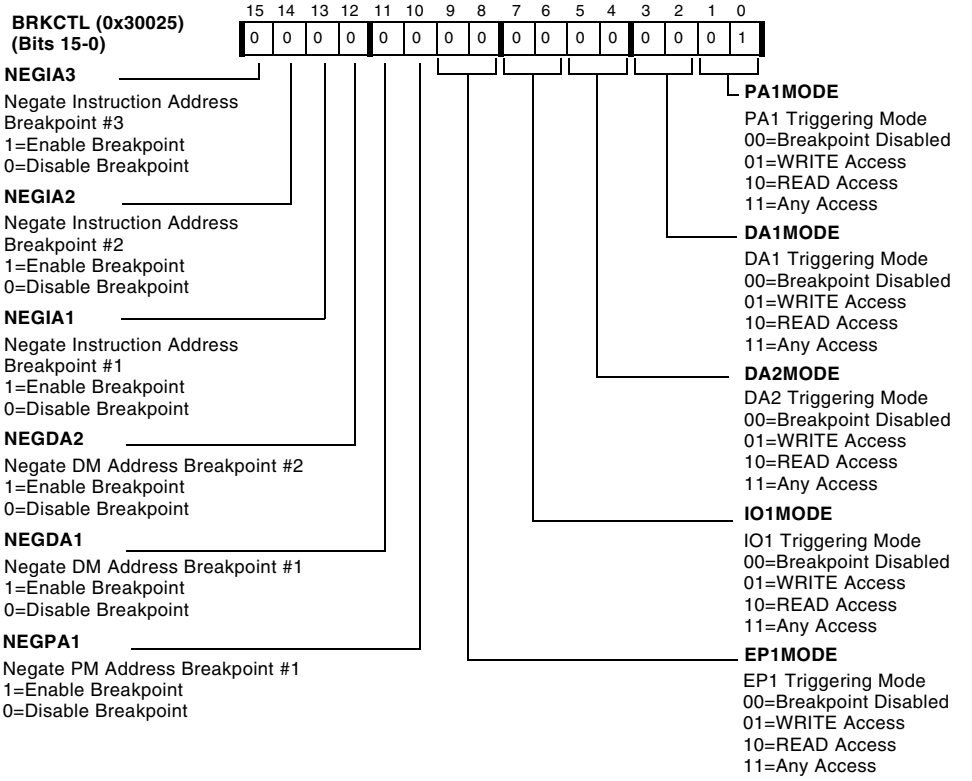


Figure A-5. BRKCTL Register (Lower Bits)

Table A-4. BRKCTL Register Bit Descriptions

Bit #	Name	Function
1–0	PA1MODE	**PA1Triggering Mode** 00 = Breakpoint Disabled 01 = WRITE Access 10 = READ Access 11 = Any access
3–2	DA1MODE	**DA1 Triggering Mode** 00 = Breakpoint Disabled 01 = WRITE Access 10 = READ Access 11 = Any access
5–4	DA2MODE	**DA2 Triggering Mode** 00 = Breakpoint Disabled 01 = WRITE Access 10 = READ Access 11 = Any Access
7–6	IO1MODE	**IO1 Triggering Mode** trigger on the following conditions: Mode Triggering condition 00 = Breakpoint is disabled 01 = WRITE accesses only 10 = READ accesses only 11 = Any access
9–8	EP1MODE	**EP1 Triggering Mode** 00 = Breakpoint Disabled 01 = WRITE Access 10 = READ Access 11 = Any Access
10	NEGPA1	**Negate Program Memory Data Address Breakpoint** Enable breakpoint events if the address is greater than the end register value OR less than the start register value. This function is useful to detect index range violations in user code. 0 = Disable Breakpoint 1 = Enable Breakpoint
11	NEGDA1	**Negate Data Memory Address Breakpoint #1** For more information, see NEGPA1 bit description.

Table A-4. BRKCTL Register Bit Descriptions (Cont'd)

Bit #	Name	Function
12	NEGDA2	**Negate Data Memory Address Breakpoint #2** For more information, see NEGPA1 bit description.
13	NEGIA1	**Negate Instruction Address Breakpoint #1** 0 = Disable Breakpoint 1 = Enable Breakpoint
14	NEGIA2	**Negate Instruction Address Breakpoint #2** For more information, see NEGPA1 bit description.
15	NEGIA3	**Negate Instruction Address Breakpoint #3** For more information, see NEGPA1 bit description.
16	NEGIA4	**Negate Instruction Address Breakpoint #4** For more information, see NEGPA1 bit description.
17	NEGIO1	**Negate I/O Address Breakpoint** For more information, see NEGPA1 bit description.
18	NEGEP1	**Negate EP Address Breakpoint** For more information, see NEGPA1 bit description.
19	ENBPA	**Enable Program Memory Data Address Breakpoints** The ENB* bits enable each breakpoint group. Note that when the ANDBKP bit is set, breakpoint types not involved in the generation of the effective breakpoint must be disabled. 0 = Disable Breakpoints 1 = Enable Breakpoints
20	ENBDA	**Enable Data Memory Address Breakpoints** For more information, see ENBPA bit description.
21	ENBIA	**Enable Instruction Address Breakpoints.** For more information, see ENBPA bit description.
22	Reserved	
23	ENBEP	**Enable External Port Address Breakpoint.** For more information, see ENBPA bit description.

Table A-4. BRKCTL Register Bit Descriptions (Cont'd)

Bit #	Name	Function
24	ANDBKP	**AND composite breakpoints** Enables ANDing of each breakpoint type to generate an effective breakpoint from the composite breakpoint signals. 0 = OR Breakpoint Types 1 = AND Breakpoint Types
25	UMODE	**User Mode Breakpoint Functionality Enable** Address Breakpoint 3 0 = Disable Breakpoint 1 = Enable Breakpoint
27–26	IODISABLE	**Enable I/O Breakpoints** 00 = Breakpoint Disabled 01 = WRITE Access 10 = READ Access 11 = Any Access
31–28	Reserved	

Serial Port Registers

The following section describes Serial Port (SPORT) registers.

SPORT Serial Control Registers (SPCTLx)

The SPORT Serial Control registers' addresses are:

SPCTL0 – 0xc00	SPCTL1 – 0xc01
SPCTL2 – 0x400	SPCTL3 – 0x401
SPCTL4 – 0x800	SPCTL5 – 0x801

The reset value for these registers is 0x0000 0000. The `SPCTLx` registers are Transmit and Receive Control registers for the corresponding serial ports (SPORT 0 through 5).

- Figure A-6 and Figure A-7 provide bit definitions for the `SPCTLx` register in Standard DSP Serial mode.
- Figure A-8 and Figure A-9 provides bit definitions in Left-justified Sample Pair and I^2S mode.
- Figure A-10 and Figure A-11 provides bit definitions for SPORTS 1, 3, and 5 (receive) in Multichannel mode.
- Figure A-12 and Figure A-13 provides bit definitions for SPORTS 0, 2, and 4 (transmit) in Multichannel mode.

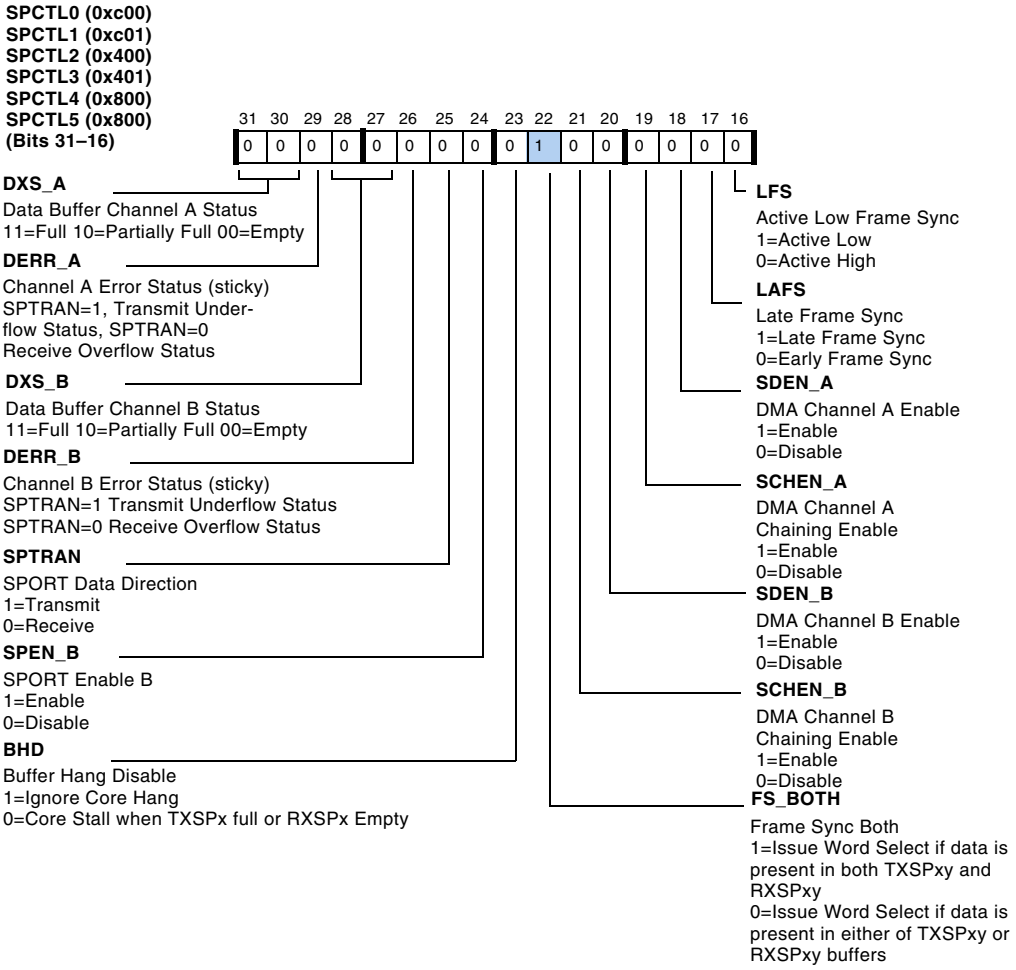


Figure A-6. SPCTLx Control Bits for Standard DSP Serial Mode (Upper)

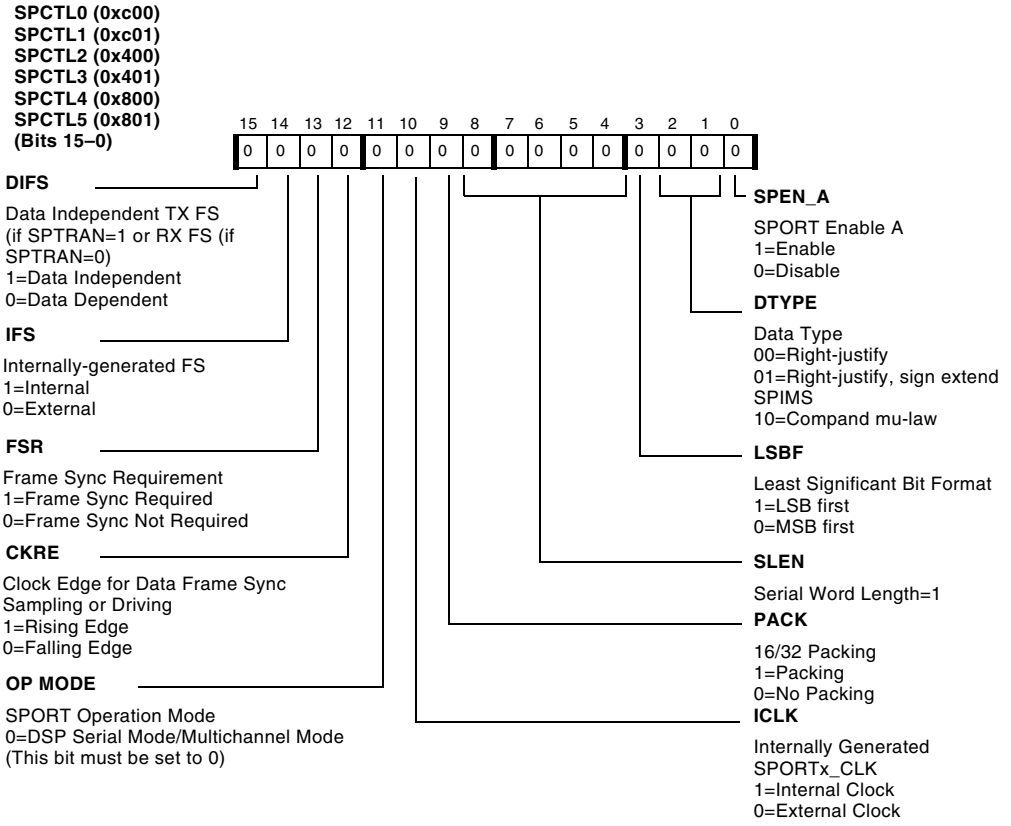


Figure A-7. SPCTLx Control Bits for Standard DSP Serial Mode (Lower)

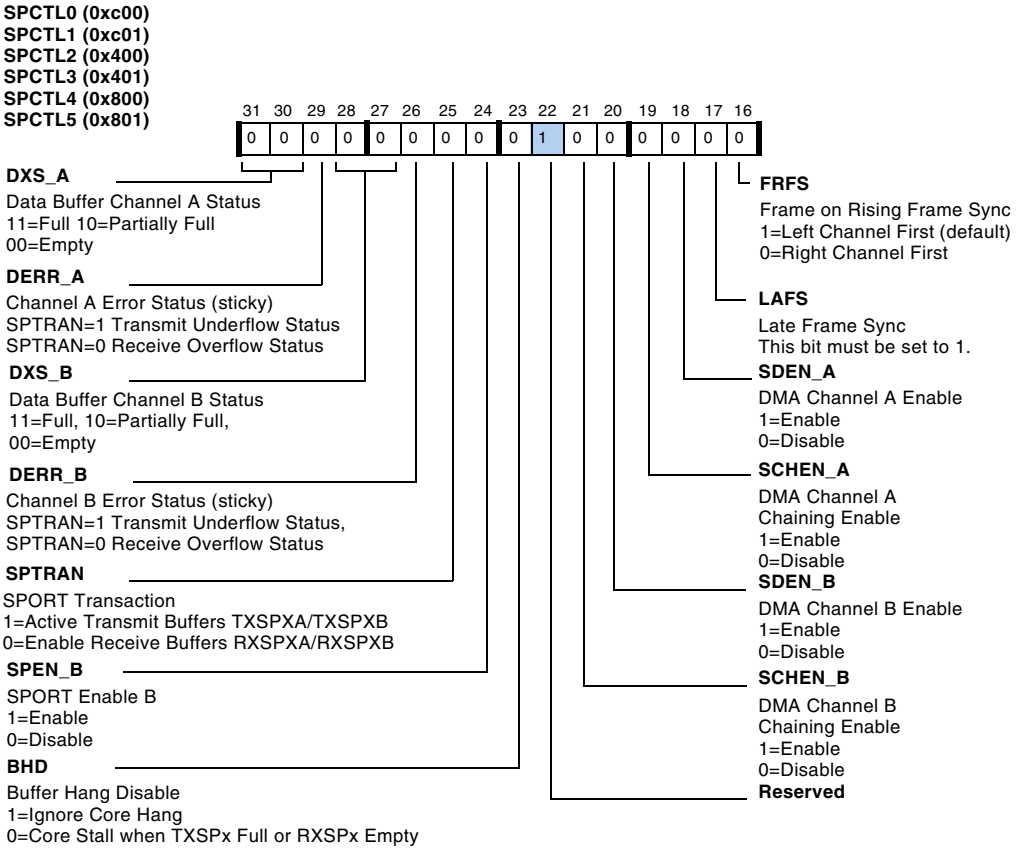


Figure A-8. SPCTLx Control Bits – for I^2S and Related Modes (Upper)

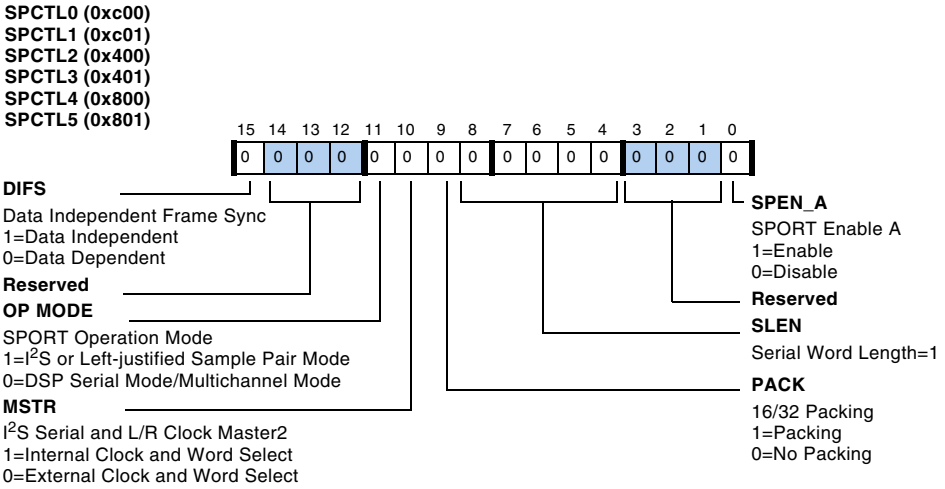


Figure A-9. SPCTLx Control Bits – for I^2S and Related Modes (Lower)

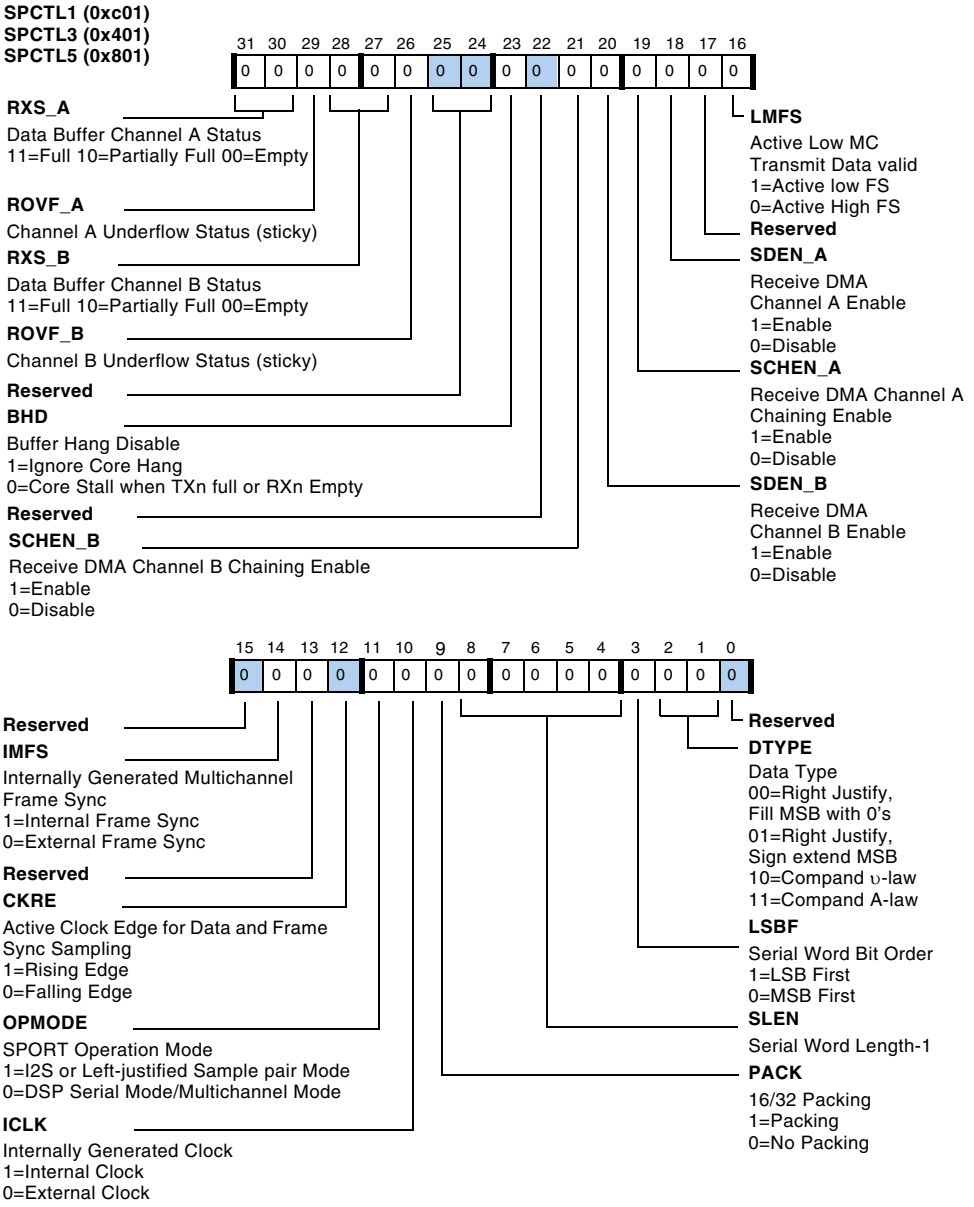


Figure A-10. SPCTLx Receive Control Bits – Multichannel Mode

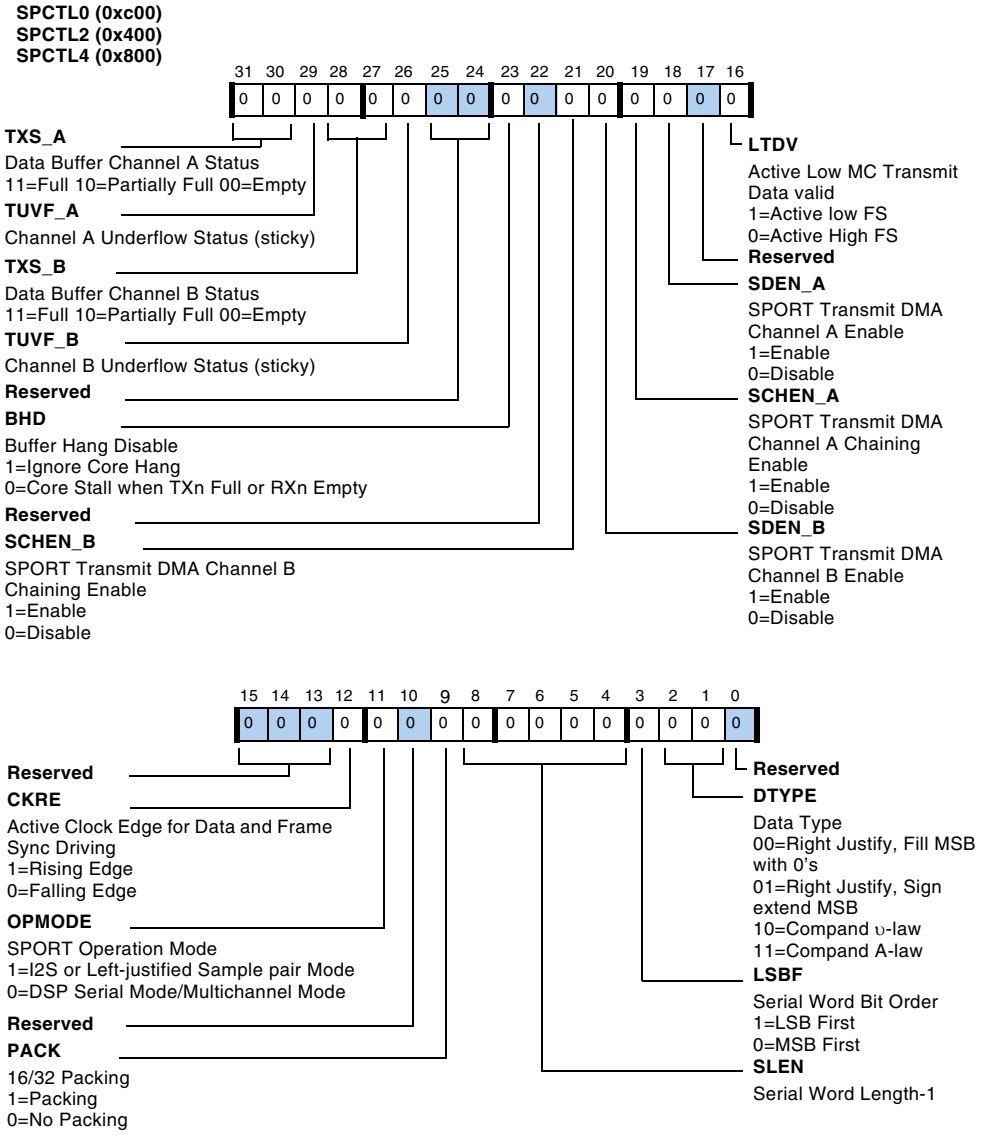


Figure A-11. SPCTLx Transmit Control Bits – Multichannel Mode

When changing SPORT operating modes, programs should clear a serial port's Control register before writing new settings to the Control register.

Table A-5. SPCTLx Register Bit Descriptions

Bits	Name	Definition
0	SPEN_A	**Enable Channel A Serial Port.** Enables if set, (= 1) or disables if cleared, (= 0) the corresponding serial port A channel. This bit is reserved when the SPORT is in Multichannel mode.
2–1	DTYPE	**Data Type Select.** Selects the data type formatting for normal and multichannel transmissions as follows: **Normal** **Multi** **Data Type Formatting** 00 x0 Right-justify, zero-fill unused MSBs 01 x1 Right-justify, sign-extend unused MSBs 10 0x Compand using μ-law 11 1x Compand using A-law
3	LSBF	**Serial Word Endian Select.** Selects little-endian words (LSB first, if set, = 1) or big-endian words (MSB first, if cleared, = 0). This bit is reserved when the SPORT is in I²S or Left-Justified Sample Pair mode.
8–4	SLEN	**Serial Word Length Select.** Selects the word length in bits. For DSP serial and multichannel modes, word sizes can be from 3 bits (SLEN = 2) to 32 bits (SLEN = 31). For I²S and Left-justified modes, word sizes can be from 8 bits (SLEN = 7) to 32 bits (SLEN = 31).
9	PACK	**16-bit to 32-bit Word Packing Enable.** Enables if set, (= 1) or disables if cleared, (= 0) 16- to 32-bit word packing.
10	ICLK MSTR (I^2S mode only)	**Internal Clock Select.** Selects the internal transmit clock if set, (= 1) or external transmit clock if cleared, (= 0). This bit applies to DSP Serial and multichannel modes. In I^2S and Left-justified Sample Pair mode, this bit selects the word source and internal clock if set, (= 1) or external clock if cleared, (= 0)
11	OPMODE	**Sport Operation Mode.** Selects the I^2S/Left-justified Sample Pair mode if set (= 1) or DSP Serial/Multichannel mode if cleared (= 0).
12	CKRE	**Clock Rising Edge Select.** Selects whether the serial port uses the rising edge if set, (= 1) or falling edge if cleared, (= 0) of the clock signal to sample data and the frame sync. CKRE is reserved when the SPORT is in I^2S and Left-justified Sample Pair mode.

Table A-5. SPCTLx Register Bit Descriptions (Cont'd)

Bits	Name	Definition
13	FSR	**Frame Sync Required Select.** Selects whether the serial port requires if set, (= 1) or does not require if cleared, (= 0) a transfer frame sync. FSR is reserved when the SPORT is in I^2S mode, Left-Justified Sample Pair mode and multichannel mode.
14	IFS (IMFS)	**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^2S, Left-justified Sample Pair mode and Multichannel mode.
15	DIFS	**Data Independent Frame Sync Select.** Selects whether the serial port uses a data-independent frame sync (sync at selected interval, if set, = 1) or uses a data-dependent frame sync (sync when TX FIFO is not empty or when RX FIFO is not full). This bit is reserved when the SPORT is in Multichannel mode.
16	LFS (LMFS, FRFS)	**Active Low Frame Sync Select.** Selects an active low FS if set, (= 1) or active high FS if cleared, (= 0).
17	LAFS	**Late Transmit Frame Sync Select.** Selects a late frame sync (FS during first bit, if set, = 1) or an early frame sync (FS before first bit, if cleared, = 0). This bit is reserved when the SPORT is in multichannel mode.
18	SDEN_A	**Enable Channel A Serial Port DMA.** Enables if set, (= 1) or disables if cleared, (= 0) the serial port's A channel DMA.
19	SCHEN_A	**Enable Channel A Serial Port DMA Chaining.** Enables if set, (= 1) or disables if cleared, (= 0) the serial port's channel A DMA chaining.
20	SDEN_B	**Enable Channel B Serial Port DMA.** Enables if set, (= 1) or disables if cleared, (= 0) the serial port's channel B DMA.
21	SCHEN_B	**Enable Channel B Serial Port DMA Chaining.** Enables if set, (= 1) or disables if cleared, (= 0) the serial port's channel B DMA chaining.
22	FS_BOTH	**FS Both Enable.** This bit issues WS if data is present in **both** transmit buffers, if set (= 1). If cleared (= 0), WS is issued if data is present in either transmit buffer. This bit is reserved when the SPORT is in multichannel, I^2S and Left-justified Sample Pair mode.

Table A-5. SPCTLx Register Bit Descriptions (Cont'd)

Bits	Name	Definition
23	BHD	**Buffer Hang Disable.** This bit ignores a core hang, when set (= 1). When cleared (= 0), this bit indicates a core stall. The core stall occurs when the transmit buffer is full or the receive buffer is empty and the core tries to write or read from the FIFO respectively. This bit applies to all modes
24	SPEN_B	**Enable Channel B Serial Port.** Enables if set, (= 1) or disables if cleared, (= 0) the corresponding serial port B channel. This bit is reserved when the SPORT is in Multichannel mode.
25	SPTRAN	**Data Direction Control.** Enables receive buffers if cleared (= 0), or activates transmit buffers if set (= 1). This bit is reserved when the SPORT is in Multichannel mode.
26	ROVF_B, TUVF_B	**Channel B Error Status (sticky, read-only).** Indicates if the serial transmit operation has underflowed or a receive operation has overflowed in the channel B data buffer. This bit is reserved when the SPORT is in Multichannel mode.
28–27	DXS_B	**Channel B Data Buffer Status (read-only).** Indicates the status of the serial port's channel B data buffer as follows: 11 = full, 00 = empty, 10 = partially full. This bit is reserved when the SPORT is in Multichannel mode.
29	ROVF_A or TUVF_A	**Channel A Error Status (sticky, read-only).** Indicates if the serial transmit operation has underflowed or a receive operation has overflowed in the channel A data buffer.
31–30	RXS_A or TXS_A	**Channel A Data Buffer Status (read-only).** Indicates the status of the serial port's channel A data buffer as follows: 11 = full, 00 = empty, 10 = partially full.

SPORT Multichannel Control Registers (SPMCTLxy)

The SPORT Multichannel Control registers' addresses are:

SPMCTL01	0xc04
SPMCTL23	0x404
SPMCTL45	0x804

The SPMCTL01 register is the Multichannel Control register for SPORTs 0 and 1. The SPMCTL23 register is the Multichannel Control register for SPORTs 2 and 3. The SPMCTL45 register is the Multichannel Control register for SPORTs 4 and 5. The reset value for these registers is undefined.

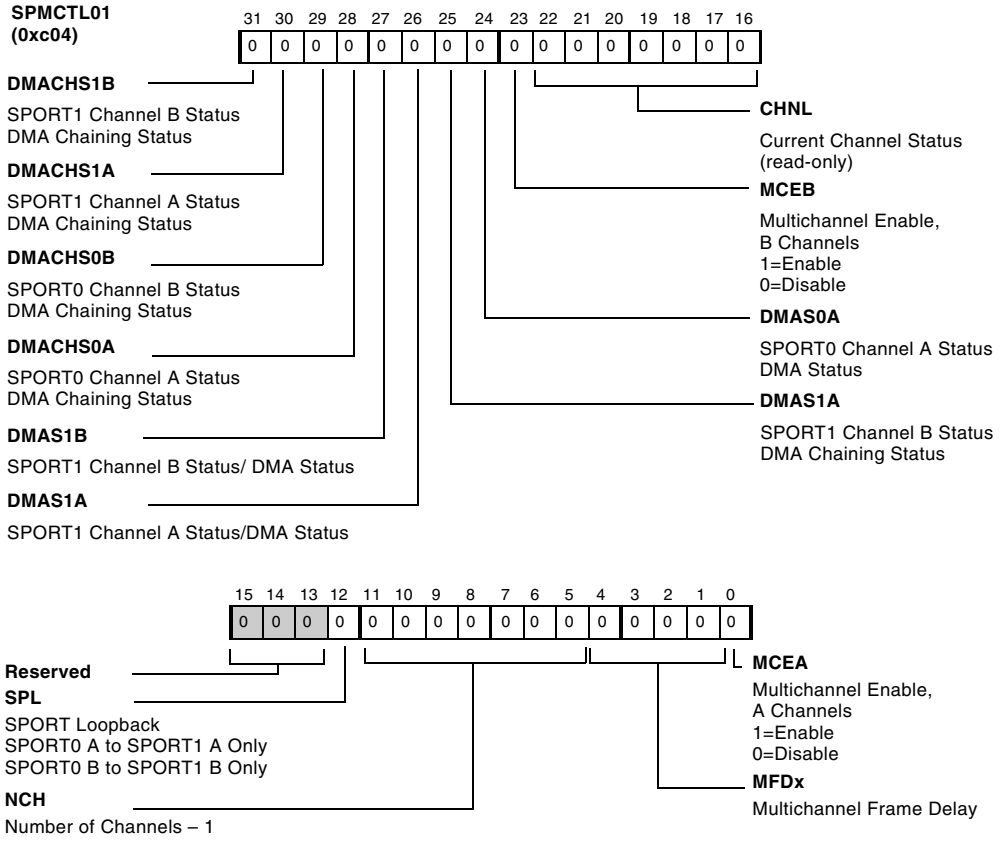


Figure A-12. SPMCTL01 Register – Multichannel Mode

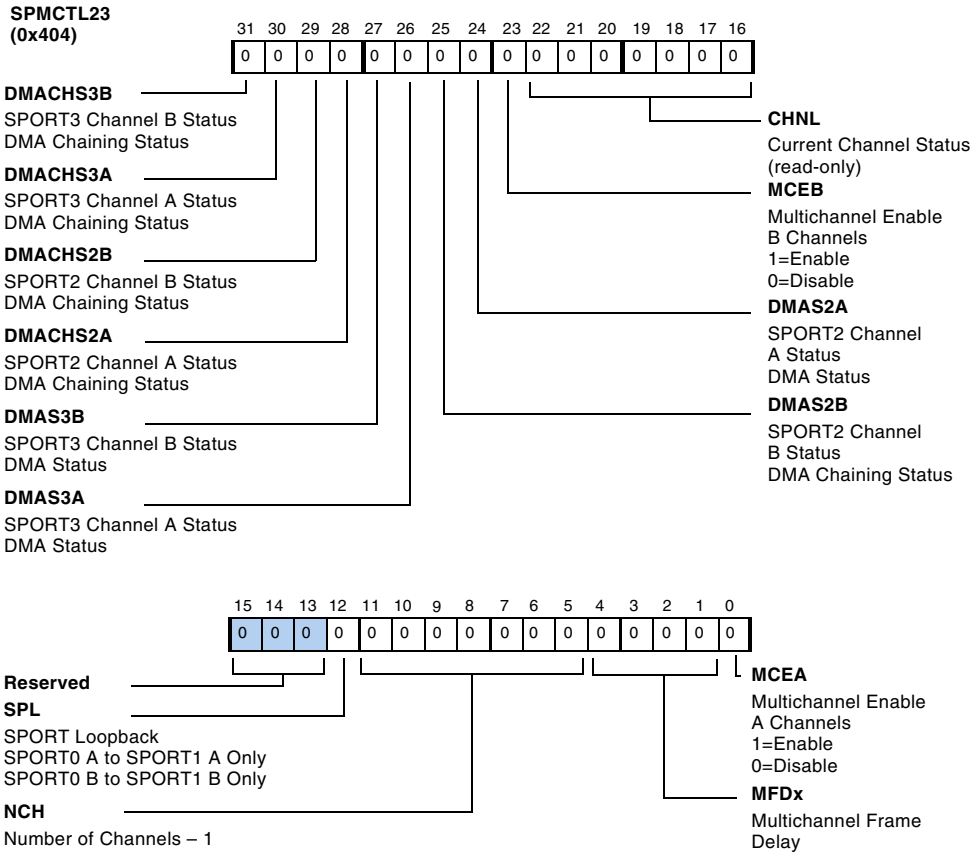


Figure A-13. SPMCTL23 Registers – Multichannel Mode

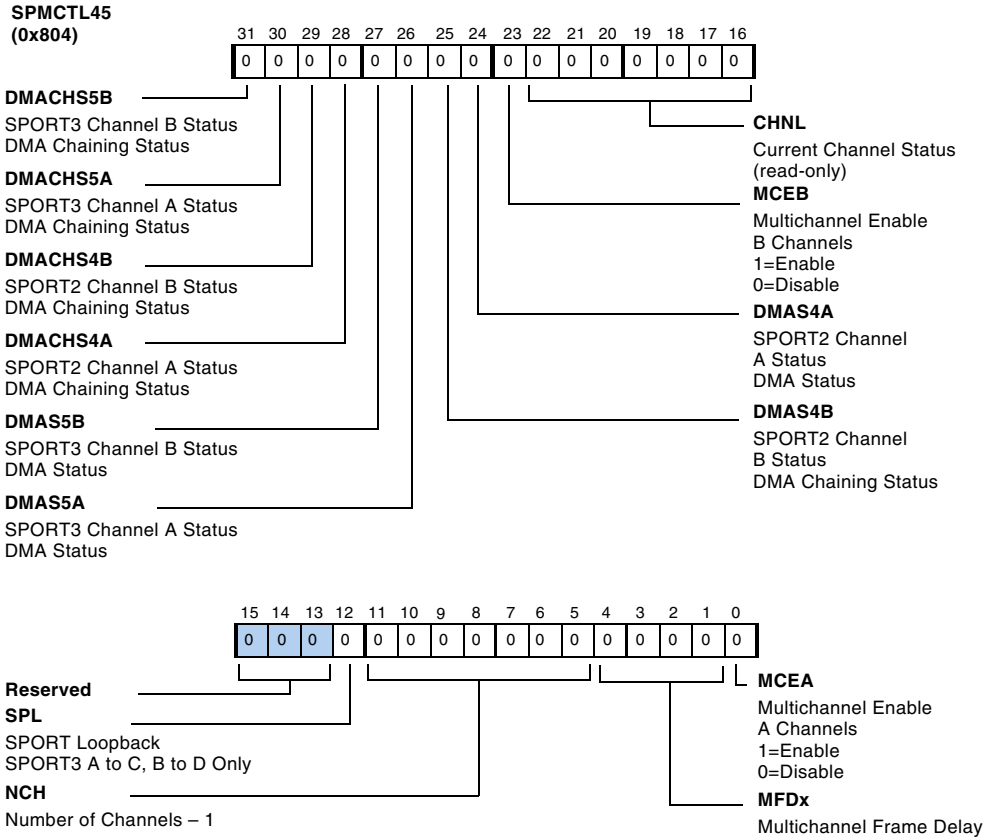


Figure A-14. SPMCTL45 Registers – Multichannel Mode

For a detailed description of the bits in the `SPMCTLxy` register, refer to Table A-6.

Table A-6. SPMCTLxy Register Bit Descriptions

Bits	Name	Definition
0	MCEA	**Multichannel Mode Enable.** Standard and Multichannel modes only. One of two configuration bits that enable and disable multichannel mode on serial port channels. See also, OPMODE on page A-26. 0 = Disable multichannel operation 1 = Enable multichannel operation if OPMODE = 0
4–1	MFD	**Multichannel Frame Delay.** Set 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 SPMCTL01 [4:1] or SPMCTL23[4:1] or SPMCTL45[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.
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.

Table A-6. SPMCTLxy Register Bit Descriptions (Cont'd)

Bits	Name	Definition
12	SPL	**SPORT Loopback Mode** SPORT Loopback Mode. Enables if set (= 1) or disables if cleared (= 0) the channel loopback mode. Loopback mode enables developers to run internal tests and to debug applications. Loopback only works under the following SPORT configurations: • SPORT0 (configured as a receiver or transmitter) together with SPORT1 (configured as a transmitter or receiver). SPORT0 can only be paired with SPORT1, controlled via the SPL bit in the SPMCTL01 register. • SPORT2 (as a receiver or transmitter) together with SPORT3 (as a transmitter or receiver). SPORT2 can only be paired with SPORT3, controlled via the SPL bit in the SPMCTL23 register. • SPORT4 (configured as a receiver or transmitter) together with SPORT5 (configured as a transmitter or receiver). SPORT4 can only be paired with SPORT5, controlled via the SPL bit in the SPMCTL45 register. Either of the two paired SPORTs can be set up to transmit or receive, depending on their SPTRAN bit configurations.
15–13	Reserved	
22–16	CHNL	**Current Channel Selected (Read-only, Sticky).** Identify the currently selected transmit channel slot (0 to 127).
23	MCEB	**Multichannel Enable, B Channels.** 0 = Disable 1 = Enable
27–24	DMASxy	**DMA Status.** Selects the transfer format. 0 = Inactive 1 = Active (Read-only)
31–28	DMACHSxy	**DMA Chaining Status.** 0 = Inactive 1 = Active) (Read-only)

SPORT Transmit Buffer Registers (TXSPx)

The addresses of the TXSPx registers are:

TXSP0A – 0xc60	TXSP0B – 0xc62
TXSP1A – 0xc64	TXSP1B – 0xc66
TXSP2A – 0x460	TXSP2B – 0x462
TXSP3A – 0x464	TXSP3B – 0x466
TXSP4A – 0x860	TXSP4B – 0x862
TXSP5A – 0x864	TXSP5B – 0x866

The reset value for these registers is undefined. The 32-bit TXSPx registers hold the output data for serial port transmit operations. For more information on how transmit buffers work, see "Transmit and Receive Data Buffers" on page 4-59.

SPORT Receive Buffer Registers (RXSPx)

The addresses of the RXSPx registers are:

RXSP0A – 0xc61	RXSP0B – 0xc63
RXSP1A – 0xc65	RXSP1B – 0xc67
RXSP2A – 0x461	RXSP2B – 0x463
RXSP3A – 0x465	RXSP3B – 0x467
RXSP4A – 0x861	RXSP4B – 0x863
RXSP5A – 0x865	RXSP5B – 0x867

The reset value for these registers is undefined. The 32-bit RXSPx registers hold the input data from serial port receive operations. For more information on how receive buffers work, see "Transmit and Receive Data Buffers" on page 4-59.

SPORT Divisor Registers (DIVx)

The addresses of the DIVx registers are:

DIV0 – 0xc02	DIV1 – 0xc03
DIV2 – 0x402	DIV3 – 0x403
DIV4 – 0x802	DIV5 – 0x803

The reset value for these registers is undefined. These registers contain two fields:

- Bits 15–1 are CLKDIV. These bits identify the Serial Clock Divisor value for internally-generated SCLK as follows:

$$CLKDIV = \frac{f_{CCLK}}{4(f_{SCLK})} - 1$$

- Bits 31–16 are FSDIV. These bits select the Frame Sync Divisor for internally-generated TFS as follows:

$$FSDIV = \frac{f_{SCLK}}{f_{SFS}} - 1$$

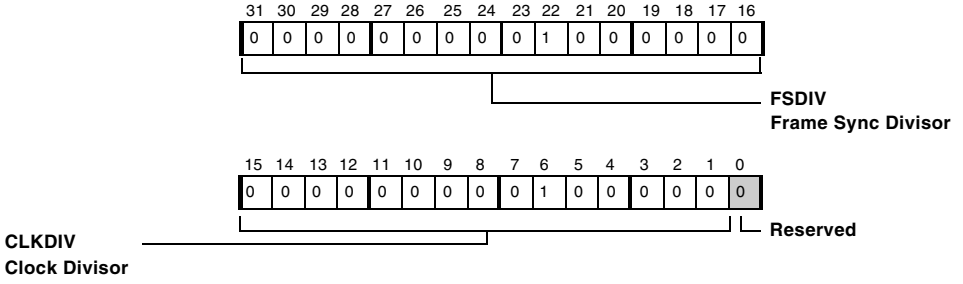


Figure A-15. DIVx Register

SPORT Count Registers (SPCNTx)

The addresses of the SPCNTx registers are:

SPCNT0 – 0xC15	SPCNT1 – 0xC16
SPCNT2 – 0x415	SPCNT3 – 0x416
SPCNT4 – 0x815	SPCNT5 – 0x816

The reset value for these registers is undefined. The SPCNTx registers provides status information for the internal clock and frame sync.

SPORT Transmit Select Registers (MTxCSy)

The addresses of the MTxCSy registers are:

MT0CS0 – 0xC05	MT0CS1 – 0xC06
MT0CS2 – 0xC07	MT0CS3 – 0xC08
MT2CS0 – 0x405	MT2CS1 – 0x406
MT2CS2 – 0x407	MT2CS3 – 0x408
MT4CS0 – 0x805	MT4CS1 – 0x806
MT4CS2 – 0x807	MT4CS3 – 0x808

The reset value for these registers is undefined.

Each bit, 31–0, set (= 1) in one of four MTxCSy registers corresponds to an active transmit channel, 127–0, on a Multichannel mode serial port. When the MTxCSy registers activate a channel, the serial port transmits the word in that channel's position of the data stream. When a channel's bit in the MTxCSy register is cleared (= 0), the serial port's data transmit pin three-states during the channel's transmit time slot.

SPORT Transmit Compand Registers (MTxCCSy)

The addresses of the MTxCCSy registers are:

MT0CCS0 – 0xC0D	MT0CCS1 – 0xC0E
MT0CCS2 – 0xC0F	MT0CCS3 – 0xc10
MT2CCS0 – 0x40D	MT2CCS1 – 0x40E
MT2CCS2 – 0x40F	MT2CCS3 – 0x410
MT4CCS0 – 0x80D	MT4CCS1 – 0x80E
MT4CCS2 – 0x80F	MT4CCS3 – 0x810

The reset value for these registers is undefined.

Each bit, 31–0, set (= 1) in one of four MTxCCSx registers corresponds to an companded transmit channel, 127–0, on a Multichannel mode serial port. When the MTCCSx register activates companding for a channel, the serial port applies the companding from the serial port's DTYPE selection to the transmitted word in that channel's position of the data stream. When a channel's bit in the MTCCSx register is cleared (= 0), the serial port does not compand the output during the channel's receive time slot.

SPORT Receive Select Registers (MRxCSx)

The addresses of the MRxCSx registers are:

MR1CS0 – 0xC09	MR1CS1 – 0xC0A
MR1CS2 – 0xC0B	MR1CS3 – 0xC0C
MR3CS0 – 0x409	MR3CS1 – 0x40A
MR3CS2 – 0x40B	MR3CS3 – 0x40C
MR5CS0 – 0x809	MR5CS1 – 0x80A
MR5CS2 – 0x80B	MR5CS3 – 0x80C

The reset value for these registers is undefined.

Each bit, 31–0, set (= 1) in one of the four MRCSx registers corresponds to an active receive channel, 127–0, on a Multichannel mode serial port. When the MRxCSx register activates a channel, the SPORT receives the word in that channel's position of the data stream and loads the word into the RXSPx buffer. When a channel's bit in the MRxCSx register is cleared (= 0), the serial port ignores any input during the channel's receive time slot.

SPORT Receive Compand Registers (MRxCCSx)

These addresses for the MRxCCSx registers are:

MR1CCS0 – 0xC11	MR1CCS1 – 0xC12
MR1CCS2 – 0xC13	MR1CCS3 – 0xC14
MR3CCS0 – 0x411	MR3CCS1 – 0x412
MR3CCS2 – 0x413	MR3CCS3 – 0x414
MR5CCS0 – 0x811	MR5CCS1 – 0x812
MR5CCS2 – 0x813	MR5CCS3 – 0x814

The reset value for these registers is undefined.

Each bit, 31–0, set (= 1) in the MRxCCSy registers corresponds to an companded receive channel, 127–0, on a Multichannel mode serial port. When one of the four MRxCCSy registers activate companding for a channel, the serial port applies the companding from the serial port's DTYPE selection to the received word in that channel's position of the data stream. When a channel's bit in the MRxCCSy registers are cleared (= 0), the serial port does not compand the input during the channel's receive time slot.

SPORT DMA Index Registers (IISPx)

The addresses of the IISPx registers are:

IISP0A – 0xC40	IISP0B – 0xC44
IISP1A – 0xC48	IISP1B – 0xC4C
IISP2A – 0x440	IISP2B – 0x444
IISP3A – 0x448	IISP3B – 0x44C
IISP4A – 0x840	IISP4B – 0x844
IISP5A – 0x848	IISP5B – 0x84C

The reset value for these registers is undefined. The IISPx register is 19 bits wide and it holds an address and acts as a pointer to memory for a DMA transfer. For more information, see Chapter 2, I/O Processor.

SPORT DMA Modifier Registers (IMSPx)

The addresses of the IMSPx registers are:

IMSP0A – 0xC41	IMSP0B – 0xC45
IMSP1A – 0xC49	IMSP1B – 0xC4D
IMSP2A – 0x441	IMSP2B – 0x445
IMSP3A – 0x449	IMSP3B – 0x44D
IMSP4A – 0x841	IMSP4B – 0x845
IMSP5A – 0x849	IMSP5B – 0x84D

The reset value for these registers is undefined. The IMSPx register is 16 bits wide and it provides the increment or step size by which an IISPx register is post-modified during a DMA operation. For more information, see Chapter 2, I/O Processor.

SPORT DMA Count Registers (CSPx)

The CSPx registers' addresses are:

CSP0A – 0xC42	CSP0B – 0xC46
CSP1A – 0xC4A	CSP1B – 0xC4E
CSP2A – 0x442	CSP2B – 0x446
CSP3A – 0x44A	CSP3B – 0x44E
CSP4A – 0x842	CSP4B – 0x846
CSP5A – 0x84A	CSP5B – 0x84E

The reset value for these registers is undefined. The CSPx registers are 16 bits wide and they hold the word count for a DMA transfer. For more information, see Chapter 2, I/O Processor.

SPORT Chain Pointer Registers (CPSP)

The addresses of the CPSP registers are:

CPSP0A – 0xC43	CPSP0B – 0xC47
CPSP1A – 0xC4B	CPSP1B – 0xC4F
CPSP2A – 0x443	CPSP2B – 0x447
CPSP3A – 0x44B	CPSP3B – 0x44F
CPSP4A – 0x843	CPSP4B – 0x847
CPSP5A – 0x84B	CPSP5B – 0x84F

The reset value for these registers is undefined. The CPSPx registers are 20 bits wide and they hold the address for the next Transfer Control Block in a chained DMA operation. For more information, see Chapter 2, I/O Processor.

SPI Registers

The following sections describe the registers associated with the Serial Peripheral Interface (SPI).

SPI Port Status Register (SPISTAT)

The SPI Port Status register's address is 0x1002. The reset value for this register is 0x01.The SPISTAT register is a read-only register used to detect when an SPI transfer is complete, if transmission/reception errors occur, and the status of the TXSPI and RXSPI FIFOs.

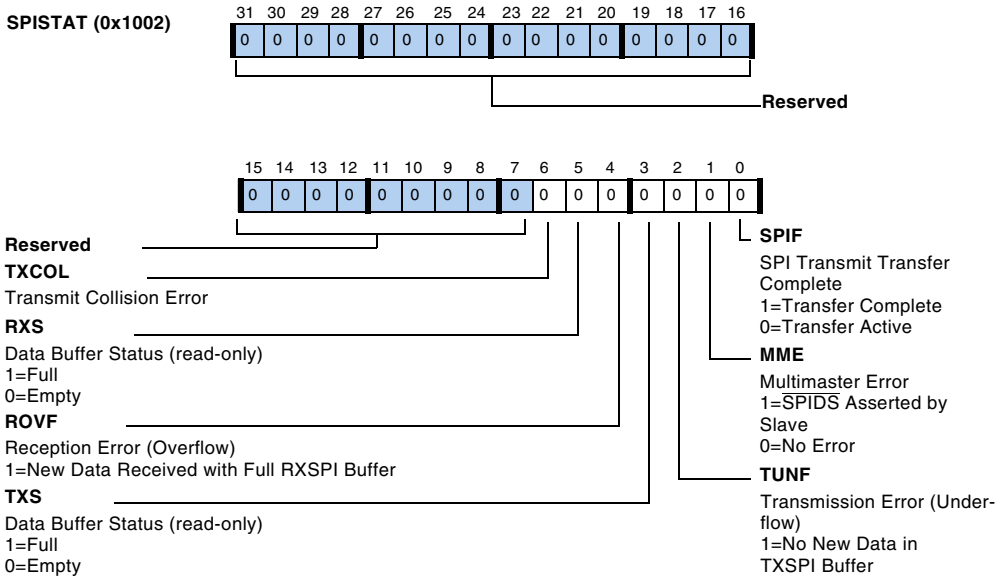


Figure A-16. SPISTAT Register

Table A-7. SPISTAT Register Bit Descriptions

Bits	Name	Definition
0	SPIF	**SPI Transmit or Receive Transfer Complete.** Set when an SPI single word transfer is complete.
1	MME	**Multimaster Error or Mode-fault error.** Set in a master device when some other device tries to become the master.
2	TUNF	**Transmission Error.** Set when transmission occurred with no new data in the TXSPI register.
3	TXS	**Transmit Data Buffer Status.** 0 = Empty 1 = Full
4	ROVF	**Reception Error.** Set when data is received with the receive buffer full.
5	RXS	**Receive Data Buffer Status.** 0 = Empty 1 = Full
6	TXCOL	**Transmit Collision Error.** When set, it is possible that corrupt data was transmitted.
31–7	Reserved	

SPI Port Flags Register (SPIFLG)

This register's address is 0x1001. The reset value for this register is 0x0F80.The SPIFLG register is used to enable individual SPI slave select lines when the SPI is enabled as a master.

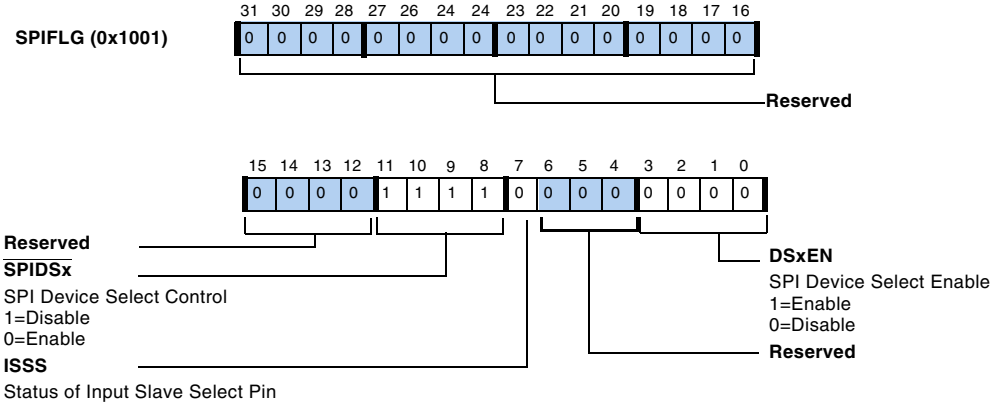


Figure A-17. SPIFLG Register

Table A-8. SPIFLG Register Bits

Bit	Name	Function
3–0	DSxEN (3–0)	**SPI Device Select Bits.** This bit enables or disables if set, (= 1 or if cleared = 0) the corresponding flag output to be used for an SPI slave-select.
6–4	Reserved	
7	ISSS	**Input Service Select Bit.** This read-only bit reflects the status of the slave select input pin.
11–8	SPIFLGx (3–0)	**SPI Device Select Control.** This bit if cleared, (= 0) selects a corresponding flag output to be used for SPI slave-select.
31–12	Reserved	

SPI Control Register (SPICTL)

This register's address is 0x1000. The reset value for this register is 0x0400. The SPI Control register (SPICTL) is used to configure and enable the SPI system. This register is used to set up SPI configurations such as selecting the device as a master or slave or determining the data transfer rate and word size.

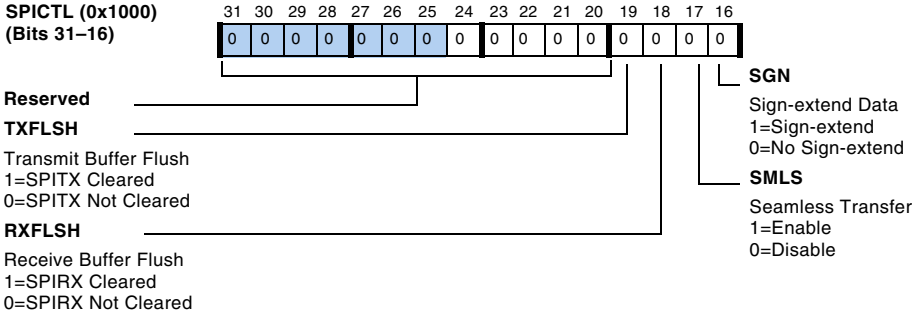


Figure A-18. SPICTL Register (Upper bits)

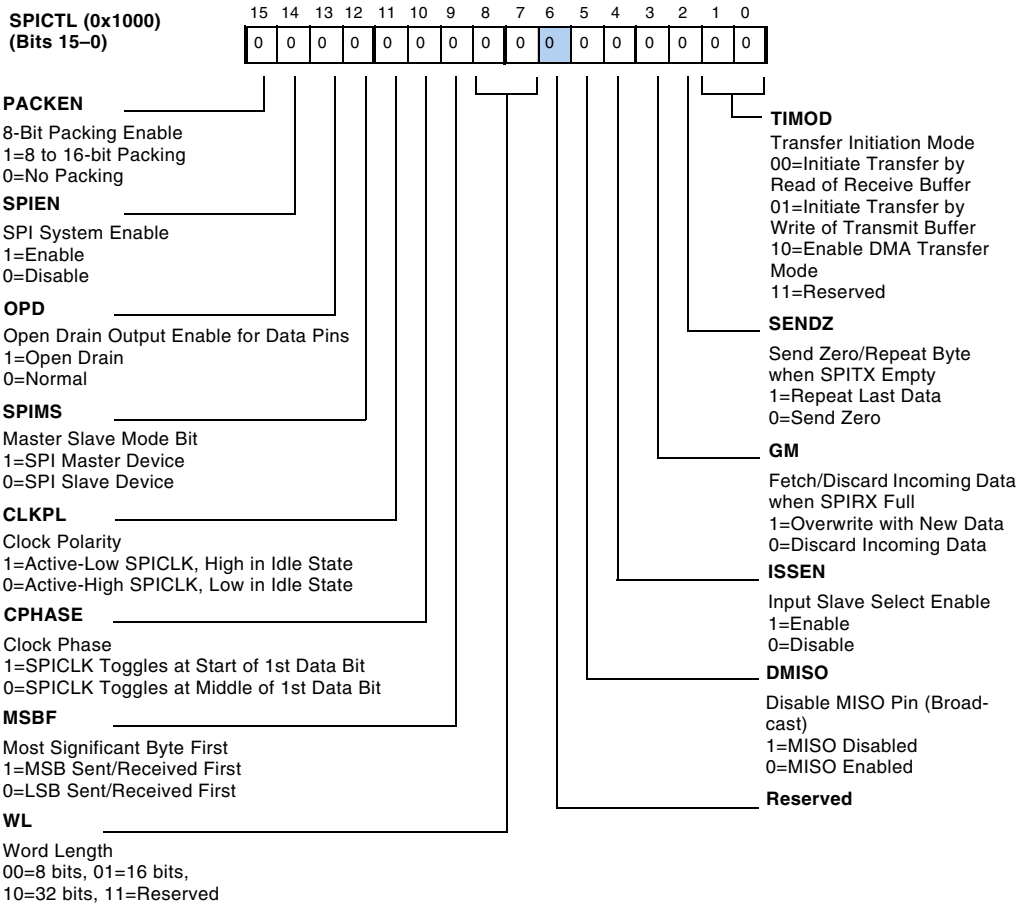


Figure A-19. SPICTL Register (Lower bits)

SPI Receive Buffer Register (RXSPI)

The SPI Receive Buffer register's address is 0x1004. The reset value for this register is undefined. This is a 32-bit read-only register accessible by the core or DMA controller. At the end of a data transfer, the `RXSPI` register is loaded with the data in the Shift register. During a DMA receive

operation, the data in RXSPI is automatically loaded into the internal memory. For core- or interrupt-driven transfers, programs can also use the RXS status bits in the SPISTAT register to determine if the receive buffer is full.

Table A-9. SPICTL Register Bit Descriptions

Bits	Name	Definition
1–0	TIMOD	**Transfer Initiation Mode.** Defines the 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 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 ($\overline{\text{SPIDS}}$) input for the master. When not used, $\overline{\text{SPIDS}}$ can be disabled, freeing up a chip pin as a general-purpose I/O pin. 0 = Disable 1 = Enable
5	DMISO	**Disable MISO Pin.** Disables MISO as an output in an environment where the master wishes to transmit to various slaves at one time (broadcast). Only one slave is allowed to transmit data back to the master. Except for the slave from whom the master wishes to receive, all other slaves should have this bit set. 0 = MISO Enabled 1 = MISO Disabled
6	Reserved	
8–7	WL	**Word Length.** 00 = 8 bits, 01 = 16 bits, 10 = 32 bits

Table A-9. SPICTL Register Bit Descriptions (Cont'd)

Bits	Name	Definition
9	MSBF	**Most Significant Byte First.** 1 = MSB sent/received first 0 = LSB 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
11	CLKPL	**Clock Polarity.** 0 = Active-high SPICLK (SPICLK low is the idle state) 1 = Active-low SPICLK (SPICLK high is the idle state)
12	SPIMS	**Master Select.** Configures SPI module as master or slave 0 = Device is a slave device 1 = Device is a master device
13	OPD	**Open Drain Output Enable.** Enables open drain data output enable (for MOSI and MISO) 0 = Normal 1 = Open Drain
14	SPIEN	**SPI Port Enable.** 0 = SPI Module is disabled 1 = SPI Module is enabled
15	PACKEN	**Packing Enable.** 0 = No Packing 1 = 8-16 Packing Note: This bit may be 1 only when WL = 00 (8-bit transfer). When in transmit mode, PACKEN bit will unpack data.
16	SGN	**Sign Extend Bit.** 0 = No sign extension 1 = Sign Extension
17	SMLS	**Seamless Transfer Bit.** 0 = Seamless transfer disabled 1 = Seamless transfer enabled not supported in mode TIMOD[1:0] = 00 and CPHASE = 0 for all modes.
18	TXFLSH	**Flush Transmit Buffer.** Write a 1 to this bit to clear TXSPI 0 = TXSPI not Cleared 1 = TXSPI Cleared

Table A-9. SPICTL Register Bit Descriptions (Cont'd)

Bits	Name	Definition
19	RXFLSH	**Clear RXSPI.** Write a 1 to this bit to clear RXSPI 0 = RXSPI not Cleared 1 = RXSPI Cleared
31–20	Reserved	

RXSPI Shadow Register (RXSPI_SHADOW)

The RXSPI Shadow register's address is 0x1006. The reset value for this register is undefined. This register acts as a shadow register for the Receive Data Buffer (RXSPI) register, and is used to aid software debugging. The RXSPI_SHADOW register is at a different address from the RXSPI register, but its contents are identical. When a software read of the RXSPI register occurs, the RXS bit is cleared and an SPI transfer may be initiated (if TIMOD = 00). No such hardware action occurs when the shadow register is read.

SPI Transmit Buffer Register (TXSPI)

The SPI Transmit Buffer register's address is 0x1003.The reset value for this register is undefined. This SPI Transmit Data register is a 32-bit register which is part of the IOP register set and can be accessed by the core or the DMA controller. Data is loaded into this register before being transmitted. Prior to the beginning of a data transfer, data in the TXSPI register is automatically loaded into the Transmit Shift register. During a DMA transmit operation, the data in the TXSPI register is automatically loaded from internal memory.

SPI Baud Rate Register (SPIBAUD)

The SPI Baud Rate register's address is 0x1005 and its reset value is undefined. This SPI register is a 16-bit read-write register that is used to set the bit transfer rate for a master device. When configured as a slave, the value written to this register is ignored. The SPIBAUD register can be read or written at any time. The serial clock rate is determined by the following formula:

```
SPI Baud Rate=(Core clock rate)/(4*SPIBAUD[15:1])
```

Writing a value of zero or one to the register disables the serial clock. Therefore, the maximum serial clock rate is one-fourth the core clock rate (CCLK).

Table A-10. SPIBAUD Register Bit Descriptions

Bits	Name	Definition
0	Reserved	
15–1	BAUDR	Enables the SPICLK per the following equation: SPICLK baud rate = core clock (CCLK)/4 x BAUDR) Default = 0.
31–16	Reserved	

Table A-11. SPI Master Baud Rate Example

BAUDR (Decimal Value)	SPI Clock Divide Factor	Baud Rate for CCLK @ 200 MHz
0	N/A	N/A
1	4	50.0 MHz
2	8	25.0 MHz
3	12	16.67 MHz
4	16	12.5 MHz
32,767, (0x7FFF)	131,068	1.53 KHz

SPI DMA Registers

There are five SPI DMA-specific registers:

SPI DMA Configuration Register (SPIDMAC)

The SPI DMA Configuration register's address is 0x1084 and its reset value is undefined. This SPI register is a 17-bit register used to control DMA transfers.

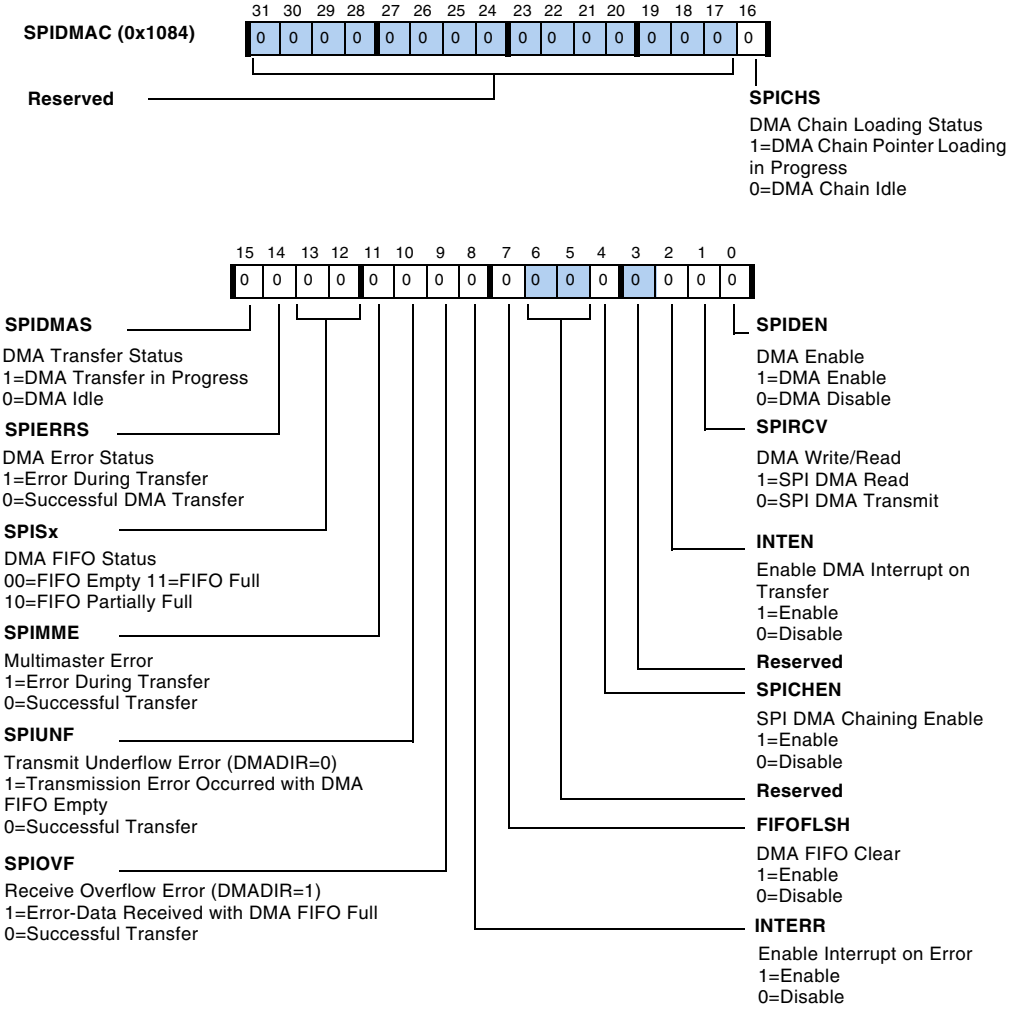


Figure A-20. SPIDMAC Register

Table A-12. SPIDMAC Register Bit Descriptions

Bits	Name	Definition
0	SPIDEN	**DMA Enable.** 1 = Enable 0 = Disable
1	SPIRCV	**DMA Write/Read.** 0 = SPI DMA Transmit 1 = SPI DMA Read
2	INTEN	**Enable DMA Interrupt on Transfer.** 1 = Enable 0 = Disable
3	Reserved	
4	SPICHEN	**SPI DMA Chaining Enable.** 0 = Disable 1 = Enable
6–5	Reserved	
7	FIFOFLSH	**DMA FIFO Clear.** 0 = Disable 1 = Enable
8	INTERR	**Enable Interrupt on Error.** 1 = Enable 0 = Disable
9	SPIOVF	**Receive OverFlow Error (SPIRCV = 1).** 0 = Successful Transfer 1 = Error - Data Received with DMA FIFO full
10	SPIUNF	**Transmit Underflow Error (SPIRCV = 0).** 0 = Successful Transfer 1 = Error Occurred in Transmission with DMA FIFO Empty
11	SPIMME	**Multimaster Error.** 0 = Successful Transfer 1 = Error During Transfer

Table A-12. SPIDMAC Register Bit Descriptions (Cont'd)

Bits	Name	Definition
13–12	SPISx	**DMA FIFO Status.** 00 = FIFO empty 11 = FIFO full 10 = FIFO partially full 01 = Reserved
14	SPIERRS	**DMA Error Status.** 0 = Successful DMA Transfer 1 = Errors during DMA Transfer
15	SPIDMAS	**DMA Transfer Status.** 0 = DMA Idle 1 = DMA in Progress
16	SPICHS	**DMA Chain Loading Status.** 0 = Chain Idle 1 = Chain Loading in Progress
31–17	Reserved	

SPI DMA Start Address Register (IISPI)

The SPI DMA Start Address register's address is 0x1080. The reset value for this register is undefined. This SPI register is a 19-bit read/write register that contains the start address of the buffer in memory.

SPI DMA Address Modify Register (IMSPI)

The SPI DMA Address Modify register's address is 0x1081. The reset value for this register is undefined. This SPI register is a 16-bit read/write register that contains the address modifier.

SPI DMA Word Count Register (CSPI)

The SPI DMA Word Count register's address is 0x1082.The reset value for this register is undefined. This SPI register is a 16-bit read/write register that contains the number of words to be transferred.

SPI DMA Chain Pointer Register (CPSPI)

The SPI DMA Chain Pointer register's address is 0x1083.The reset value for this register is undefined. This SPI register is a 20-bit read/write register that contains the address of the next TCB when DMA chaining is enabled.

Parallel Port Registers

The Parallel Port peripheral in the ADSP-2126x processor includes several user-accessible registers. One register, (PPCTL), contains control and status. Two registers, (RXPP and TXPP), are used for buffering receive and transmit operations. Six registers are used for DMA functionality—IIPP, IMPP, ICPP, EIPP, EMPP, and ECPP.

Table A-13. Parallel Port Registers

Function	Registers	Description
Control and Status Register	PPCTL	"Parallel Port Control Register (PPCTL)" on page A-55
Buffering Receive and Transmit Data	TXPP	"Parallel Port DMA Transmit Register (TXPP)" on page A-56
	RXPP	"Parallel Port DMA Receive Register (RXPP)" on page A-58

Table A-13. Parallel Port Registers (Cont'd)

Function	Registers	Description
DMA functionality	IIPP	"Parallel Port DMA Start Internal Index Address Register (IIPP)" on page A-59r
	IMPP	"Parallel Port DMA Internal Modifier Address Register (IMPP)" on page A-59
	ICPP	"Parallel Port DMA Internal Word Count Register (ICPP)" on page A-59
	EIPP	"Parallel Port DMA Start External Index Address Register (EIPP)" on page A-59
	EMPP	"Parallel Port DMA External Modifier Address Register (EMPP)" on page A-59
	ECPP	"Parallel Port DMA External Word Count Register (ECPP)" on page A-60

Parallel Port Control Register (PPCTL)

The Parallel Port Control Register (PPCTL) is used to configure and enable the parallel port system. This register's address is 0x1800. Figure A-21 describes the bit fields within this register.

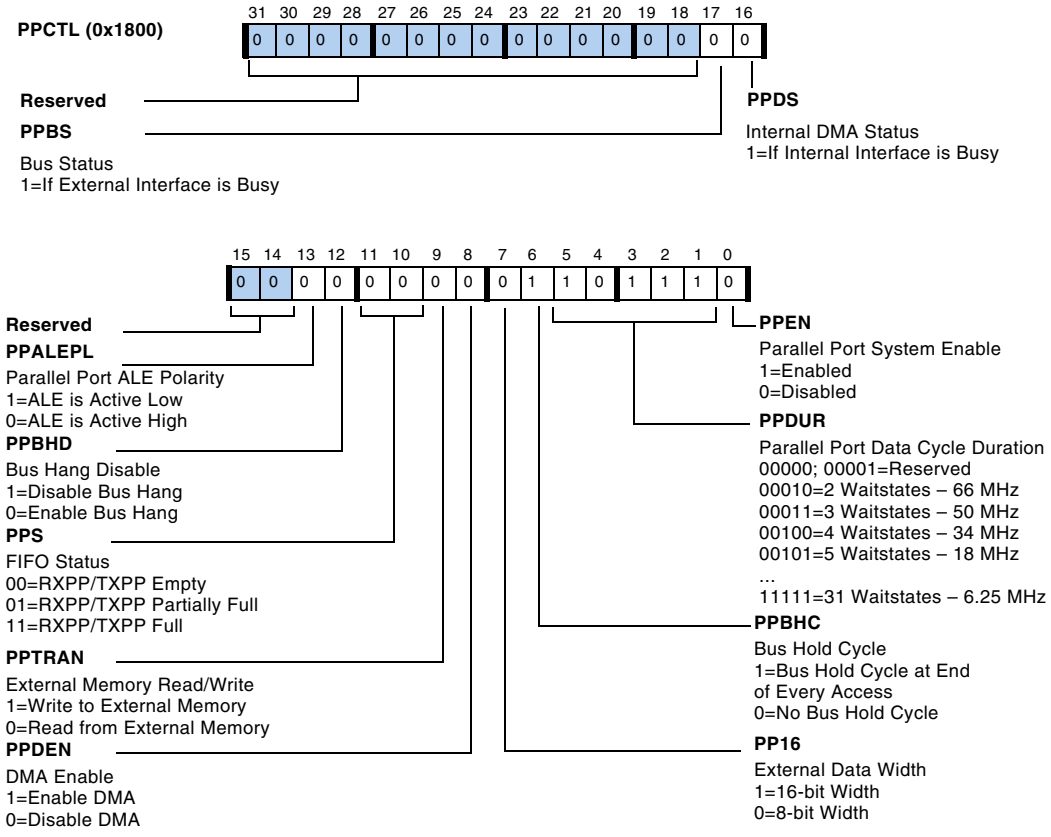


Figure A-21. PPCTL Register

Parallel Port DMA Transmit Register (TXPP)

This register's address is 0x1808. This Transmit Data register is a 32-bit register that is part of the IOP register set and can be accessed by the core or the DMA controller. Data is loaded into this register before being transmitted. Prior to the beginning of a data transfer, data in the TXPP register is automatically loaded into the Transmit Shift register. During a DMA transmit operation, the data in TXPP is automatically loaded from internal memory.

Table A-14. Parallel Port Register (PPCTL) Bit Definitions

Bit	Name	Definition	Default
0	PPEN	**Parallel Port Enable.** Enables if set, (= 1) or disables if cleared, (= 0) the parallel port. Clearing this bit clears FIFO and status. If an $\overline{RD}$, $\overline{WR}$, or ALE cycle has already started, it completes normally before the port is disabled. The parallel port is ready to transmit or receive 2 cycles after enabling. An ALE cycle always occurs before the first read or write cycle after PPEN is enabled.	0
5–1	PPDUR	**Parallel Port Duration.** The duration of Parallel Port data cycles is based on core-clock and controlled by these five bits as follows: 00000 = Reserved 00001 = Reserved 00010 = 2 Wait States = 3 Core Clock Cycles; 66 MHz throughput 00011 = 3 Wait States = 4 Core Clock Cycles; 50 MHz throughput 00100 = 4 Wait States = 5 Core Clock Cycles; 40 MHz throughput 00101 = 5 Wait States = 6 Core Clock Cycles; 33 MHz throughput ... 11111 = 31 wait states; 6.25MHz throughput	Bit 1 = 1 Bit 2=1 Bit 3=1 Bit 4=0 Bit 5=1
6	PPBHC	**Bus Hold Cycle.** Inserts a bus hold cycle at the end of every access (read or write cycle) if set, (= 1) or no bus hold cycle occurs if cleared, (= 0). During a BHC address and/or data continue to be driven for one cycle.	1
7	PP16	**Parallel Port External Data Width.** Selects the external data width to 16 bits if set, (= 1) or 8 bits if cleared, (= 0).	0
8	PPDEN	**Parallel Port DMA Enable.** Enables if set, (= 1) DMA on the parallel port or disables DMA if cleared, (= 0). When PPDEN is cleared, any DMA requests already in the pipeline complete, and no new DMA requests are made. This does not affect FIFO status.	0
9	PPTRAN	**Parallel Port Transmit/Receive Select.** Indicates if the processor is reading from external memory if cleared, (= 0) or writing to external memory if set, (= 1).	0

Table A-14. Parallel Port Register (PPCTL) Bit Definitions (Cont'd)

Bit	Name	Definition	Default
11-10	PPS	**Parallel Port FIFO Status.** These read-only bits indicate the status of the parallel port FIFO as follows: 00 = RXPP/TXPP is empty 01 = RXPP/TXPP is partially full 11 = RXPP/TXPP is full	0
12	PPBHD	**Parallel Port Buffer Hang Disable.** When cleared (= 0), core stalls occur normally when the core attempts to write to a full transmit buffer or read from an empty receive buffer. Prevents a core hang when set (= 1). The old data present in the receive buffer is read again if the core tries to read it. If a write to the transmit buffer is performed, the core will overwrite the current data in the buffer.	0
13	PPALEPL	**Parallel Port ALE Polarity Level.** Asserts ALE active low if set, (= 1) or active high if cleared, (= 0).	0
15–14	Reserved		0
16	PPDS	**DMA Status.** Indicates that the internal DMA interface is active if set, (= 1) or not active if cleared, (= 0).	0
17	PPBS	**Parallel Port Bus Status.** Indicates that the external bus interface is busy if set, (= 1) or available if cleared, (= 0). The bus will be "busy" until one ALE cycle, $\overline{\text{RD}}$ cycle or $\overline{\text{WR}}$ cycle has taken place.	0
31–18	Reserved		0

Parallel Port DMA Receive Register (RXPP)

This register's address is 0x1809. This is a 32-bit read-only register accessible by the core or the DMA controller. At the end of a data transfer, RXPP is loaded with the data in the shift register. During a DMA receive operation, the data in the RXPP register is automatically loaded into the internal memory. For core or interrupt driven transfer, you can also use the RXS status bits in the PPSTAT register to determine if the receive buffer is full.

Parallel Port DMA Start Internal Index Address Register (IIPP)

This register's address is 0x1818. This 19-bit register contains the offset from the DMA starting address of 32-bit internal memory.

Parallel Port DMA Internal Modifier Address Register (IMPP)

This register's address is 0x1819. This 16-bit register contains the internal memory DMA address modifier.

Parallel Port DMA Internal Word Count Register (ICPP)

This register's address is 0x181A. This 16-bit register contains the number of words in internal memory to be transferred via DMA.

Parallel Port DMA Start External Index Address Register (EIPP)

This register's address is 0x1810. This 24-bit register contains the external memory DMA address index.

Parallel Port DMA External Modifier Address Register (EMPP)

This register's address is 0x1811. This 2-bit register contains the external memory DMA address modifier. It supports only +1, 0, -1.

Parallel Port DMA External Word Count Register (ECPP)

This register's address is 0x1812. This 24-bit register contains the number of words in external memory to be transferred via DMA.

Signal Routing Unit Registers

The Digital Audio Interface (DAI) is comprised of a group of peripherals and the signal routing unit (SRU).

The SRU is a matrix routing unit that enables the peripherals provided by the DAI and serial ports to be interconnected under software control. This removes the limitations associated with hard-wiring audio or other peripherals to each other and to the rest of the processor. The SRU allows the programs to make optimal use of the peripherals for a wide variety of applications. This flexibility enables a much larger set of algorithms than would be possible with non-configurable signal paths.

The SRU provides groups of control registers, described in the sections that follow. The registers define the interconnections between the functional modules within the DAI as well as to the core and to the pins. The SRU is a series of multiplexers that connect the:

- serial ports, `SPORT[5:0]`
- input data port, (IDP) `IDP[7:0]` including the parallel data acquisition port pins and their output enable drivers: `DAI_PB[20:1]`
- precision clock generators, (`PCG_CTLA_1` and `PCG_CTLB_1`)

Each of these modules is separated from each other by the SRU, and their input and output signals (the "junctions") may only be connected via the SRU.

Clock Routing Control Registers (Group A)

The Clock Routing Control registers route a serial data clock, a sample clock, and signals to the SPORTs and the Input Data Port (IDP) channels. Each of the clock inputs specified are connected to a clock source, based on the five bit values in the Table A-15. When either of the precision clock generators is in external source mode, the `SRU_CLK3[4:0]` and/or `SRU_CLK3[9:5]` bits specify the source.

The Clock Routing Control registers correspond to the Group A clock sources, listed in Table A-15. Thirty-two possible clock sources can be connected using these read/write registers:

- `SRU_CLK0`, described in Figure A-22
- `SRU_CLK1`, described in Figure A-23
- `SRU_CLK2`, described in Figure A-24 on page A-62
- `SRU_CLK3`, described in Figure A-25 on page A-63

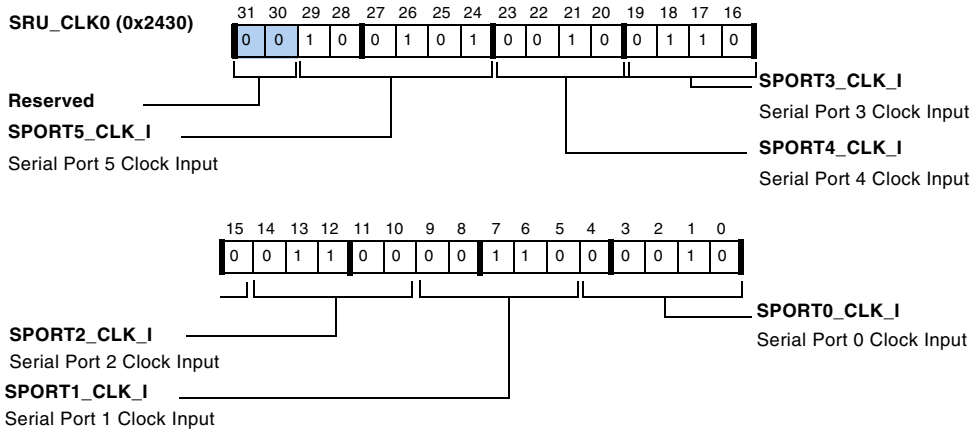


Figure A-22. SRU_CLK0 Register

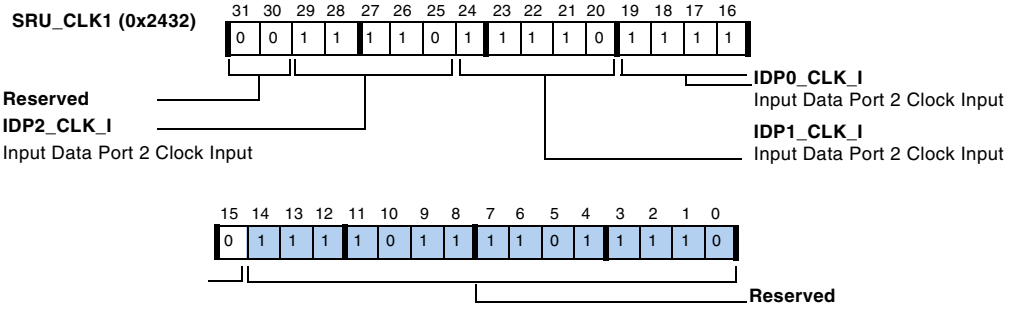


Figure A-23. SRU_CLK1 Register

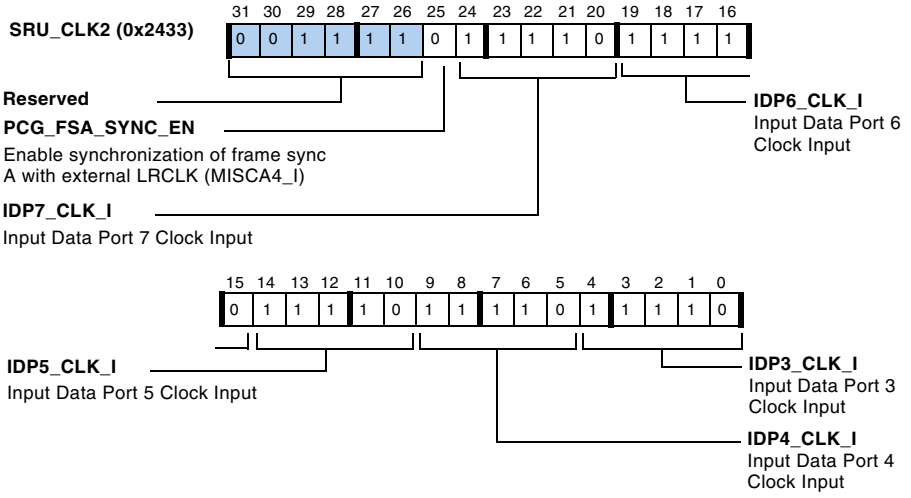


Figure A-24. SRU_CLK2 Register

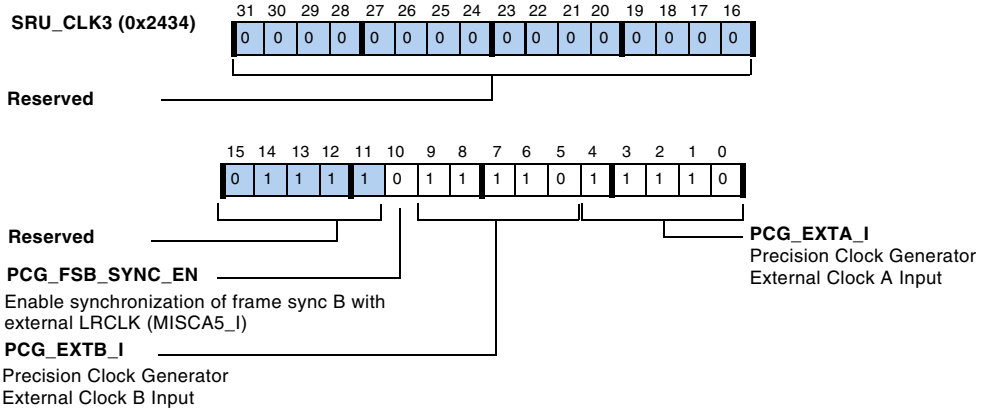


Figure A-25. SRU_CLK3 Register

Table A-15. Group A Sources – Serial Clock

Selection Code	Source Signal	Description
00000 (0x0)	DAI_PB01_O	Select DAI Pin Buffer 1 as the source
00001 (0x1)	DAI_PB02_O	Select DAI Pin Buffer 2 as the source
00010 (0x2)	DAI_PB03_O	Select DAI Pin Buffer 3 as the source
00011 (0x3)	DAI_PB04_O	Select DAI Pin Buffer 4 as the source
00100 (0x4)	DAI_PB05_O	Select DAI Pin Buffer 5 as the source
00101 (0x5)	DAI_PB06_O	Select DAI Pin Buffer 6 as the source
00110 (0x6)	DAI_PB07_O	Select DAI Pin Buffer 7 as the source
00111 (0x7)	DAI_PB08_O	Select DAI Pin Buffer 8 as the source
01000 (0x8)	DAI_PB09_O	Select DAI Pin Buffer 9 as the source
01001 (0x9)	DAI_PB10_O	Select DAI Pin Buffer 10 as the source
01010 (0xA)	DAI_PB11_O	Select DAI Pin Buffer 11 as the source
01011 (0xB)	DAI_PB12_O	Select DAI Pin Buffer 12 as the source
01100 (0xC)	DAI_PB13_O	Select DAI Pin Buffer 13 as the source

Table A-15. Group A Sources – Serial Clock (Cont'd)

Selection Code	Source Signal	Description
01101 (0xD)	DAI_PB14_O	Select DAI Pin Buffer 14 as the source
01110 (0xE)	DAI_PB15_O	Select DAI Pin Buffer 15 as the source
01111 (0xF)	DAI_PB16_O	Select DAI Pin Buffer 16 as the source
10000 (0x10)	DAI_PB17_O	Select DAI Pin Buffer 17 as the source
10001 (0x11)	DAI_PB18_O	Select DAI Pin Buffer 18 as the source
10010 (0x12)	DAI_PB19_O	Select DAI Pin Buffer 19 as the source
10011 (0x13)	DAI_PB20_O	Select DAI Pin Buffer 20 as the source
10100 (0x14)	SPORT0_CLK_O	Select SPORT 0 Clock as the source
10101 (0x15)	SPORT1_CLK_O	Select SPORT 1 Clock as the source
10110 (0x16)	SPORT2_CLK_O	Select SPORT 2 Clock as the source
10111 (0x17)	SPORT3_CLK_O	Select SPORT 3 Clock as the source
11000 (0x18)	SPORT4_CLK_O	Select SPORT 4 Clock as the source
11001 (0x19)	SPORT5_CLK_O	Select SPORT 5 Clock as the source
11010 (0x1A)	Reserved	
11011 (0x1B)	Reserved	
11100 (0x1C)	PCG_CLKA_O	Select Precision Clock A Output as the source
11101 (0x1D)	PCG_CLKB_O	Select Precision Clock B Output as the source
11110 (0x1E)	LOW	Select Logic Level Low (0) as the source
11111 (0x1F)	HIGH	Select Logic Level High (1) as the source

Setting `SRU_CLK3[4:0]` = 28 connects `PCG_EXTA_I` to logic low, not to `PCG_CLKA_O`.

Setting `SRU_CLK3[9:5]` = 29 connects `PCG_EXTB_I` to logic low, not to `PCG_CLKB_O`.

Serial Data Routing Registers (SRU_DATx, Group B)

The Serial Data Routing Control registers route serial data to the serial ports (a, b) and the IDP. Each of the data inputs specified are connected to a data source based on the six bit values shown in Table A-16. Sixty-four possible data sources can be designated for these registers:

- SRU_DAT0, described in Figure A-26
- SRU_DAT1, described in Figure A-27
- SRU_DAT2, described in Figure A-28
- SRU_DAT3, described in Figure A-29
- SRU_DAT4, described in Figure A-30

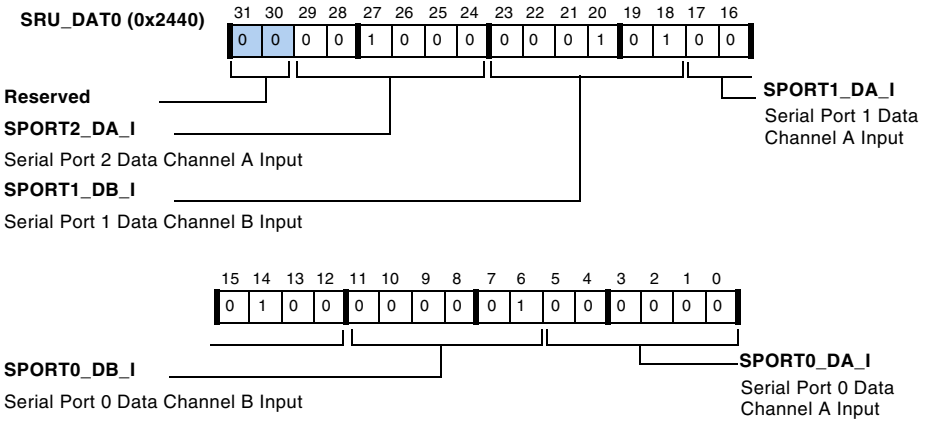


Figure A-26. SRU_DAT0 Register

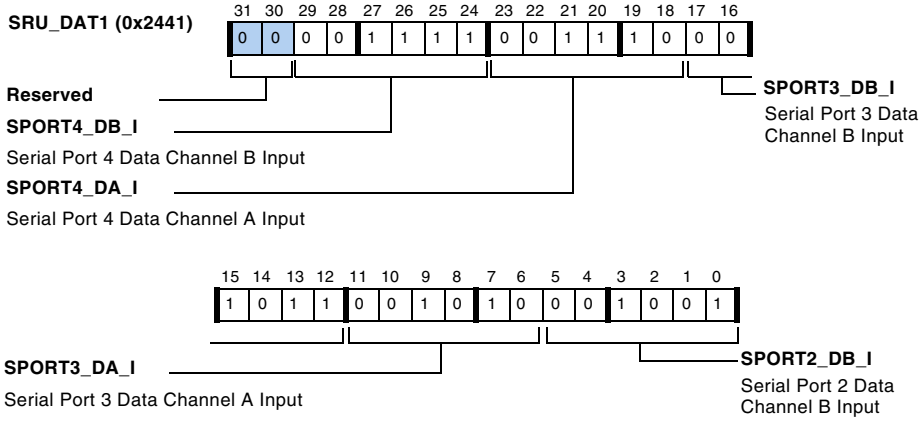


Figure A-27. SRU_DAT1 Register

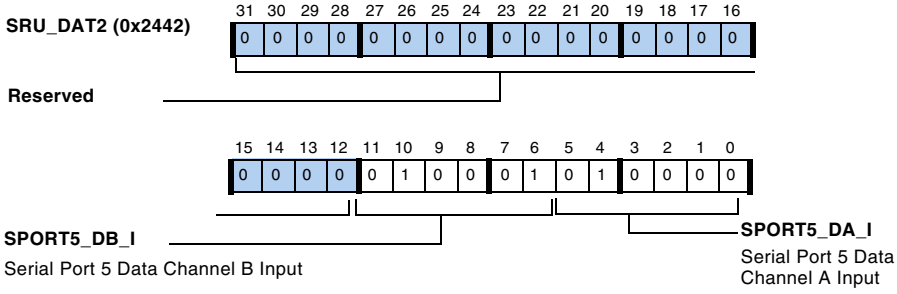


Figure A-28. SRU_DAT2 Register

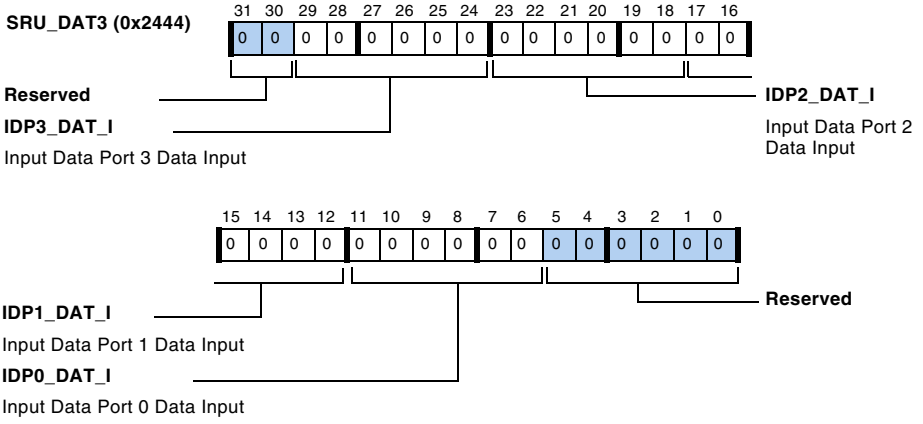


Figure A-29. SRU_DAT3 Register

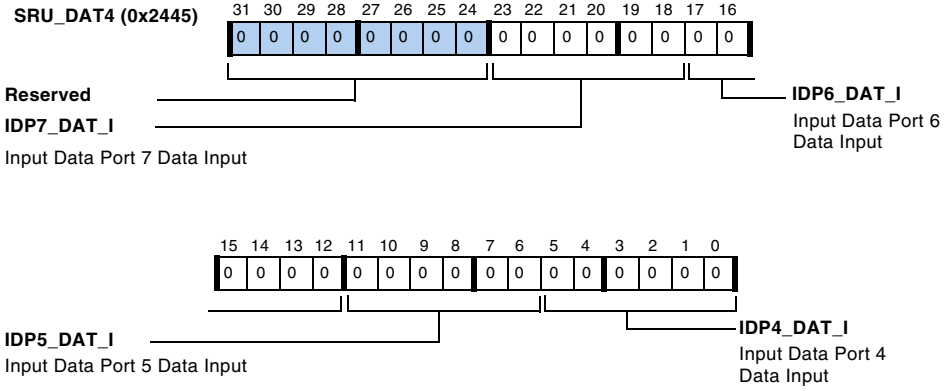


Figure A-30. SRU_DAT4 Register

Table A-16. Group B Sources – Serial Data

Selection Code	Source Signal	Description
000000 (0x0)	DAI_PB01_O	Select DAI Pin Buffer 1 as the source
000001 (0x1)	DAI_PB02_O	Select DAI Pin Buffer 2 as the source
000010 (0x2)	DAI_PB03_O	Select DAI Pin Buffer 3 as the source
000011 (0x3)	DAI_PB04_O	Select DAI Pin Buffer 4 as the source
000100 (0x4)	DAI_PB05_O	Select DAI Pin Buffer 5 as the source
000101 (0x5)	DAI_PB06_O	Select DAI Pin Buffer 6 as the source
000110 (0x6)	DAI_PB07_O	Select DAI Pin Buffer 7 as the source
000111 (0x7)	DAI_PB08_O	Select DAI Pin Buffer 8 as the source
001000 (0x8)	DAI_PB09_O	Select DAI Pin Buffer 9 as the source
001001 (0x9)	DAI_PB10_O	Select DAI Pin Buffer 10 as the source
001010 (0xA)	DAI_PB11_O	Select DAI Pin Buffer 11 as the source
001011 (0xB)	DAI_PB12_O	Select DAI Pin Buffer 12 as the source
001100 (0xC)	DAI_PB13_O	Select DAI Pin Buffer 13 as the source
001101 (0xD)	DAI_PB14_O	Select DAI Pin Buffer 14 as the source
001110 (0xE)	DAI_PB15_O	Select DAI Pin Buffer 15 as the source
001111 (0xF)	DAI_PB16_O	Select DAI Pin Buffer 16 as the source
010000 (0x10)	DAI_PB17_O	Select DAI Pin Buffer 17 as the source
010001 (0x11)	DAI_PB18_O	Select DAI Pin Buffer 18 as the source
010010 (0x12)	DAI_PB19_O	Select DAI Pin Buffer 19 as the source
010011 (0x13)	DAI_PB20_O	Select DAI Pin Buffer 20 as the source
010100 (0x14)	SPORT0_DA_O	Select SPORT 0A data as the source
010101 (0x15)	SPORT0_DB_O	Select SPORT 0B data as the source
010110 (0x16)	SPORT1_DA_O	Select SPORT 1A data as the source
010111 (0x17)	SPORT1_DB_O	Select SPORT 1B data as the source

Table A-16. Group B Sources – Serial Data (Cont'd)

Selection Code	Source Signal	Description
011000 (0x18)	SPORT2_DA_O	Select SPORT 2A data as the source
011001 (0x19)	SPORT2_DB_O	Select SPORT 2B data as the source
011010 (0x1A)	SPORT3_DA_O	Select SPORT 3A data as the source
011011 (0x1B)	SPORT3_DB_O	Select SPORT 3B data as the source
011100 (0x1C)	SPORT4_DA_O	Select SPORT 4A data as the source
011101 (0x1D)	SPORT4_DB_O	Select SPORT 4B data as the source
011110 (0x1E)	SPORT5_DA_O	Select SPORT 5A data as the source
011111 (0x1F)	SPORT5_DB_O	Select SPORT 5B data as the source
100000 (0x20) – 101001 (0x29)	Reserved	
101010 (0x2A)	LOW	Select Logic Level Low (0) as the source
101011 (0x2B)	HIGH	Select Logic Level High (1) as the source
101100 (0x2C) – 111111 (0x3F)	Reserved	

Frame Sync Routing Control Registers (SRU_FSx, Group C)

The Frame Sync Routing Control registers route frame sync, or a word clock, to the serial ports and IDP. Each of the frame sync inputs specified is connected to a frame sync source based on the values described in the Group C frame sync sources listed in Table A-17. Thirty-two possible frame sync sources can be connected using these registers:

- SRU_FS0, described in Figure A-31
- SRU_FS1, described in Figure A-32
- SRU_FS2, described in Figure A-33

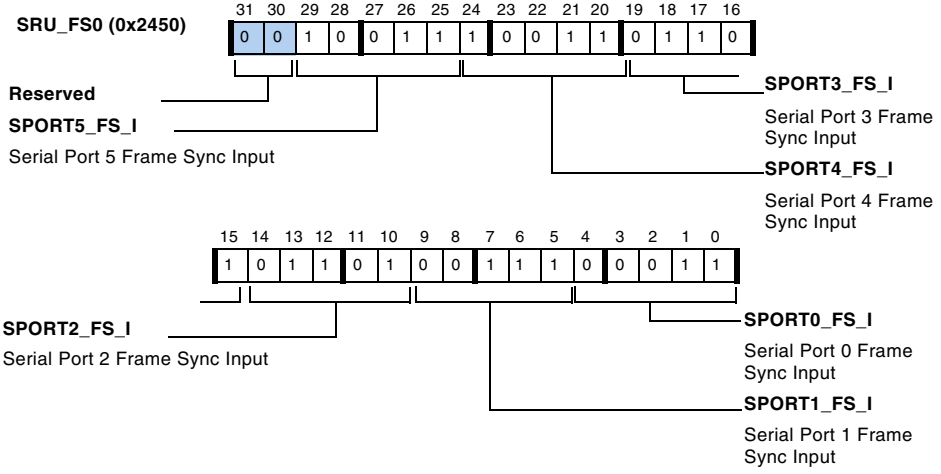


Figure A-31. SRU_FS0 Register

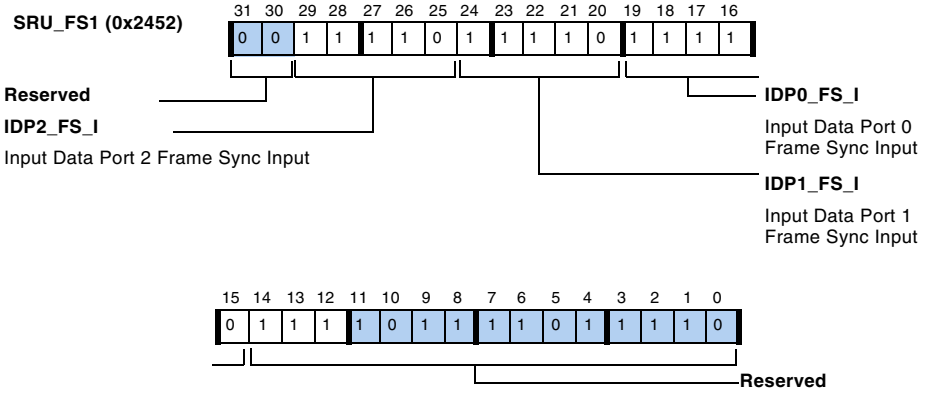


Figure A-32. SRU_FS1 Register

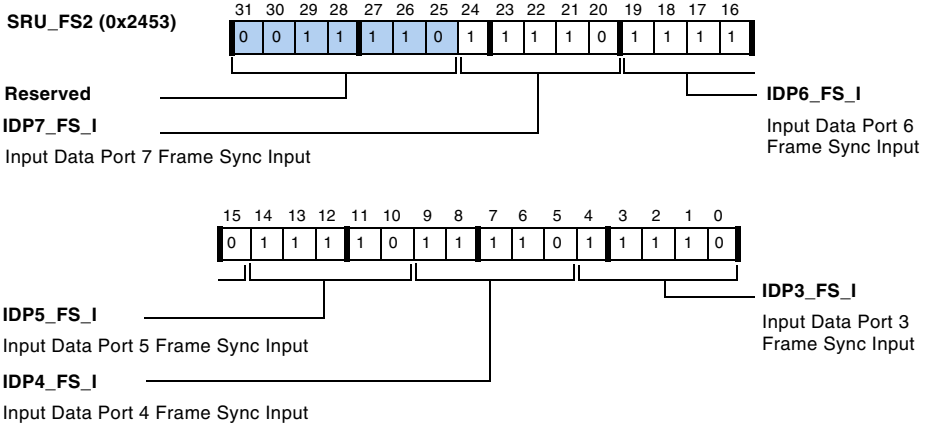


Figure A-33. SRU_FS2 Register

Table A-17. Group C Sources – Frame Sync

Selection Code	Source Signal	Description
00000 (0x0)	DAI_PB01_O	Select DAI Pin Buffer 1 as the source
00001 (0x1)	DAI_PB02_O	Select DAI Pin Buffer 2 as the source
00010 (0x2)	DAI_PB03_O	Select DAI Pin Buffer 3 as the source
00011 (0x3)	DAI_PB04_O	Select DAI Pin Buffer 4 as the source
00100 (0x4)	DAI_PB05_O	Select DAI Pin Buffer 5 as the source
00101 (0x5)	DAI_PB06_O	Select DAI Pin Buffer 6 as the source
00110 (0x6)	DAI_PB07_O	Select DAI Pin Buffer 7 as the source
00111 (0x7)	DAI_PB08_O	Select DAI Pin Buffer 8 as the source
01000 (0x8)	DAI_PB09_O	Select DAI Pin Buffer 9 as the source
01001 (0x9)	DAI_PB10_O	Select DAI Pin Buffer 10 as the source
01010 (0xA)	DAI_PB11_O	Select DAI Pin Buffer 11 as the source
01011 (0xB)	DAI_PB12_O	Select DAI Pin Buffer 12 as the source
01100 (0xC)	DAI_PB13_O	Select DAI Pin Buffer 13 as the source
01101 (0xD)	DAI_PB14_O	Select DAI Pin Buffer 14 as the source
01110 (0xE)	DAI_PB15_O	Select DAI Pin Buffer 15 as the source
01111 (0xF)	DAI_PB16_O	Select DAI Pin Buffer 16 as the source
10000 (0x10)	DAI_PB17_O	Select DAI Pin Buffer 17 as the source
10001 (0x11)	DAI_PB18_O	Select DAI Pin Buffer 18 as the source
10010 (0x12)	DAI_PB19_O	Select DAI Pin Buffer 19 as the source
10011 (0x13)	DAI_PB20_O	Select DAI Pin Buffer 20 as the source
10100 (0x14)	SPORT0_FS_O	Select SPORT 0 Frame Sync as the source
10101 (0x15)	SPORT1_FS_O	Select SPORT 1 Frame Sync as the source
10110 (0x16)	SPORT2_FS_O	Select SPORT 2 Frame Sync as the source
10111 (0x17)	SPORT3_FS_O	Select SPORT 3 Frame Sync as the source

Table A-17. Group C Sources – Frame Sync (Cont'd)

Selection Code	Source Signal	Description
11000 (0x18)	SPORT4_FS_O	Select SPORT 4 Frame Sync as the source
11001 (0x19)	SPORT5_FS_O	Select SPORT 5 Frame Sync as the source
11010 (0x1A)	Reserved	
11011 (0x1B)	Reserved	
11100 (0x1C)	PCG_FSA_O	Select Precision Frame Sync A Output as the source
11101 (0x1D)	PCG_FSB_O	Select Precision Frame Sync B Output as the source
11110 (0x1E)	LOW	Select Logic Level Low (0) as the source
11111 (0x1F)	HIGH	Select Logic Level High (1) as the source

Pin Signal Assignment Registers (SRU_PINx, Group D)

Each physical pin (connected to a bonded pad) can be routed via the SRU to any of the inputs or outputs of the DAI peripherals, based on the 6-bit values listed in Table A-18. The SRU can also be used to route signals that control the pins in other ways. The pin signal assignments are shown in the following figures:

- `SRU_PIN0`, described in Figure A-34
- `SRU_PIN1`, described in Figure A-35
- `SRU_PIN2`, described in Figure A-36
- `SRU_PIN3`, described in Figure A-37

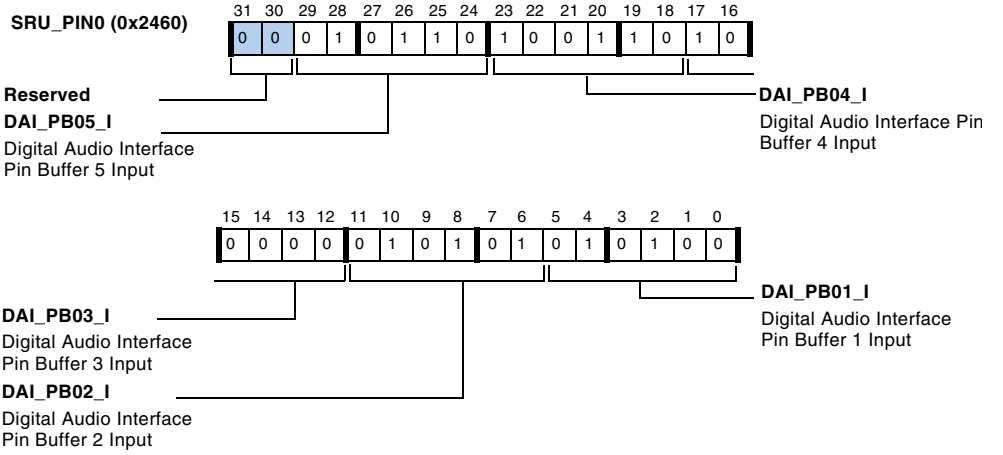


Figure A-34. SRU_PIN0 Register

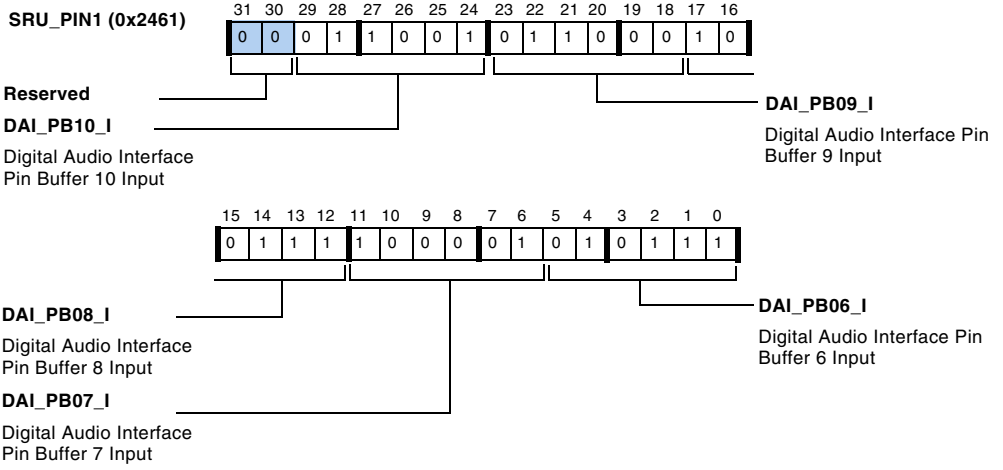


Figure A-35. SRU_PIN1 Register

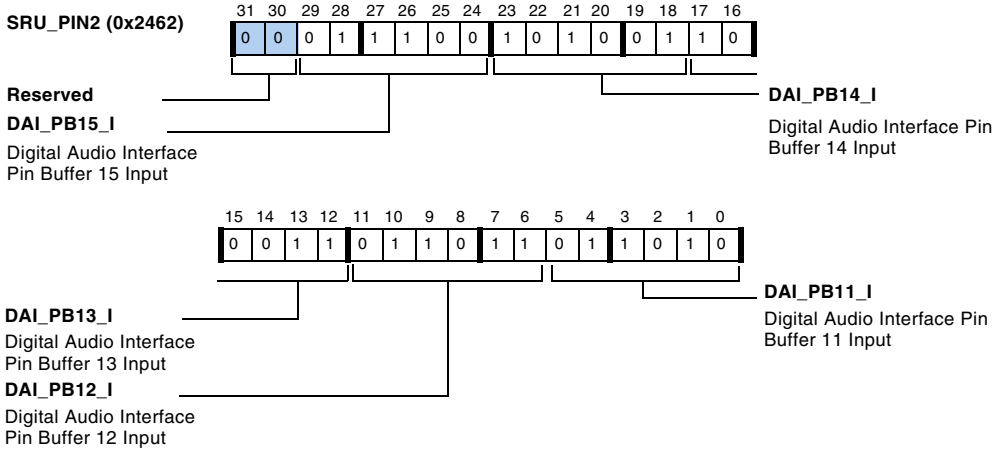


Figure A-36. SRU_PIN2 Register

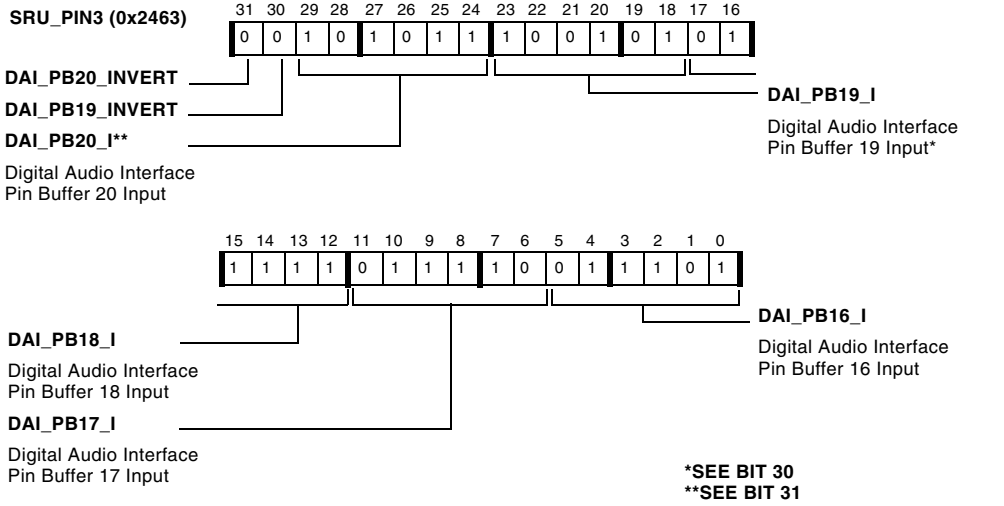


Figure A-37. SRU_PIN3 Register

Setting the SRU_PIN3 bit 30 to HIGH inverts the level of the DAI_PB18_I pin. Setting the SRU_PIN3[31] bit to HIGH inverts the level of DAI_PB19_I.

If the SRU_PIN3[23:18] bits = 18, then setting SRU_PIN3[30] to HIGH does not invert the output. If the SRU_PIN3[29:24] bits = 19, then setting SRU_PIN3[31] to HIGH does not invert the output.

Table A-18. Group D Sources – Pin Signal Assignments

Selection Code	Source Signal	Description
000000 (0x0)	DAI_PB01_O	Select DAI Pin Buffer 1 as the source
000001 (0x1)	DAI_PB02_O	Select DAI Pin Buffer 2 as the source
000010 (0x2)	DAI_PB03_O	Select DAI Pin Buffer 3 as the source
000011 (0x3)	DAI_PB04_O	Select DAI Pin Buffer 4 as the source
000100 (0x4)	DAI_PB05_O	Select DAI Pin Buffer 5 as the source
000101 (0x5)	DAI_PB06_O	Select DAI Pin Buffer 6 as the source
000110 (0x6)	DAI_PB07_O	Select DAI Pin Buffer 7 as the source
000111 (0x7)	DAI_PB08_O	Select DAI Pin Buffer 8 as the source
001000 (0x8)	DAI_PB09_O	Select DAI Pin Buffer 9 as the source
001001 (0x9)	DAI_PB10_O	Select DAI Pin Buffer 10 as the source
001010 (0xA)	DAI_PB11_O	Select DAI Pin Buffer 11 as the source
001011 (0xB)	DAI_PB12_O	Select DAI Pin Buffer 12 as the source
001100 (0xC)	DAI_PB13_O	Select DAI Pin Buffer 13 as the source
001101 (0xD)	DAI_PB14_O	Select DAI Pin Buffer 14 as the source
001110 (0xE)	DAI_PB15_O	Select DAI Pin Buffer 15 as the source
001111 (0xF)	DAI_PB16_O	Select DAI Pin Buffer 16 as the source
010000 (0x10)	DAI_PB17_O	Select DAI Pin Buffer 17 as the source
010001 (0x11)	DAI_PB18_O	Select DAI Pin Buffer 18 as the source
010010 (0x12)	DAI_PB19_O	Select DAI Pin Buffer 19 as the source

Table A-18. Group D Sources – Pin Signal Assignments (Cont'd)

Selection Code	Source Signal	Description
010011 (0x13)	DAI_PB20_O	Select DAI Pin Buffer 20 as the source
010100 (0x14)	SPORT0_DA_O	Select SPORT 0A Data as the source
010101 (0x15)	SPORT0_DB_O	Select SPORT 0B Data as the source
010110 (0x16)	SPORT1_DA_O	Select SPORT 1A Data as the source
010111 (0x17)	SPORT1_DB_O	Select SPORT 1B Data as the source
011000 (0x18)	SPORT2_DA_O	Select SPORT 2A Data as the source
011001 (0x19)	SPORT2_DB_O	Select SPORT 2B Data as the source
011010 (0x1A)	SPORT3_DA_O	Select SPORT 3A Data as the source
011011 (0x1B)	SPORT3_DB_O	Select SPORT 3B Data as the source
011100 (0x1C)	SPORT4_DA_O	Select SPORT 4A Data as the source
011101 (0x1D)	SPORT4_DB_O	Select SPORT 4B Data as the source
011110 (0x1E)	SPORT5_DA_O	Select SPORT 5A Data as the source
011111 (0x1F)	SPORT5_DB_O	Select SPORT 5B Data as the source
100000 (0x20)	SPORT0_CLK_O	Select SPORT 0 Clock as the source
100001 (0x21)	SPORT1_CLK_O	Select SPORT 1 Clock as the source
100010 (0x22)	SPORT2_CLK_O	Select SPORT 2 Clock as the source
100011 (0x23)	SPORT3_CLK_O	Select SPORT 3 Clock as the source
100100 (0x24)	SPORT4_CLK_O	Select SPORT 4 Clock as the source
100101 (0x25)	SPORT5_CLK_O	Select SPORT 5 Clock as the source
100110 (0x26)	SPORT0_FS_O	Select SPORT 0 Frame Sync as the source
100111 (0x27)	SPORT1_FS_O	Select SPORT 1 Frame Sync as the source
101000 (0x28)	SPORT2_FS_O	Select SPORT 2 Frame Sync as the source
101001 (0x29)	SPORT3_FS_O	Select SPORT 3 Frame Sync as the source
101010 (0x2A)	SPORT4_FS_O	Select SPORT 4 Frame Sync as the source

Table A-18. Group D Sources – Pin Signal Assignments (Cont'd)

Selection Code	Source Signal	Description
101011 (0x2B)	SPORT5_FS_O	Select SPORT 5 Frame Sync as the source
101100 (0x2C)	TIMER0_O	Select Timer 0 as the source
101101 (0x2D)	TIMER1_O	Select Timer 1 as the source
101110 (0x2E)	TIMER2_O	Select Timer 2 as the source
101111 (0x2F)	FLAG10_O	Select Flag 10 as the source[1]
110000 (0x30)	MISCB_2_O	Select Miscellaneous Control B-2 as the source
110001 (0x31)	MISCB_3_O	Select Miscellaneous Control B-3 as the source
110010 (0x32)	MISCB_4_O	Select Miscellaneous Control B-4 as the source
110011 (0x33)	MISCB_5_O	Select Miscellaneous Control B-5 as the source
110100 (0x34)	FLAG11_O	Select Flag 11 as the source
110101 (0x35)	FLAG12_O	Select Flag 12 as the source
110110 (0x36)	FLAG13_O	Select Flag 13 as the source
110111 (0x37)	FLAG14_O	Select Flag 14 as the source
111000 (0x38)	PCG_CLKA_O	Select Precision Clock A as the source
111001 (0x39)	PCG_CLKB_O	Select Precision Clock B as the source
111010 (0x3A)	PCG_FSA_O	Select Precision Frame Sync A as the source
111011 (0x3B)	PCG_FSB_O	Select Precision Frame Sync B as the source
111100 (0x3C)	FLAG15_O	Select Flag 15 as the source
111101 (0x3D)	Reserved	
111110 (0x3E)	LOW	Select Logic Level Low (0) as the source
111111 (0x3F)	HIGH	Select Logic Level High (1) as the source

1 The ADSP-2126x SHARC processor supports 16 flags including:
-Four pins in the IRQ,
-6 (FLAG10-15) in the Digital Application Interface (DAI), and
-16 address pins in the Parallel Port can act as flags.
Parallel ports function as flags when the PPFLGS bit (bit 20) of the SYSCTL register is set (= 1). This bit causes the 16 address pins to function as FLAG0–FLAG15.
The IRQ acts as a flag that also generates an interrupt.

Miscellaneous SRU Registers (SRU_EXT_MISCx, Group E)

The Miscellaneous Signal Routing registers are a very powerful and versatile feature of the DAI. These registers allow external pins, timers, and clocks to serve as interrupt sources or timer inputs and outputs. They also allow pins to connect to other pins, or to invert the logic of other pins. Note that `MISCB2_I`, `MISCB3_I`, `MISCB4_I`, and `MISCB5_I` may be mapped directly to the DAI pin buffers using Group D registers (see Table A-18).

The Miscellaneous Signal Routing registers correspond to the Group E miscellaneous signals listed in Table A-19. Thirty-two possible signal sources can be connected using these read/write registers:

- `SRU_EXT_MISCA`, described in Figure A-38 on page A-80
- `SRU_EXT_MISCB`, described in Figure A-39 on page A-81

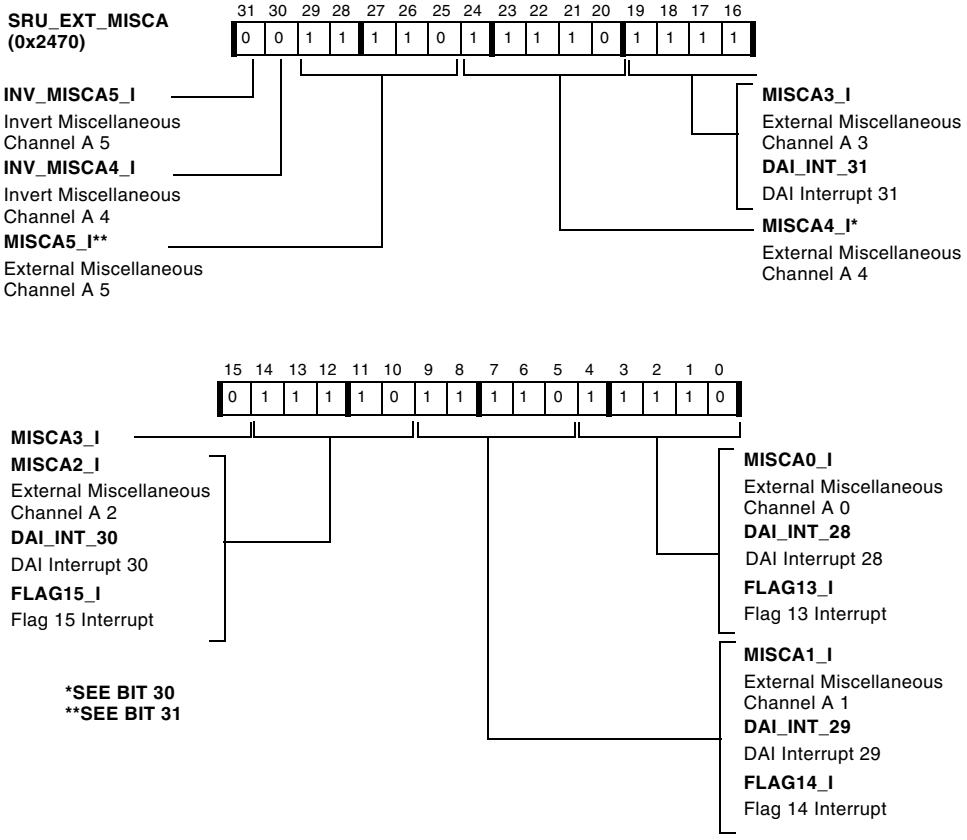


Figure A-38. SRU_EXT_MISCA Register

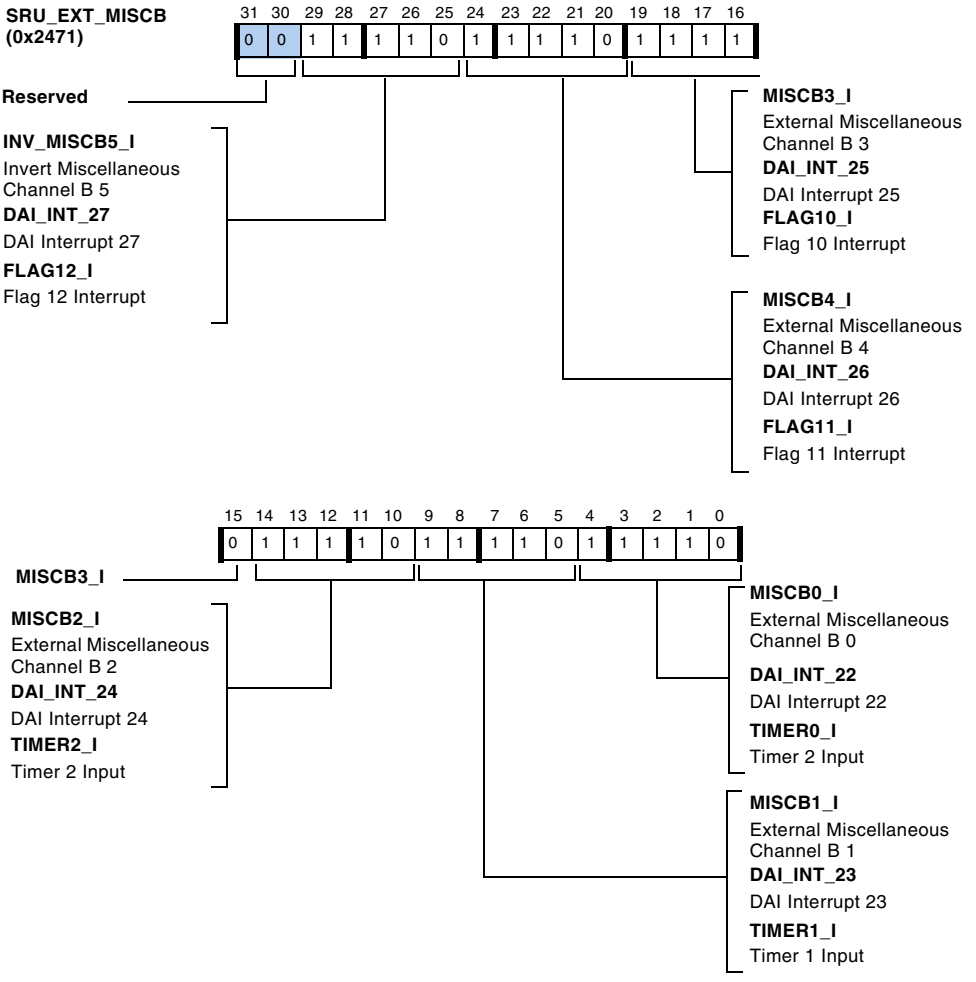


Figure A-39. SRU_EXT_MISCB Register

Setting the SRU_EXT_MISCA[30] bit to HIGH inverts the level of MISCA4_I, and setting the SRU_EXT_MISCA[31] bit to HIGH inverts the level of MISCA5_I.

Table A-19. Group E Sources – Miscellaneous Signals

Selection Code	Source Signal	Description
00000 (0x0)	DAI_PB01_O	Select DAI Pin Buffer 1 Output as the source
00001 (0x1)	DAI_P02_O	Select DAI Pin Buffer 2 Output as the source
00010 (0x2)	DAI_P03_O	Select DAI Pin Buffer 3 Output as the source
00011 (0x3)	DAI_P04_O	Select DAI Pin Buffer 4 Output as the source
00100 (0x4)	DAI_P05_O	Select DAI Pin Buffer 5 Output as the source
00101 (0x5)	DAI_P06_O	Select DAI Pin Buffer 6 Output as the source
00110 (0x6)	DAI_P07_O	Select DAI Pin Buffer 7 Output as the source
00111 (0x7)	DAI_P08_O	Select DAI Pin Buffer 8 Output as the source
01000 (0x8)	DAI_P09_O	Select DAI Pin Buffer 9 Output as the source
01001 (0x9)	DAI_P10_O	Select DAI Pin Buffer 10 Output as the source
01010 (0xA)	DAI_P11_O	Select DAI Pin Buffer 11 Output as the source
01011 (0xB)	DAI_P12_O	Select DAI Pin Buffer 12 Output as the source
01100 (0xC)	DAI_P13_O	Select DAI Pin Buffer 13 Output as the source
01101 (0xD)	DAI_P14_O	Select DAI Pin Buffer 14 Output as the source
01110 (0xE)	DAI_P15_O	Select DAI Pin Buffer 15 Output as the source
01111 (0xF)	DAI_P16_O	Select DAI Pin Buffer 16 Output as the source
10000 (0x10)	DAI_P17_O	Select DAI Pin Buffer 17 Output as the source
10001 (0x11)	DAI_P18_O	Select DAI Pin Buffer 18 Output as the source
10010 (0x12)	DAI_P19_O	Select DAI Pin Buffer 19 Output as the source
10011 (0x13)	DAI_P20_O	Select DAI Pin Buffer 20 Output as the source
10100 (0x14)	TIMER0_O	Select Timer 0 Output as the source
10101 (0x15)	TIMER1_O	Select Timer 1 Output as the source
10110 (0x16)	TIMER2_O	Select Timer 2 Output as the source
10111 (0x17); 11000 (0x18)	Reserved	

Table A-19. Group E Sources – Miscellaneous Signals (Cont'd)

Selection Code	Source Signal	Description
11001 (0x19)	PDAP_STRB_O	Select PDAP Strobe Output as the source
11010 (0x1A)	PCG_CLKA_O	Select Precision Clock A Output as the source
11011 (0x1B)	PCG_FSA_O	Select Precision Frame Sync A Output as the source
11100 (0x1C)	PCG_CLKB_O	Select Precision Clock B Output as the source
11101 (0x1D)	PCG_FSB_O	Select Precision Frame Sync B Output as the source
11110 (0x1E)	MISC_LOW_MISC_O	Select Logic Level Low (0) as the source
11111 (0x1F)	MISC_HIGH_MISC_O	Select Logic Level High (1) as the source

Setting the SRU_EXT_MISCA[30] bit to HIGH inverts the level of the EXT_MSCA_4 bit. Setting the SRU_EXT_MISCA[31] bit to HIGH inverts the level of the EXT_MISCA_5 bit.

DAI Pin Buffer Enable Registers (Group F)

The Pin Enable Control registers activate the drive buffer for each of the 20 DAI pins. When the pins are not enabled (driven), they can be used as inputs. Each of the pin enables are connected, based on the 6-bit values in Table A-20. Sixty-four possible signal sources can be designated for these read/write registers:

- SRU_PBEN0, described in Figure A-40
- SRU_PBEN1, described in Figure A-41
- SRU_PBEN2, described in Figure A-42
- SRU_PBEN3, described in Figure A-43

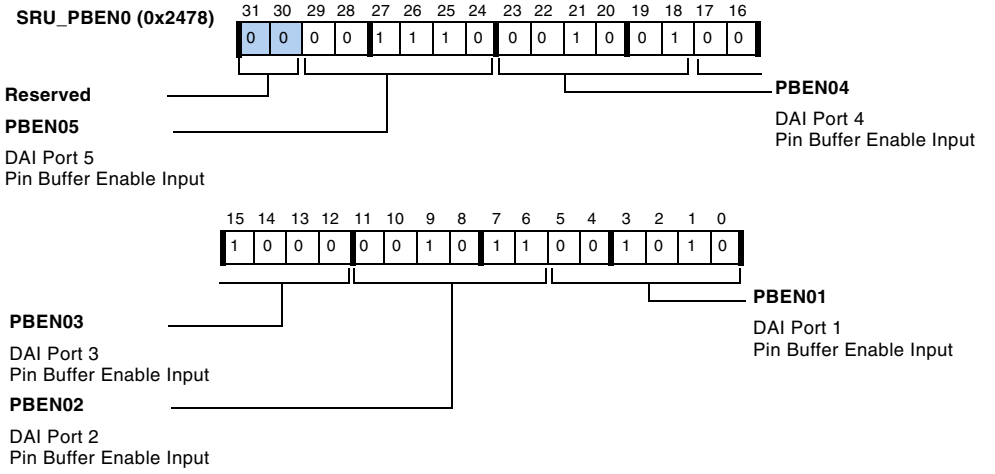


Figure A-40. SRU_PBEN0

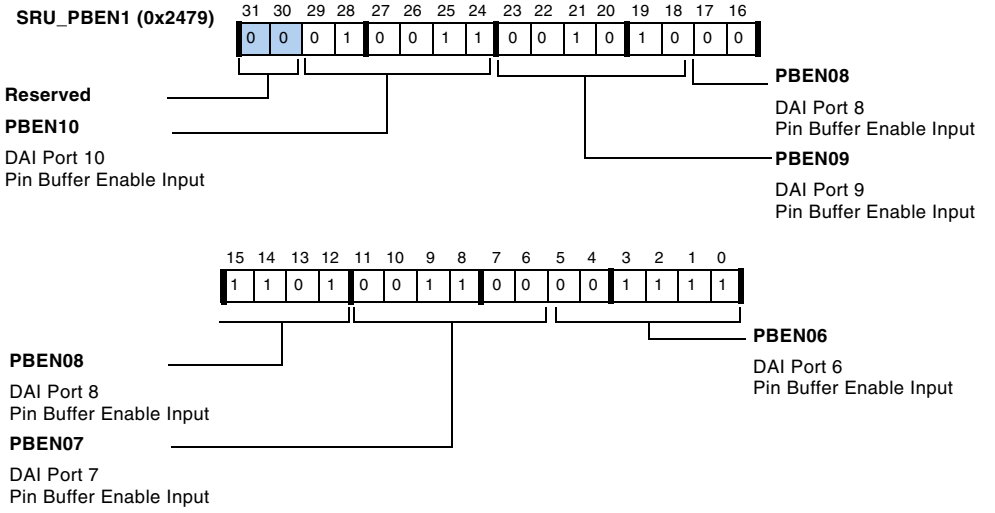


Figure A-41. SRU_PBEN1

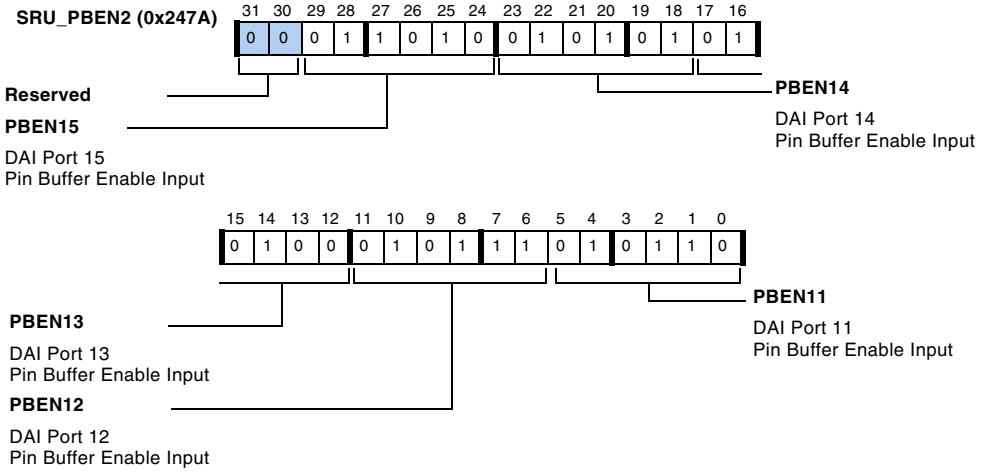


Figure A-42. SRU_PBEN2

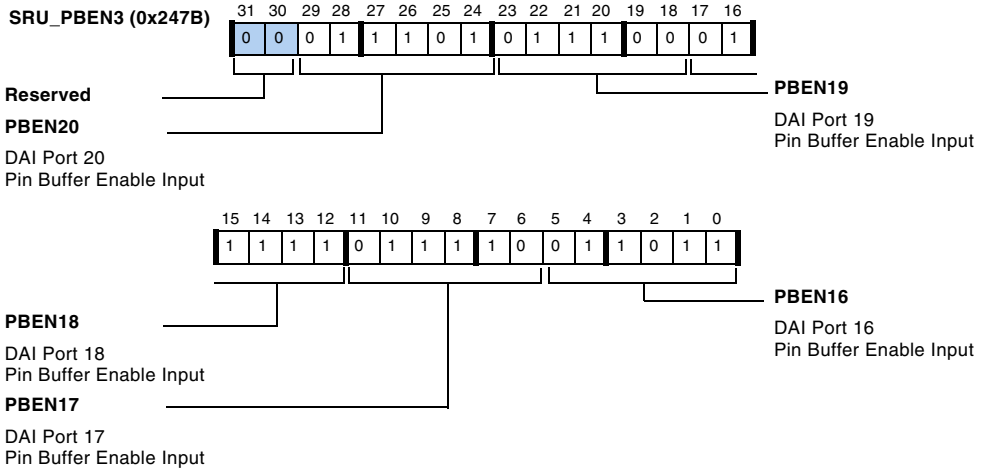


Figure A-43. SRU_PBEN3

Table A-20. Group F Sources – Pin Output Enable

Selection Code	Source Signal	Description
000000 (0x0)	LOW	Select Logic Level Low (0) as the source
000001 (0x1)	HIGH	Select Logic Level High (1) as the source
000010 (0x2)	MISCA0_O	Assign Miscellaneous Control A0 Output to a pin
000011 (0x3)	MISCA1_O	Assign Miscellaneous Control A1 Output to a pin
000100 (0x4)	MISCA2_O	Assign Miscellaneous Control A2 Output to a pin
000101 (0x5)	MISCA3_O	Assign Miscellaneous Control A3 Output to a pin
000110 (0x6)	MISCA4_O	Assign Miscellaneous Control A4 Output to a pin
000111 (0x7)	MISCA5_O	Assign Miscellaneous Control A5 Output to a pin
001000 (0x8)	SPORT0_CLK_PBEN_O	Select Serial Port 0 Clock Output Enable as the source
001001 (0x9)	SPORT0_FS_PBEN_O	Select Serial Port 0 Frame Sync Output Enable as the source
001010 (0xA)	SPORT0_DA_PBEN_O	Select Serial Port 0 Data Channel A Output Enable as the source
001011 (0xB)	SPORT0_DB_PBEN_O	Select Serial Port 0 Data Channel B Output Enable as the source
001100 (0xC)	SPORT1_CLK_PBEN_O	Select Serial Port 1 Clock Output Enable as the source
001101 (0xD)	SPORT1_FS_PBEN_O	Select Serial Port 1 Frame Sync Output Enable as the source
001110 (0xE)	SPORT1_DA_PBEN_O	Select Serial Port 1 Data Channel A Output Enable as the source
001111 (0xF)	SPORT1_DB_PBEN_O	Select Serial Port 1 Data Channel B Output Enable as the source
010000 (0x10)	SPORT2_CLK_PBEN_O	Select Serial Port 2 Clock Output Enable as the source
010001 (0x11)	SPORT2_FS_PBEN_O	Select Serial Port 2 Frame Sync Output Enable as the source

Table A-20. Group F Sources – Pin Output Enable (Cont'd)

Selection Code	Source Signal	Description
010010 (0x12)	SPORT2_DA_PBEN_O	Select Serial Port 2 Data Channel A Output Enable as the source
010011 (0x13)	SPORT2_DB_PBEN_O	Select Serial Port 2 Data Channel B Output Enable as the source
010100 (0x14)	SPORT3_CLK_PBEN_O	Select Serial Port 3 Clock Output Enable as the source
010101 (0x15)	SPORT3_FS_PBEN_O	Select Serial Port 3 Frame Sync Output Enable as the source
010110 (0x16)	SPORT3_DA_PBEN_O	Select Serial Port 3 Data Channel A Output Enable as the source
010111 (0x17)	SPORT3_DB_PBEN_O	Select Serial Port 3 Data Channel B Output Enable as the source
011000 (0x18)	SPORT4_CLK_PBEN_O	Select Serial Port 4 Clock Output Enable as the source
011001 (0x19)	SPORT4_FS_PBEN_O	Select Serial Port 4 Frame Sync Output Enable as the source
011010 (0x1A)	SPORT4_DA_PBEN_O	Select Serial Port 4 Data Channel A Output Enable as the source
011011 (0x1B)	SPORT4_DB_PBEN_O	Select Serial Port 4 Data Channel B Output Enable as the source
011100 (0x1C)	SPORT5_CLK_PBEN_O	Select Serial Port 5 Clock as the Output Enable source
011101 (0x1D)	SPORT5_FS_PBEN_O	Select Serial Port 5 Frame Sync Output Enable as the source
011110 (0x1E)	SPORT5_DA_PBEN_O	Select Serial Port 5 Data Channel A Output Enable as the source
011111 (0x1F)	SPORT5_DB_PBEN_O	Select Serial Port 5 Data Channel B Output Enable as the source
100000 (0x20)	TIMER0_PBEN_O	Select Timer 0 Output Enable as the source
100001 (0x21)	TIMER1_PBEN_O	Select Timer 1 Output Enable as the source

Table A-20. Group F Sources – Pin Output Enable (Cont'd)

Selection Code	Source Signal	Description
100010 (0x22)	TIMER2_PBEN_O	Select Timer 2 Output Enable as the source
100011 (0x23)	FLAG10_PBEN_O	Select Flag 10 Output Enable as the source
100100 (0x24)	FLAG11_PBEN_O	Select Flag 11 Output Enable as the source
100101 (0x25)	FLAG12_PBEN_O	Select Flag 12 Output Enable as the source
100110 (0x26)	FLAG13_PBEN_O	Select Flag 13 Output Enable as the source
100111 (0x27)	FLAG14_PBEN_O	Select Flag 14 Output Enable as the source
101000 (0x28)	FLAG15_PBEN_O	Select Flag 15 Output Enable as the source
101001 (0x29)	Reserved	

Precision Clock Generator Registers

The Precision Clock Generator (PCG) consists of two identical units. Each of these two units (A and B) generates one clock signal (`CLKA_O` or `CLKB_O`) and one frame sync (`FSA_O` or `FSB_O`) output. The PCGs are controlled by the following five memory-mapped registers in the DAI:

- `PCG_CTLA_O`, described in Figure A-44
- `PCG_CTLA_1`, described in Figure A-45
- `PCG_CTLB_O`, described in Figure A-46
- `PCG_CTLB_1`, described in Figure A-47
- `PCG_PW`, described in Figure A-48

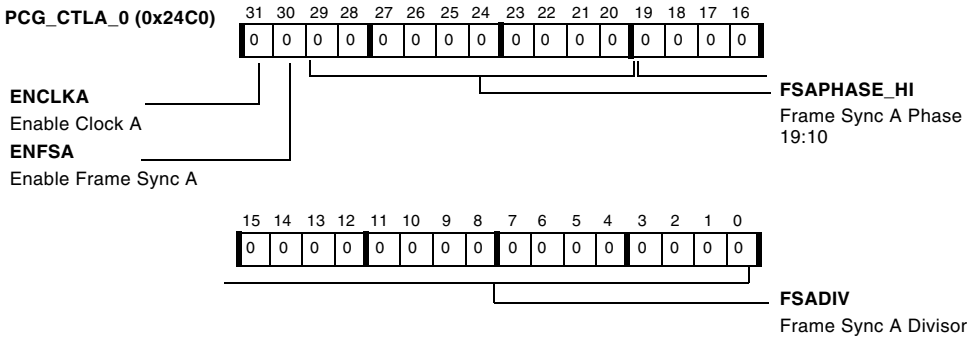


Figure A-44. PCG_CTLA_0 Register

Table A-21. PCG_CTLA_0 Register Bit Descriptions

Bits	Name	Description
19–0	FSADIV	**Divisor for Frame Sync A.**
29–20	FSAPHASE_HI	**Phase for Frame Sync A.** Represents the upper half of the 20-bit value for the channel A frame sync phase. The phase represents the number of input clocks remaining in the first frame after the signal is enabled. See FSAPHASE_LO (Bits 29-20) in PCG_CTLA_1 described on on page A-90.
30	ENFSA	**Enable Frame Sync A.** 0 = Frame Sync A generation disabled 1 = Frame Sync A generation enabled
31	ENCLKA	**Enable Clock A.** 0 = Clock A generation disabled 1 = Clock A generation enabled

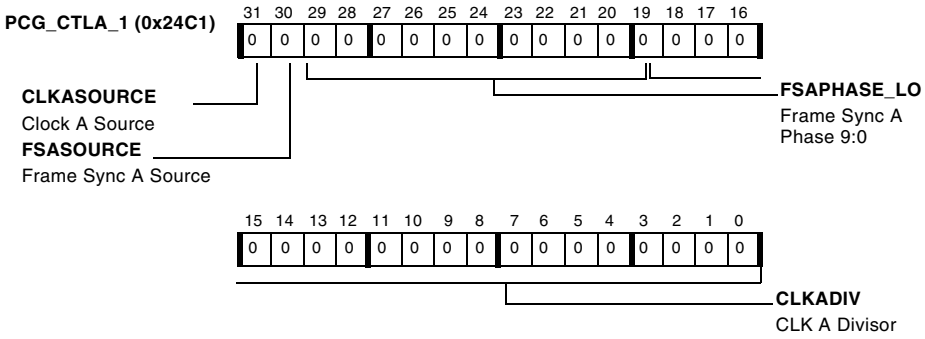


Figure A-45. PCG_CTLA_1 Register

Table A-22. PCG_CTLA_1 Register Bit Descriptions

Bits	Name	Description
19–0	CLKADIV	**Divisor for Clock A.**
29–20	FSAPHASE_LO	**Phase for Frame Sync A.** Note: This field represents the lower half of the 20-bit value for the channel A frame sync phase. The phase represents the number of input clocks remaining in the first frame after the signal is enabled. See also FSAPHASE_HI (Bits 29-20) in PCG_CTLA_0 shown in on page A-89.
30	FSASOURCE	**Frame Sync A Source Master Clock Source for Frame Sync A.** 0 = XTAL buffer output selected for Frame Sync A 1 = PCG_EXTA_I selected for Frame Sync A
31	CLKASOURCE	**Clock A Source Master Clock Source for Clock A.** 0 = XTAL buffer output selected for Clock A 1 = PCG_EXTA_I selected for Clock A

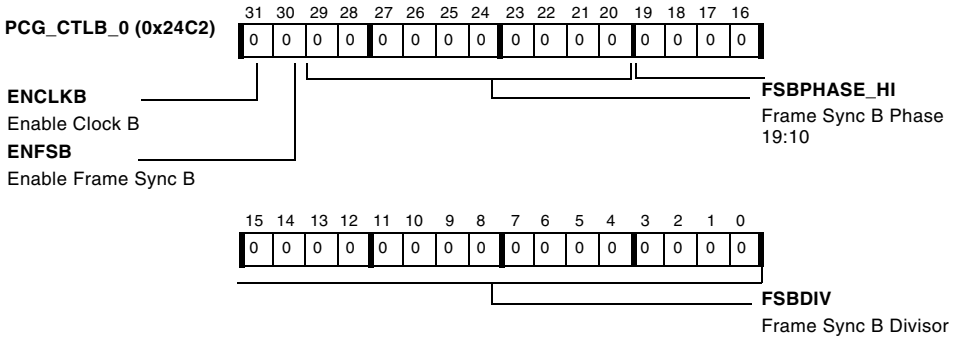


Figure A-46. PCG_CTLB_0 Register

Table A-23. PCG_CTLB_0 Register Bit Descriptions

Bits	Name	Description
19–0	FSBDIV	**Divisor for Frame Sync B.**
29–20	FSBPHASE_HI	**Phase for Frame Sync B.** This field represents the upper half of the 20-bit value for the channel B frame sync phase. The phase represents the number of input clocks remaining in the first frame after the signal is enabled. See also FSBPHASE_LO (Bits 29-20) in PCG_CTLB_1 shown in Figure A-47 on page A-92.
30	ENFSB	**Enable Frame Sync B.** 0 = Frame Sync B generation disabled 1 = Frame Sync B generation enabled
31	ENCLKB	**Enable Clock B.** 0 = Clock B generation disabled 1 = Clock B generation enabled

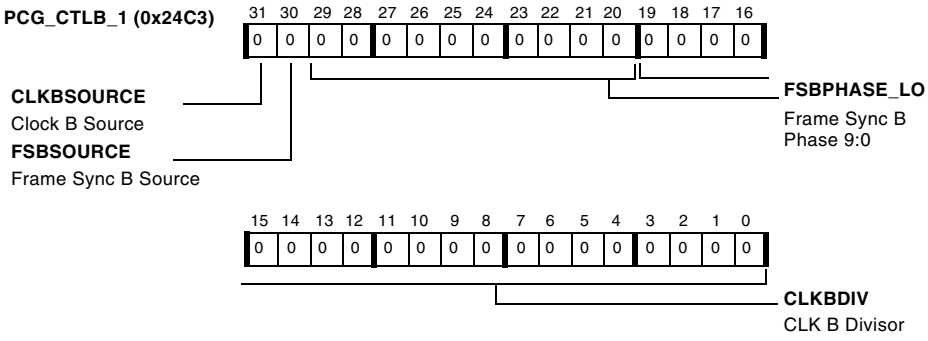


Figure A-47. PCG_CTLB_1 Register

Table A-24. PCG_CTLB_1 Register Bit Descriptions

Bits	Name	Description
19–0	CLKBDIV	**Divisor for Clock B.**
29–20	FSBPHASE_LO	**Phase for Frame Sync B.** Note: This field represents the lower half of the 20-bit value for the channel B frame sync phase. The phase represents the number of input clocks remaining in the first frame after the signal is enabled. See also FSBPHASE_HI (Bits 29-20) in PCG_CTLB_0 shown in Figure A-46 on page A-91.
30	FSBSOURCE	**Frame Sync B Source.** Master Clock Source for Frame Sync B. 0 = XTAL buffer output selected for Frame Sync B 1 = PCG_EXTB_I selected for Frame Sync B
31	CLKBSOURCE	**Clock B Source.** Master Clock Source for Clock B. 0 = XTAL buffer output selected for Clock B 1 = PCG_EXTB_I selected for Clock B

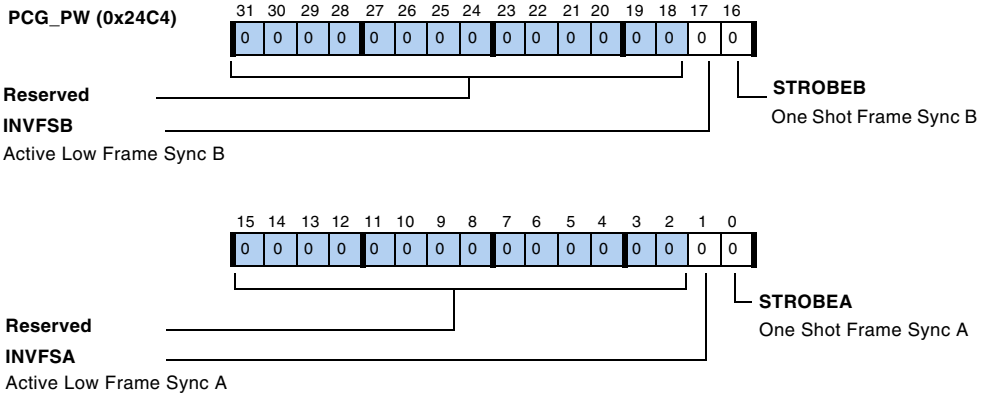


Figure A-48. PCG_PW Register (Bypass Mode)

Table A-25. PCG_PW Register (Bypass Mode)

Bits	Name	Description
0	STROBEA	**One Shot Frame Sync A.** Frame sync is a pulse with duration equal to one period of MISCA2_I signal repeating at the beginning of every frame. Note: This is valid in bypass mode only.
1	INVFSA	**Active Low Frame Sync Select for Frame Sync A.** Selects an active low FS if set, (= 1) or active high FS if cleared, (= 0).
15–2	Reserved[1]	
16	STROBEB	**One Shot Frame Sync B.** Frame Sync is a pulse with duration equal to one period of MISCA3_I signal repeating at the beginning of every frame. Note: This is valid in bypass mode only.
17	INVFSB	**Active Low Frame Sync Select.** Selects an active low FS if set, (= 1) or active high FS if cleared, (= 0).
31–18	Reserved[1]	

1 In bypass mode, Bits 15-2 and Bits 31-18 are ignored.

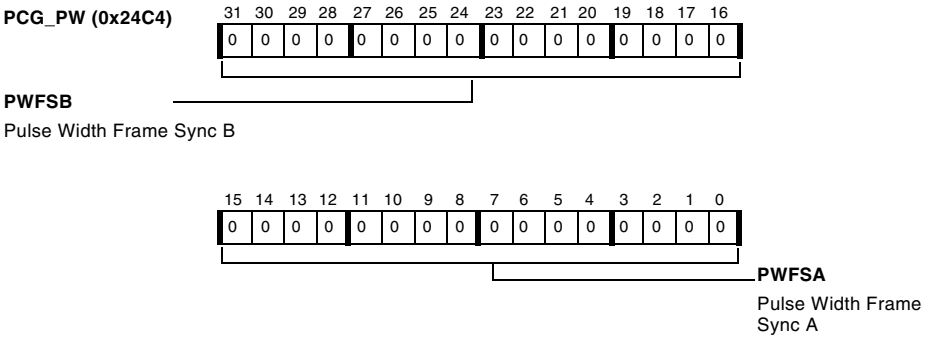


Figure A-49. PCG_PW Register (Normal Mode)

Table A-26. PCG_PW Register (Normal Mode)

Number of Bits	Name	Description
15–0	PWFSA	**Pulse Width for Frame Sync A.** These bits are valid when not in Bypass mode.
31–16	PWFSB	**Pulse Width for Frame Sync B.** These bits are valid when not in Bypass mode.

Input Data Port Registers

The Input Data Port (IDP) provides an additional input path to the processor core, configurable 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. Seven registers are used to specify modes, track status of inputs and outputs, permit the IDP FIFO buffer to be read, and so on.

- IDP_CTL, described in Figure A-50
- DAI_STAT, described in Figure A-56
- IDP_FIFO, described on Figure A-51
- IDP_DMA_Ix (including IDP_DMA_I0, IDP_DMA_I1, IDP_DMA_I2, IDP_DMA_I3, IDP_DMA_I4, IDP_DMA_I5, IDP_DMA_I6, and IDP_DMA_I7) described beginning with Figure A-52
- IDP_DMA_Mx (including IDP_DMA_M0, IDP_DMA_M1, IDP_DMA_M2, IDP_DMA_M3, IDP_DMA_M4, IDP_DMA_M5, IDP_DMA_M6, and IDP_DMA_M7), described beginning with Figure A-53
- IDP_DMA_Cx (including IDP_DMA_C0, IDP_DMA_C1, IDP_DMA_C2, IDP_DMA_C3, IDP_DMA_C4, IDP_DMA_C5, IDP_DMA_C6, and IDP_DMA_C7), described beginning with Figure A-54
- IDP_PDAP_CTL, described in Figure A-55

Input Data Port Control Registers (IDP_CTL)

The IDP_CTL[31:8] bits control the input format modes for each of the eight channels.

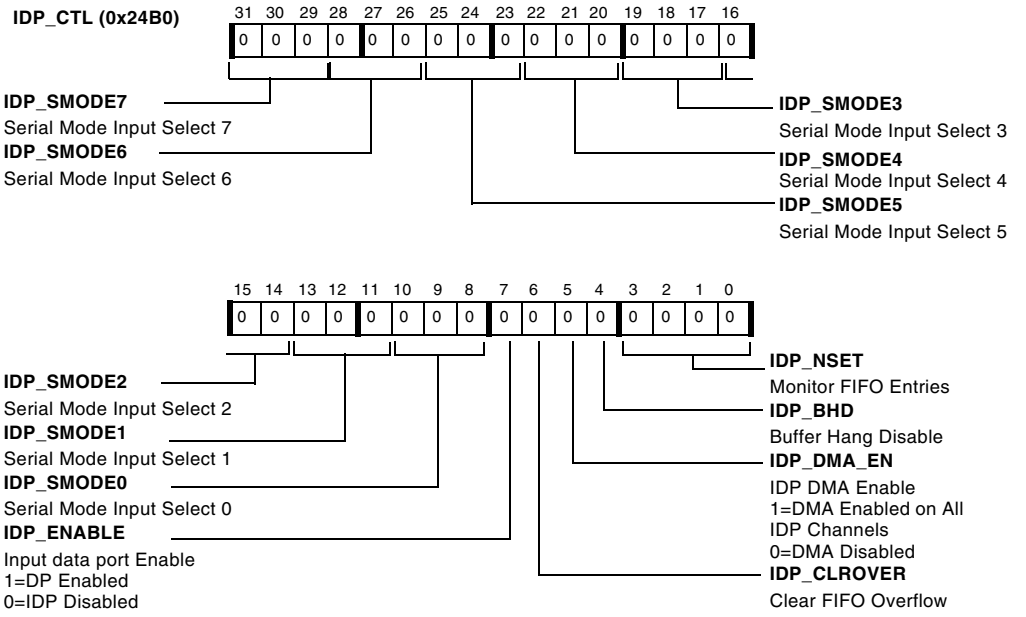


Figure A-50. IDP_CTL Register

Table A-27. IDP_CTL Register

Bits	Name	Description
3–0	IDP_NSET	Monitored number of FIFO entries where N > samples raises Interrupt Controller bit 8.
4	IDP_BHD	**IDP Buffer Hang Disable.** Reads of an empty FIFO or writes to a full FIFO make the 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). 1 = Core hang is disabled 0 = Core hang is enabled
5	IDP_DMA_EN	**DMA Enable.** Enables DMA on all IDP channels.

Table A-27. IDP_CTL Register (Cont'd)

Bits	Name	Description
6	IDP_CLROVR	**FIFO Overflow Clear Bit.** Writes of 1 to this bit will clear the overflow condition in the DAI_STAT register. Because this is a write-only bit; it always returns LOW when read.
7	IDP_ENABLE	**Enable IDP.** 1 to 0 transition on this bit clears the IDP_FIFO. 1 = IDP is enabled 0 = IDP is disabled and data does not come to IDP_FIFO from IDP channels
10–8	IDP_SMODE0	**Serial Input Mode Select.** These eight inputs (0-7), each of which is 3-bits, indicate the mode of the serial input for each of the eight IDP channels. Input format: 000 = Left-justified Sample Pair mode 001 = I^2S mode 010 = RESERVED 011 = RESERVED 100 = Right-justified 24-bits 101 = Right-justified 20-bits 110 = Right-justified 18-bits 111 = Right-justified 16-bits
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 FIFO Register (IDP_FIFO)

The `IDP_FIFO` register provides information about the output of the IDP FIFO. Normally, the `IDP_FIFO` register is used only to read and remove the top sample from the FIFO. However, the core may also write to this register. When it does so, 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 FIFO is an eight-deep FIFO.

Channel encoding provides for eight combinations, corresponding to the eight inputs. When using Channels 1–7, this register format applies, as well as when using Channel 0 in Serial mode. When using Channel 0 in Parallel mode, refer to the descriptions of the four possible packing modes. For more information, see “Packing Unit” on page 6-8.

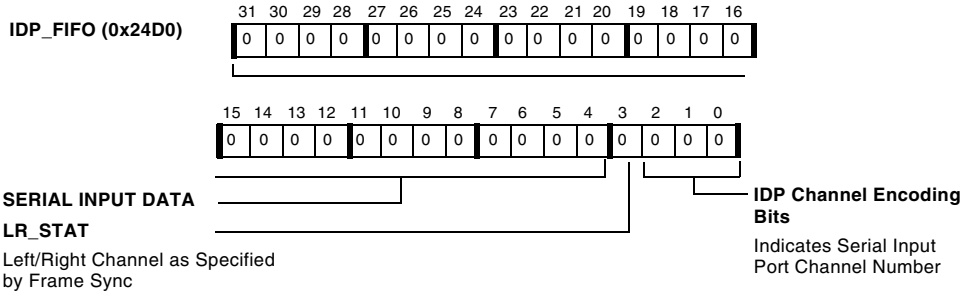


Figure A-51. IDP_FIFO Register

Table A-28. IDP_FIFO Register Bit Descriptions

Bits	Name	Description
2–0		**IDP Channel Encoding Bits.** Indicate serial input port channel number that gave this serial input data. Note: This information is not valid when data comes from PDAP channel.
3	LR_STAT	**Left/Right Channel Status.** Indicate 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 IDP_CTL description in Table A-27 on page A-96.
31–4		**Input Data (Serial).** Some LSBs can be zero, depending on the mode.

The information in this table is not valid for the case where data comes from the PDAP channel.

Input Data Port DMA Control Registers

Each of the eight DMA channels have an I register with an Index pointer (19 bits), an M register with an M modifier/stride (6 bits), and a C register with a count (16 bits). For example, `IDP_DMA_I0`, `IDP_DMA_M0` and `IDP_DMA_C0` control the DMA for Channel 0. These registers are:

- `IDP_DMA_Ix` (Index) registers
- `IDP_DMA_Mx` (Modifier) registers
- `IDP_DMA_Cx` (Count) registers

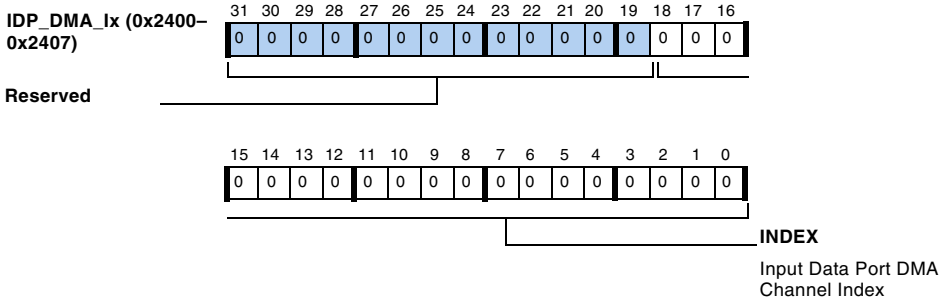


Figure A-52. IDP_DMA_Ix Register

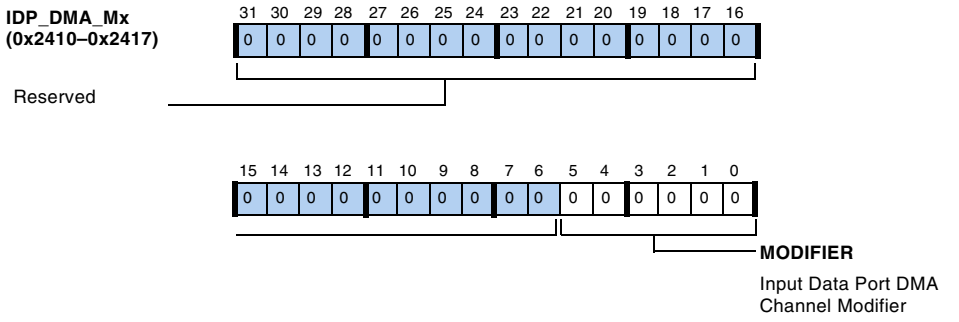


Figure A-53. IDP_DMA_Mx Register

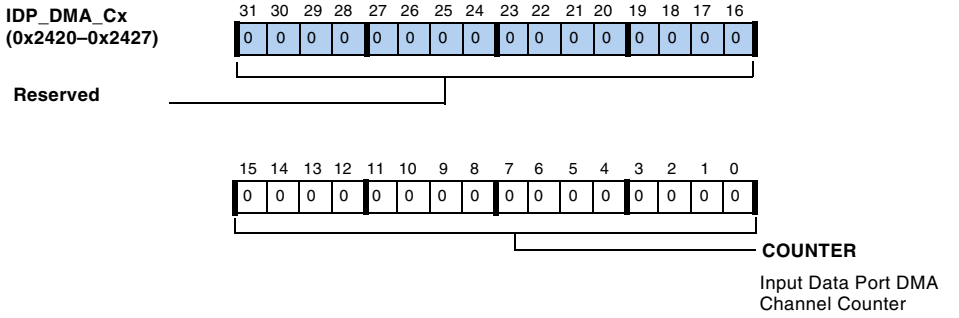


Figure A-54. IDP_DMA_Cx Register

Parallel Data Acquisition Port Control Register (IDP_PDAP_CTL)

Setting the `IDP_PDAP_CTL[31]` bit enables either the 20 DAI pins or the 16 parallel port address/data pins to be used as a parallel input channel. These parallel words may be packed into 32-bit words for efficiency. 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.

The IDP_PDAP_CTL register provides 20 mask bits that allow the input from any of the 20 pins to be ignored. When the mask bit is cleared, the corresponding bit is cleared in the acquired data word. This register also provides a reset bit that zeros any data waiting in the packing unit to be latched into the FIFO. The RESET bit (bit 30) causes the reset circuit to strobe when asserted, and then automatically clears. Therefore, this bit always returns a value of zero when read. Bit 26 of the IDP_PDAP_CTL register selects between the two sets of pins that may be used as the parallel input port. When the IDP_PDAP_CTL[26] bit is set, the upper 16 bits are read from the AD[15:0] pins. When the IDP_PDAP_CTL[26] bit is cleared, the upper 16 bits are read from the DAI_P[20:5] pins. Note that the four LSB's of the parallel data acquisition port input are not multiplexed, and this input value is always read from the DAI pins, DAI_P[4:1].

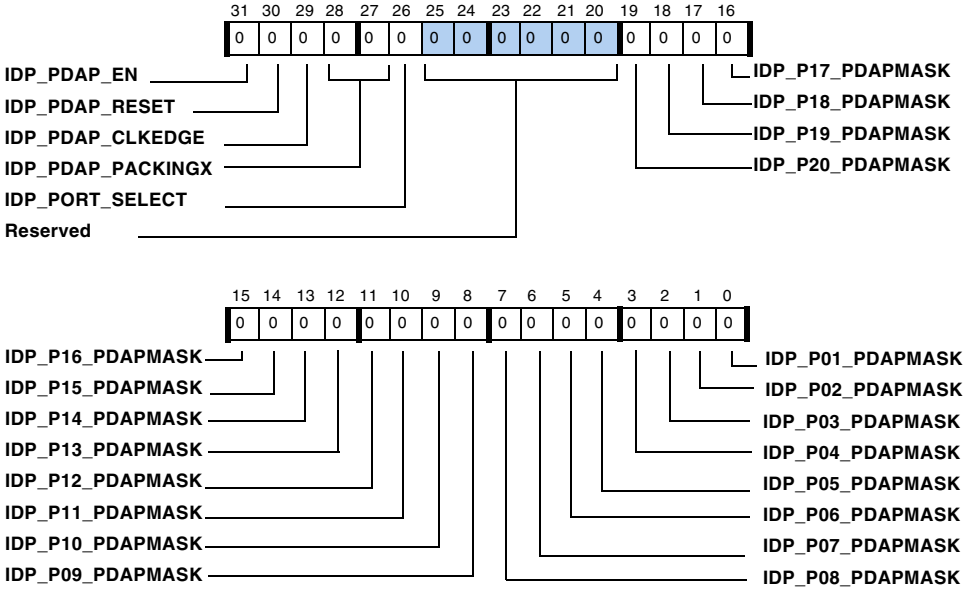


Figure A-55. IDP_PDAP_CTL Register

Table A-29. IDP_PDAP_CTL Register

Bits	Name	Description
0	IDP_P01_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_01 is masked 1 = Input data from DAI_01 is un-masked
1	IDP_P02_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_02 is masked 1 = Input data from DAI_02 is un-masked
2	IDP_P03_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_03 is masked 1 = Input data from DAI_03 is un-masked
3	IDP_P04_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_04 is masked 1 = Input data from DAI_04 is un-masked
4	IDP_P05_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_05/DATA0 is masked 1 = Input data from DAI_05/DATA0 is un-masked
5	IDP_P06_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_06/DATA1 is masked 1 = Input data from DAI_06/DATA1 is un-masked
6	IDP_P07_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_07/DATA2 is masked 1 = Input data from DAI_07/DATA2 is un-masked
7	IDP_P08_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_08/DATA3 is masked 1 = Input data from DAI_08/DATA3 is un-masked
8	IDP_P09_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_09/DATA4 is masked 1 = Input data from DAI_09/DATA4 is un-masked
9	IDP_P10_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_10/DATA5 is masked 1 = Input data from DAI_10/DATA5 is un-masked
10	IDP_P11_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_11/DATA6 is masked 1 = Input data from DAI_11/DATA6 is un-masked

Table A-29. IDP_PDAP_CTL Register (Cont'd)

Bits	Name	Description
11	IDP_P12_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_12/DATA7 is masked 1 = Input data from DAI_12/DATA7 is un-masked
12	IDP_P13_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_13/ADDR0 is masked 1 = Input data from DAI_13/ADDR0 is un-masked
13	IDP_P14_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_14/ADDR1 is masked 1 = Input data from DAI_14/ADDR1 is un-masked
14	IDP_P15_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_15/ADDR2 is masked 1 = Input data from DAI_15/ADDR2 is un-masked
15	IDP_P16_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_16/ADDR3 is masked 1 = Input data from DAI_16/ADDR3 is un-masked
16	IDP_P17_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_17/ADDR4 is masked 1 = Input data from DAI_17/ADDR4 is un-masked
17	IDP_P18_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_18/ADDR5 is masked 1 = Input data from DAI_18/ADDR5 is un-masked
18	IDP_P19_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_19/ADDR6 is masked 1 = Input data from DAI_19/ADDR6 is un-masked
19	IDP_P20_PDAPMASK	**Parallel Data Acquisition Port Mask** 0 = Input data from DAI_20/ADDR7 is masked 1 = Input data from DAI_20/ADDR7 is un-masked
25–20	Reserved	
26	IDP_PORT_SELECT	**Port Select: Input Pins Select** 1 = Selects upper 16 inputs from AD[15:0] (ADDR7–ADDR0, DATA7–DATA0) 0 = Selects upper 16 inputs from DAI_P[20:5]

Table A-29. IDP_PDAP_CTL Register (Cont'd)

Bits	Name	Description
28–27	IDP_PDAP_PACKING	**PACKING** Selects PDAP packing mode 00 = 8 to 32 packing 01 = {11,11,10} to 32 packing 10 = 16 to 32 packing 11 = 20 to 32 packing
29	IDP_PDAP_CLKEDGE	**PDAP (Rising or Falling) Clock Edge** Setting this bit (= 1) causes the data to be latched on the falling edge. Clearing this bit causes data to be latched on the rising edge of the clock (IDP0_CLK_I).
30	IDP_PDAP_RESET	**PDAP Reset** Setting this bit (= 1) causes the PDAP reset circuit to strobe; then this bit is cleared automatically. This bit will always return a value of zero when read.
31	IDP_PDAP_EN	**PDAP Enable** Setting this bit (= 1) enables either the 20 DAI pins or the 16 parallel port address/data pins to be used as a parallel input channel. Clearing this bit (= 0) disables those pins from use as parallel input channels. Note: When this bit is set to 1, then IDP Channel 0 cannot be used as a serial input port.

Digital Audio Interface Status Register (DAI_STAT)

The DAI_STAT register is a read-only register located at address 0x24B8. The state of all eight DMA channels is reflected in the IDP_DMAx_STAT bits (bits 24–17) of the DAI_STAT register. These bits are set once the IDP_DMA_EN bit is set and they remain set until the last data in that channel is transferred. Even if the IDP_DMA_EN bit is set, it goes low once the required number of data transfers occurs. Even if the DMA through some channel is not intended, its IDP_DMAx_STAT goes high.

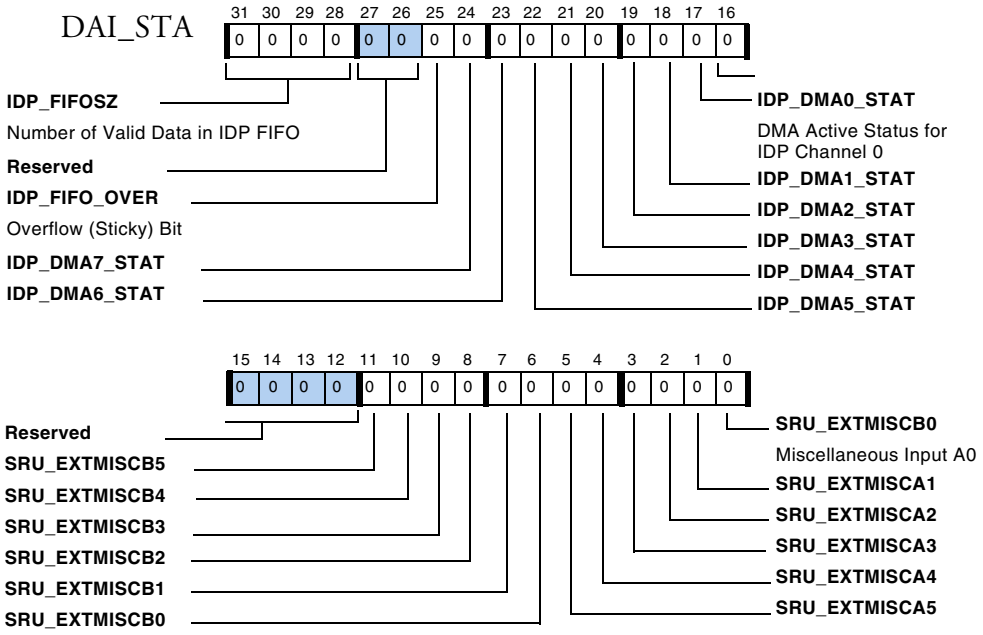


Figure A-56. DAI_STAT Register

Table A-30. DAI_STAT Register Bit Descriptions

Bits	Name	Description
11–0	SRU_EXTMISCyx	**Miscellaneous Input A/B Signals.** Indicate the status of the MISCxy_I signals.
16–12	Reserved	
24–17	IDP_DMAx_STAT	**Input Data Port DMA Status.** 1 = DMA is active 0 = DMA is not active
25	IDP_FIFO_OVER	**IDP_FIFO Overflow Status.** This (sticky) bit provides IDP FIFO overflow status information. 1 = Overflow has occurred 0 = No overflow

Table A-30. DAI_STAT Register Bit Descriptions (Cont'd)

Bits	Name	Description
27–26	Reserved	
31–28	IDP_FIFOSZ	Number of samples in FIFO

DAI Resistor Pull-up Enable Register (DAI_PIN_PULLUP)

This 20-bit read/write register is located at address 0x247D. Bits 19–0 of this register control the enabling/disabling 22.5 KΩ pull-up resistor on the `DAI_P0[19:0]` bits. Setting a bit to one enables a pull-up resistor on the corresponding pin. After `RESET`, the value of this register is `0xFFFFF`, which means pull-up is enabled on all 20 DAI pins.

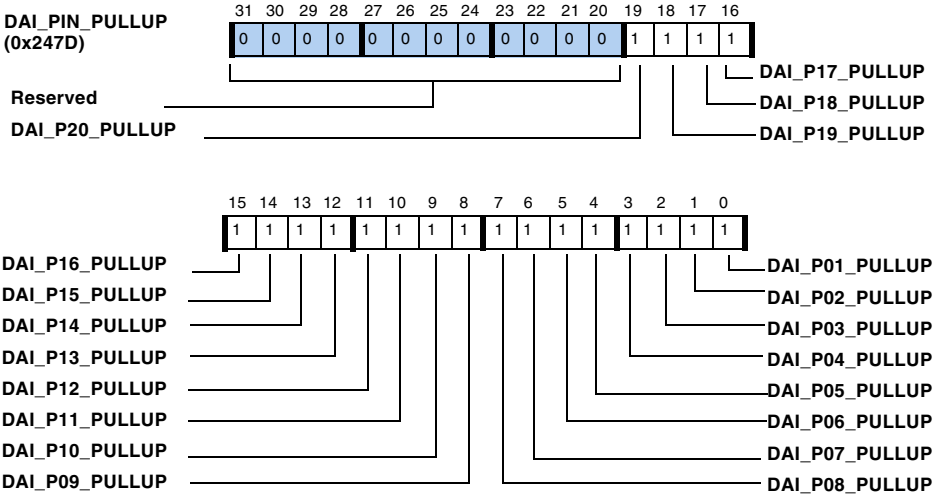


Figure A-57. DAI_PIN_PULLUP Register

Table A-31. DAI_PIN_PULLUP Register

Bits	Name	Description
0	DAI_P01_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P01** 1 = enables pull-up on DAI_P01 0 = disables pull-up on DAI_P01
1	DAI_P02_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P02** 1 = enables pull-up on DAI_P02 0 = disables pull-up on DAI_P02
2	DAI_P03_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P03** 1 = enables pull-up on DAI_P03 0 = disables pull-up on DAI_P03
3	DAI_P04_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P04** 1 = enables pull-up on DAI_P04 0 = disables pull-up on DAI_P04
4	DAI_P05_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P05** 1 = enables pull-up on DAI_P05 0 = disables pull-up on DAI_P05
5	DAI_P06_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P06** 1 = enables pull-up on DAI_P06 0 = disables pull-up on DAI_P06
6	DAI_P07_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P07** 1 = enables pull-up on DAI_P07 0 = disables pull-up on DAI_P07
7	DAI_P08_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P08** 1 = enables pull-up on DAI_P08 0 = disables pull-up on DAI_P08
8	DAI_P09_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P09** 1 = enables pull-up on DAI_P09 0 = disables pull-up on DAI_P09
9	DAI_P10_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P10** 1 = enables pull-up on DAI_P10 0 = disables pull-up on DAI_P10
10	DAI_P11_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P11** 1 = enables pull-up on DAI_P11 0 = disables pull-up on DAI_P11

Table A-31. DAI_PIN_PULLUP Register (Cont'd)

Bits	Name	Description
11	DAI_P12_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P12** 1 = enables pull-up on DAI_P12 0 = disables pull-up on DAI_P12
12	DAI_P13_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P13** 1 = enables pull-up on DAI_P13 0 = disables pull-up on DAI_P13
13	DAI_P14_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P14** 1 = enables pull-up on DAI_P14 0 = disables pull-up on DAI_P14
14	DAI_P15_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P15** 1 = enables pull-up on DAI_P15 0 = disables pull-up on DAI_P15
15	DAI_P16_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P16** 1 = enables pull-up on DAI_P16 0 = disables pull-up on DAI_P16
16	DAI_P17 _PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P17** 1 = enables pull-up on DAI_P17 0 = disables pull-up on DAI_P17
17	DAI_P18_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P18** 1 = enables pull-up on DAI_P18 0 = disables pull-up on DAI_P18
18	DAI_P19_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P19** 1 = enables pull-up on DAI_P19 0 = disables pull-up on DAI_P19
19	DAI_P20_PULLUP	**Enable/Disable 22.5 KΩ Pull-up Resistor for DAI_P20** 1 = enables pull-up on DAI_P20 0 = disables pull-up on DAI_P20
31–20	Reserved	

DAI Pin Status Register (DAI_PIN_STAT)

This 20-bit read-only register is located at address 0x24B9. Bits 19–0 of this register indicate the status of `DAI_P[20:1]`. Reads from bits 31–20 always return 0. This register is updated at ½ core clock rate.

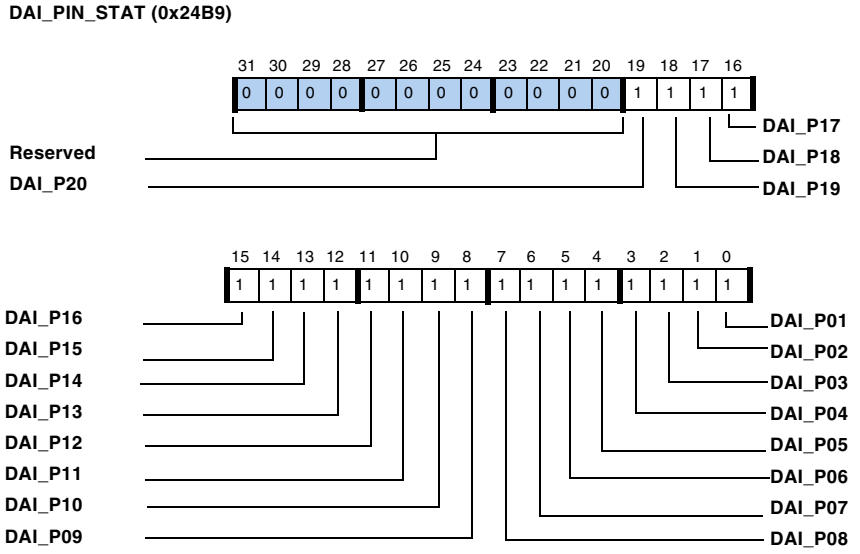


Figure A-58. DAI_PIN_STAT Register

Table A-32. DAI_PIN_STAT Register

Bits	Name	Description
0	DAI_P01	Provides status of DAI_P01 pin
1	DAI_P02	Provides status of DAI_P01 pin
2	DAI_P03	Provides status of DAI_P03 pin
3	DAI_P04	Provides status of DAI_P04 pin
4	DAI_P05	Provides status of DAI_P05 pin
5	DAI_P06	Provides status of DAI_P06 pin

Table A-32. DAI_PIN_STAT Register (Cont'd)

Bits	Name	Description
6	DAI_P07	Provides status of DAI_P07 pin
7	DAI_P08	Provides status of DAI_P08 pin
8	DAI_P09	Provides status of DAI_P09 pin
9	DAI_P10	Provides status of DAI_P10 pin
10	DAI_P11	Provides status of DAI_P11 pin
11	DAI_P12	Provides status of DAI_P12 pin
12	DAI_P13	Provides status of DAI_P13 pin
13	DAI_P14	Provides status of DAI_P14 pin
14	DAI_P15	Provides status of DAI_P15 pin
15	DAI_P16	Provides status of DAI_P16 pin
16	DAI_P17	Provides status of DAI_P17 pin
17	DAI_P18	Provides status of DAI_P18 pin
18	DAI_P19	Provides status of DAI_P19 pin
19	DAI_P20	Provides status of DAI_P20 pin
31–20	Reserved	

DAI Interrupt Controller Registers

The DAI contains its own Interrupt Controller that indicates to the core when DAI audio peripheral related events occur. Since audio events generally occur infrequently relative to the SHARC processor core, the DAI Interrupt Controller reduces all of its interrupts onto two interrupt signals within the core's primary interrupt systems—one mapped with low priority and one mapped with high priority. This architecture allows the user to indicate priority broadly. In this way, the DAI interrupt controller registers provide 32 independently configurable interrupts labeled `DAI_INT[31:0]`, respectively.

The DAI Interrupt Controller is configured using three registers. Each of the 32 interrupt lines can be independently configured to trigger based on the incoming signal's rising edge, falling edge, both or neither. Setting a bit in the DAI_IRPTL_RE or DAI_IRPTL_FE registers enables the interrupt level on the rising and falling edges, respectively.

The 32 interrupt signals within the DAI are mapped to two interrupt signals in the primary Interrupt Controller of the SHARC processor core. The DAI_IRPTL_PRI register selects if the DAI interrupt is mapped to the high priority or low priority core interrupt (1 = high priority, 0 = low priority).

The DAI_IRPTL_H register is a read-only register that has bits set for every DAI interrupt latched for the high priority core interrupt. The DAI_IRPTL_L register is a read-only register that has bits set for every DAI interrupt latched for the low priority core interrupt. When a DAI interrupt occurs, the low or high priority core ISR should query its corresponding register to determine which of the 32 interrupt sources requires service. When the DAI_IRPTL_H register is read, the high priority latched interrupts are cleared. When the DAI_IRPTL_L register is read, the low priority latched interrupts are cleared.

DMA, overflow, and greater than N interrupts can be sensed only at rising edges. Falling edges are not used for these ten interrupts (eight DMA, one overflow, and one FIFO valid data greater than N).

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 of the interrupt goes away.

The following registers are used primarily to provide status of the Resident Interrupt Controller:

- High Priority Interrupt Latch (DAI_IRPTL_H) register, described on page A-112
- Low Priority Interrupt Latch (DAI_IRPTL_L) register, described on page A-113
- Core Interrupt Priority Assignment (DAI_IRPTL_PRI) register, described on page A-114
- Rising Edge Interrupt Mask (DAI_IRPTL_RE) register, described on page A-115
- Falling Edge Interrupt Mask (DAI_IRPTL_FE) register, described on page A-116

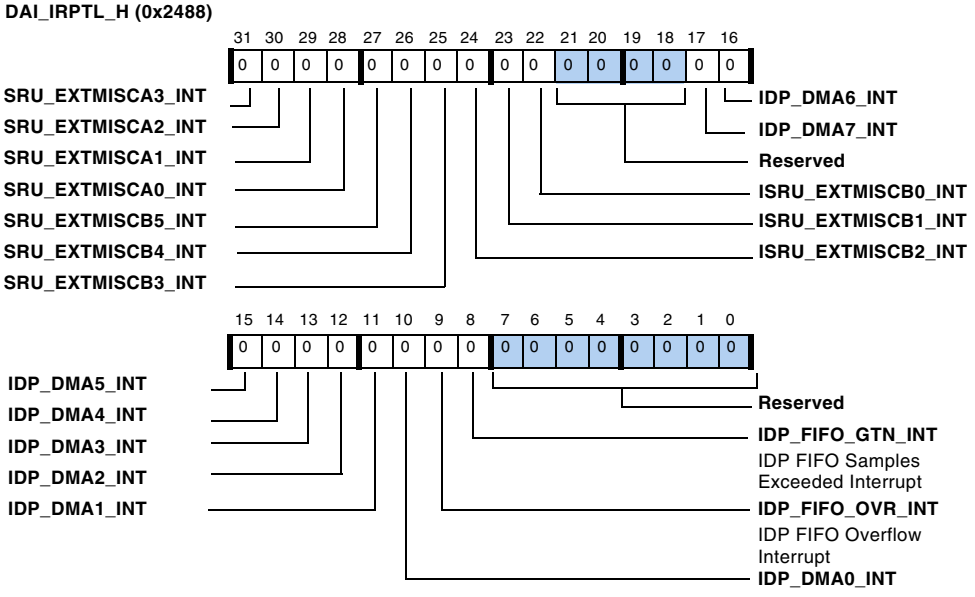


Figure A-59. DAI_IRPTL_H Register

An explicit read resets these register values to zero, except for the IDP_FIFO_GTN_INT (IDP FIFO samples exceeded interrupt) bit. The interrupt on the IDP_FIFO_GTN_INT bit clears automatically when the condition that caused the interrupt goes away.

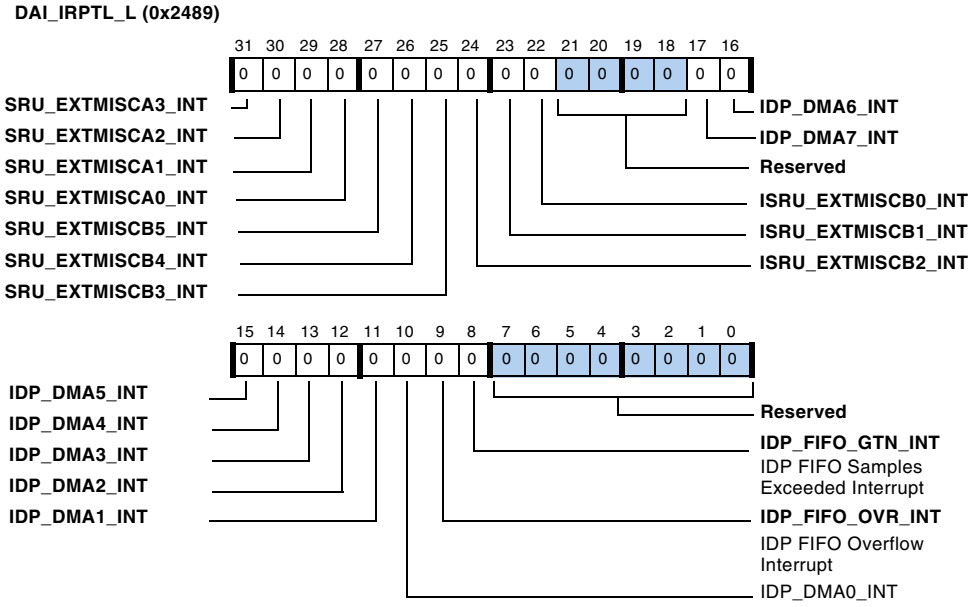


Figure A-60. DAI_IRPTL_L Register

A read resets the value to zero, except under the following condition—the IDP_FIFO_GTN_INT bit is not cleared when DAI_IRPTL_H/L registers are read. This bit is cleared when the cause of this interrupt is zero.

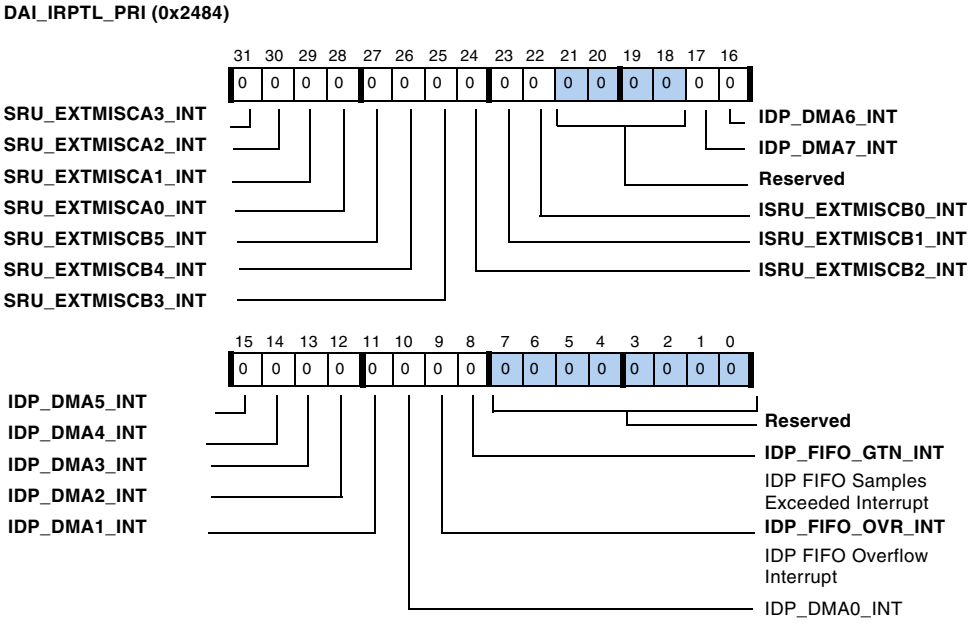


Figure A-61. DAI_IRPTL_PRI Register

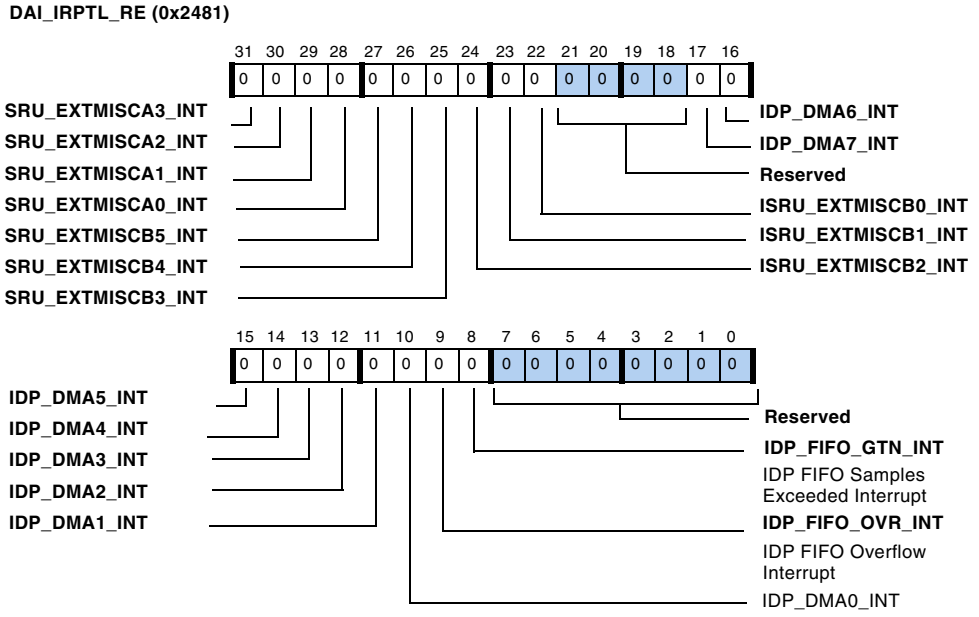


Figure A-62. DAI_IRPTL_RE Register

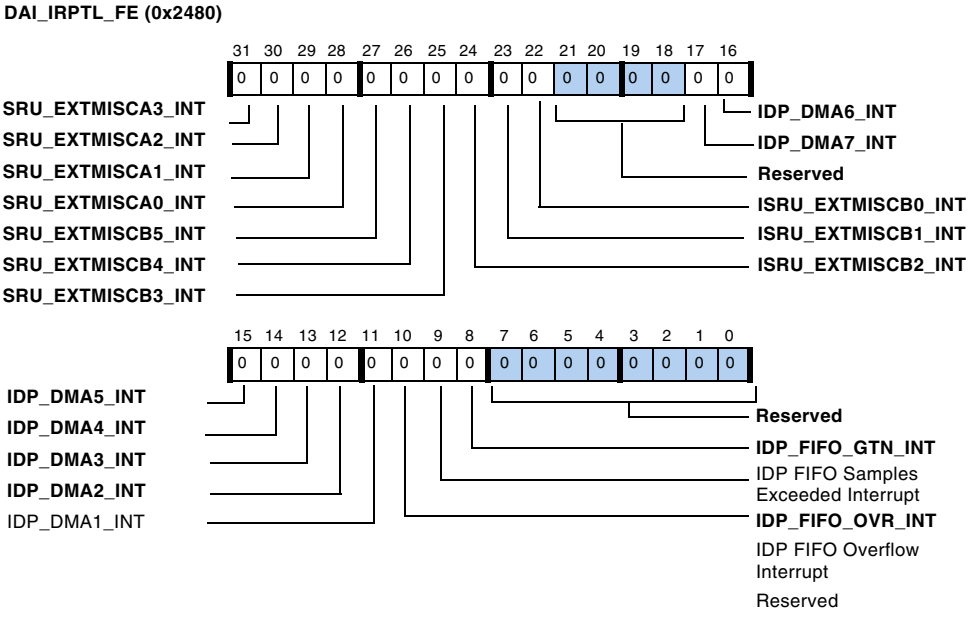


Figure A-63. DAI_IRPTL_FE Register

I INDEX

D

E

F

G

H

J

L

M

R

S

T

U

V

W