ADSP-219x/2192 DSP Hardware Reference

Revision 1.1, April 2004

Part Number
82-002001-01

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

Copyright Information

©2004 Analog Devices, Inc., ALL RIGHTS RESERVED. This document may not be reproduced in any form without prior, express written consent from Analog Devices, Inc.

Printed in the USA.

Disclaimer

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

Trademark and Service Mark Notice

The Analog Devices logo and VisualDSP++ are registered trademarks of Analog Devices, Inc.

EZ-KIT Lite is a trademark of Analog Devices, Inc.

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

CONTENTS

INTRODUCTION

Purpose 1-1
Audience 1-1
Overview—Why Fixed-Point DSP? 1-2
ADSP-219x Design Advantages 1-2
ADSP-219x Architecture Overview 1-5
DSP Core Architecture 1-7
DSP Peripherals Architecture 1-9
Memory Architecture 1-9
Internal (On-Chip) Memory 1-11
Interrupts 1-12
DMA Controller 1-12
PCI Port 1-12
USB Port 1-13
AC'97 Interface 1-13
Low Power Operation 1-13
Clock Signals 1-13
Reset Modes 1-14
JTAG Port 1-15

Development Tools 1-15
Differences from Previous DSPs 1-17
Computational Units and Data Register File 1-17
Shifter Result (SR) Register as Multiplier Dual Accumulator 1-18
Shifter Exponent (SE) Register is not Memory Accessible 1-18
Conditions (SWCOND) and Condition Code (CCODE) Register 1-19
Unified Memory Space 1-20
Data Memory Page (DMPG1 and DMPG2) Registers 1-20
Data Address Generator (DAG) Addressing Modes 1-21
Base Registers for Circular Buffers. 1-21
Program Sequencer, Instruction Pipeline, and Stacks 1-22
Conditional Execution (Difference in Flag Input Support) 1-22
Execution Latencies (Different for JUMP Instructions) 1-23
Instruction Set Enhancements 1-24
For More Information About Analog Products 1-24
For Technical or Customer Support 1-25
What's New in This Manual 1-25
Related Documents 1-25
Conventions 1-27

COMPUTATIONAL UNITS

Overview 2-1
Using Data Formats 2-4
Binary String 2-4
Unsigned 2-4
Signed Numbers: Two's Complement 2-5
Fractional Representation: 1.15 2-5
ALU Data Types 2-5
Multiplier Data Types 2-6
Shifter Data Types 2-7
Arithmetic Formats Summary 2-8
Setting Computational Modes 2-10
Latching ALU Result Overflow Status 2-10
Saturating ALU Results on Overflow 2-11
Using Multiplier Integer and Fractional Formats 2-12
Rounding Multiplier Results 2-14
Unbiased Rounding 2-14
Biased Rounding 2-15
Using Computational Status 2-16
Arithmetic Logic Unit (ALU) 2-17
ALU Operation 2-17
ALU Status Flags 2-18
ALU Instruction Summary 2-19

ALU Data Flow Details 2-21
ALU Division Support Features 2-23
Multiply—Accumulator (Multiplier) 2-28
Multiplier Operation 2-28
Placing Multiplier Results in MR or SR Registers 2-29
Clearing, Rounding, or Saturating Multiplier Results 2-30
Multiplier Status Flags 2-31
Saturating Multiplier Results on Overflow 2-31
Multiplier Instruction Summary 2-33
Multiplier Data Flow Details 2-34
Barrel-Shifter (Shifter) 2-37
Shifter Operations 2-37
Derive Block Exponent 2-39
Immediate Shifts 2-40
Denormalize 2-42
Normalize, Single Precision Input 2-44
Normalize, ALU Result Overflow 2-45
Normalize, Double Precision Input 2-47
Shifter Status Flags 2-50
Shifter Instruction Summary 2-50
Shifter Data Flow Details 2-52
Data Register File 2-57
Secondary (Alternate) Data Registers 2-59
Multifunction Computations 2-60

PROGRAM SEQUENCER

Overview 3-1

Instruction Pipeline 3-7

Instruction Cache 3-9

Using The Cache 3-11

Optimizing Cache Usage 3-12

Branches and Sequencing 3-14

Indirect Jump Page (IJPG) Register 3-15

Conditional Branches 3-16

Delayed Branches 3-16

Loops and Sequencing 3-20

Managing Loop Stacks 3-24

Restrictions On Ending Loops 3-24

Interrupts and Sequencing 3-24

Sensing Interrupts 3-30

Masking Interrupts 3-31

Latching Interrupts 3-31

Stacking Status During Interrupts 3-32

Nesting Interrupts 3-32

Interrupting Idle 3-34

Stacks and Sequencing 3-34

Conditional Sequencing 3-39

Sequencer Instruction Summary 3-42

DATA ADDRESS GENERATORS

Overview 4-1
Setting DAG Modes 4-4
Secondary (Alternate) DAG Registers 4-4
Bit-Reverse Addressing Mode 4-6
DAG Page Registers (DMPGx) 4-6
Using DAG Status 4-8
DAG Operations 4-9
Addressing with DAGs 4-9
Addressing Circular Buffers 4-11
Addressing With Bit-Reversed Addresses 4-15
Modifying DAG Registers 4-19
DAG Register Transfer Restrictions 4-20
DAG Instruction Summary 4-21

MEMORY

Overview 5-1
Internal Address and Data Buses 5-3
Internal Data Bus Exchange 5-5
ADSP-2192 Memory Map 5-8
P0 DSP Core Internal Memory Space 5-10
P1 DSP Core Internal Memory Space 5-11
Shared Memory 5-11
Host (PCI/USB) and DSP Internal Memory Space 5-12

System Control Registers 5-13
Shared I/O Memory-mapped Registers 5-13
Arranging Data in Memory 5-13
Data Move Instruction Summary 5-14

DUAL DSP CORES

Overview 6-1
Shared Dual DSP Core Settings 6-1
Unique DSP Core Settings 6-2
Setting Dual DSP Core Features 6-3
System Control 6-3
Power Down Mode Control 6-5
Clock Multiplier Mode Control 6-10
GPIO and Serial EEPROM Mode Control 6-11
Using Dual-DSP Interrupts and Flags 6-13
Controlling I/O Register Bus Accesses 6-17
Using DSP and PCI Mailbox Registers 6-20
Mailbox Status (MBXSTAT) Register 6-21
Mailbox Interrupt Control (MBXCTL) Register 6-24
InBox 0 - PCI/USB to DSP Mailbox 0 (MBX_IN0) Register 6-26
InBox 1 - PCI/USB to DSP Mailbox 1 (MBX_IN1) Register 6-26

OutBox 0 - DSP to PCI/USB Mailbox 0 (MBX_OUT0) Register 6-26
OutBox 1 - DSP to PCI/USB Mailbox 1 (MBX_OUT1) Register 6-26

I/O PROCESSOR

Overview 7-1
Setting I/O Processor—Host Port Modes 7-12
Host Port Buffer Modes 7-14
Host Port Scatter-Gather DMA Mode 7-16
Setting I/O Processor—AC’97 Port Modes 7-18
Host Port DMA Status 7-19
DMA Controller Operation 7-20
Managing DMA Channel Priority 7-21
Chaining DMA Processes 7-22
Host Port DMA 7-22
AC’97 Port DMA 7-24

HOST (PCI/USB) PORT

Overview 8-1
Host Port Selection 8-1
Mode Strap Pin Connections 8-2
PCI Parallel Interface 8-2
Configuration Spaces 8-2
Interactions Between Functions 8-5
Base Address Registers 8-8
Peripheral Device Control Registers 8-9

Power Management Interactions 8-9
PCI Clock Domain 8-11
Peripheral Device Control Register Access 8-12
Resets 8-14
Interrupts 8-14
PCI Control Register 8-16
PCI Port Priority on the PDC Bus 8-18
DSP Mailbox Registers 8-18
InBoxes 8-18
OutBoxes 8-19
Status 8-19
Control 8-21
Indirect Access to I/O Space 8-23
USB Interface 8-25
Overview 8-25
USB Requirements 8-25
Implementation 8-26
Block Diagram of USB Module 8-27
USB-SIE 8-27
Endpoint 0 Control 8-28
MCU 8-28
I/O REG Interface 8-29
DSP DMA Interface 8-29
DSP Code/Data Endpoint Control 8-29

Features and Modes 8-30
Endpoint Types 8-30
Data Transfers 8-30
References 8-32
MCU Register Definitions 8-33
Config USB Device Definitions and Descriptor Tables 8-52
Vendor-Specific Commands 8-55
DSP Register Definitions 8-58
USB DSP Register Definitions 8-58
DSP Code Download 8-65
General Comments 8-67
Starting DSP Code Execution 8-67
MCU ROM Firmware Structure 8-70
MCU Firmware Programmers Model (Endpoint 0) 8-72
Example Initialization Process 8-81
Config Device Definition 8-85
Modem Device Definition 8-85
Serial EEPROM Interface 8-86
Serial EEPROM Changeable Fields for USB Descriptors ... 8-86
ADSP-2192 USB Data Pipe Operations 8-87
OUT Transactions (Host to Device) 8-91
IN Transactions (Device to Host) 8-92
Register and Bit #Defines File 8-94

AC'97 CODEC PORT

Overview 9-1
ADSP-2192 Features and Functionality 9-1
FIFO Control and Status Register 9-3
FIFO Transmit Control and Status Register 9-3
FIFO Receive Control and Status Register 9-5
FIFO DMA Address Registers 9-8
FIFO DMA Current Count Registers 9-8
FIFO DMA Count Registers 9-9
FIFO DMA Next Address Registers 9-9
16-bit Transmit Data Register 9-9
16-bit Receive Data Register 9-9
AC-Link Digital Serial Interface Protocol 9-10
Resetting the AC'97 9-12
ADSP-2192 AC'97 Control Registers 9-13
AC'97 Link Control/Status Register (AC97LCTL) 9-15
AC'97 Link Status Register (AC97STAT) 9-19
AC'97 Slot Enable Register (AC97SEN) 9-21
AC'97 Input Slot Valid Register (AC97SVAL) 9-22
AC'97 AC97STAT:REG and Frame Interrupt Timing 9-22
AC'97 External Codec Register Spaces 9-23
AC'97 Slot Request Register (AC97SREQ) 9-24
AC'97 GPIO Status Register (AC97SIF) 9-24

ADSP-2192 AC'97 Audio Interface 9-25
External Audio Codec (AC'97) Subsystem 9-25
Resource Allocation 9-25
AC'97 2.1 Protocol Summary 9-27
Access to AC'97 Codec Control/Status Registers 9-28
AC'97 2.1 Link Powerdown States 9-30
State Transitions 9-33
Configuring AC'97 Sample Data Streams 9-36

JTAG TEST-EMULATION PORT

SYSTEM DESIGN

Overview 11-1
Sources for Additional Information 11-1
Pin Descriptions 11-3
Clock Signals 11-7
Synchronization Delay 11-9
Configurable Clock Multiplier Considerations 11-10
Maximizing Performance of DSP Algorithms 11-11
Resetting the Processor 11-13
Power On Reset 11-13
Forced Reset Via PCI/USB 11-14
Software Reset 11-14
Reset Progression 11-14
Resets and Software-Forced Rebooting 11-16

Interrupts 11-22

Flag Pins 11-22

Powerup and Powerdown 11-23

Powerup Issues 11-24

Powerup Sequence 11-24

Power Regulators 11-26

2.5V Regulator Options 11-27

Power Management Description 11-28

Powerdown 11-29

Powerdown Control 11-30

Entering and Exiting Powerdown 11-31

Powering Down the USB 11-32

Powering Down the PCI 11-32

Powering Down the AC’97 Link 11-33

Entering Powerdown 11-34

Exiting Powerdown 11-35

Ending Powerdown 11-35

Ending Powerdown with the PORST Pin 11-35

Startup Time after Powerdown 11-36

Using an External TTL/CMOS Clock 11-36

Processor Operation During Powerdown 11-36

Interrupts And Flags 11-37

Conditions for Lowest Power Consumption 11-37

AC’97 Low Power Mode 11-38

Using Powerdown as A Non-Maskable Interrupt 11-39

Emulation 11-39
EZ-KIT Lite 11-40
Recommended Reading 11-41

ADSP-219X DSP CORE REGISTERS

Overview A-1
Core Registers Summary A-2
Register Load Latencies A-5
Core Status Registers A-8
Arithmetic Status (ASTAT) Register A-9
Mode Status (MSTAT) Register A-11
System Status (SSTAT) Register A-14
Computational Unit Registers A-15
Data Register File (DREG) Registers A-16
ALU X- and Y-Input (AX0, AX1, AY0, AY1) Registers A-16
ALU Results (AR) Register A-17
Multiplier X- and Y-Input (MX0, MX1, MY0, MY1) Registers A-17
Multiplier Results (MR2, MR1, MR0) Registers A-17
Shifter Input (SI) Register A-17
Shifter Exponent (SE) and Block Exponent (SB) Registers A-18
Program Sequencer Registers A-18
Interrupt Mask (IMASK) and Interrupt Latch (IRPTL) Registers A-19
Interrupt Control (ICNTL) Register A-20

Indirect Jump Page (IJPG) Register A-21

PC Stack Page (STACKP) and
PC Stack Address (STACKA) Registers A-21

Loop Stack Page (LPSTACKP) and
Loop Stack Address (LPSTACKA) Register A-22

Counter (CNTR) Register A-22

Condition Code (CCODE) Register A-23

Cache Control (CACTL) Register A-23

Data Address Generator Registers A-24

Index Registers (Ix) A-24

Modify Registers (Mx) A-24

Length and Base (Lx,Bx) Registers A-25

Data Memory Page (DMPGx) Register A-25

Memory Interface Registers A-26

PM Bus Exchange (PX) Register A-26

I/O Memory Page (IOPG) Register A-26

Register and Bit #Defines File A-27

ADSP-2192 DSP PERIPHERAL REGISTERS

Overview B-1

Peripheral Registers B-2

DSP Peripherals Architecture B-3

Peripheral Device Register Groups B-4

Summary B-4

ADSP-2192 System Control Registers B-6
STCTLx FIFO Transmit Control Register B-9
SRCTLx FIFO Receive Control Register B-9
xxxADDR DMA Address Register B-10
xxxNXTADDR DMA Next Address Register B-10
xxxCNT DMA Count Register B-10
xxxCURCNT DMA Current Count Register B-10
ADSP-2192 Peripheral Device Control Registers B-11
ADSP-2192 Chip Control Registers B-13
Chip Control (SYSCON) Registers B-14
Power Management Functions B-18
DSP Powerdown (PWRPx) Registers B-19
DSP PLL Control (PLLCTL) Register B-23
General-purpose I/O (GPIO) Control Registers B-24
GPIO Configuration (GPIOCFG) Register B-25
GPIO Polarity (GPIOPOL) Register B-25
GPIO Sticky (GPIOSTKY) Register B-26
GPIO Wakeup Control (GPIOWAKECTL) Register B-26
GPIO Status (GPIOSTAT) Register B-26
GPIO Control (GPIOCTL) Register B-27
GPIO Pullup (GPIOPUP) Register B-27
GPIO Pulldown (GPIOPDN) Register B-27
EEPROM I/O Control/Status (SPROMCTL) Register B-28

Host Mailbox Registers B-30
Overview B-30
CardBus Function Event Registers B-32
CSTSCHG Signal B-33
INTA Signal B-34
CIS Tuple Requirements B-35
AC’97 Controller Registers B-41
AC’97 Link Control/Status Register (AC97LCTL) B-42
AC’97 Link Status Register (AC97STAT) B-42
AC’97 Slot Enable Register (AC97SEN) B-43
AC’97 Input Slot Valid Register (AC97SVAL) B-43
AC’97 Slot Request Register (AC97SREQ) B-44
AC’97 GPIO Status Register (AC97SIF) B-44
AC’97 Codec Registers B-45
AC’97 Codec Register Space-Primary Codec 0 (AC97EXT0) Register B-45
AC’97 Codec Register Space, Secondary Codec 1 (AC97EXT1) Register B-45
AC’97 Codec Register Space, Secondary Codec 2 (AC97EXT2) Register B-46
PCI DMA Address, Count Registers B-46
DMA Control Registers B-46
PCI DMA Control Registers B-46
PCI Interrupt, Control Registers B-47

DMA Transfer Count 0 - Bus Master Sample Transfer Count (PCI_MSTRCNT0) Register B-48
DMA Transfer Count 1 - Bus Master Sample Transfer Count (PCI_MSTRCNT1) Register B-48
DMA Control X - Bus Master Control and Status (PCI_DMACx) Register B-49
PCI Interrupt (PCI_IRQSTAT) Register B-50
PCI Control (PCI_CFGCTL) Register B-53
PCI Configuration Register Space B-54
Commonalities Between the Three Functions B-54
Interactions Between the Three Functions B-55
PCI Configuration Register Space, Function 0 .... B-56
PCI Configuration Register Space, Function 1 .... B-58
PCI Configuration Register Space, Function 2 .... B-59
PCI Configuration Space B-60
Interaction Between Registers B-67
USB DSP Registers B-71
Overview B-71
DSP Register Definitions B-72
DSP Memory Buffer Base Addr Register B-74
DSP Memory Buffer Size Register B-75
DSP Memory Buffer RD Pointer Offset Register B-75
DSP Memory Buffer WR Pointer Offset Register B-76
MCU Register Definitions B-76
USB Endpoint Description Register B-79

USB Endpoint NAK Counter Register B-80
USB Endpoint Stall Policy Register B-81
USB Endpoint 1 Code Download Base Address Register B-82
USB Endpoint 2 Code Download Base Address Register B-83
USB Endpoint 3 Code Download Base Address Register B-84
USB Endpoint 1 Code Download Current Write Pointer Offset Register B-85
USB Endpoint 2 Code Download Current Write Pointer Offset Register B-86
USB Endpoint 3 Code Download Current Write Pointer Offset Register B-87
USB SETUP Token Command Register B-88
USB SETUP Token Data Register B-89
USB SETUP Counter Register B-90
USB Register I/O Address Register B-91
USB Register I/O Data Register B-92
USB Control Register B-93
USB Address/Endpoint Register B-94
USB Frame Number Register B-94
Register and Bit #Defines File B-95

NUMERIC FORMATS

Overview C-1
Un/Signed: Twos-Complement Format C-1
Integer or Fractional C-1
Binary Multiplication C-5
Fractional Mode And Integer Mode C-6
Block Floating-Point Format C-7

ADSP-2192 TIMER

Overview D-1
Timer Architecture D-2
Resolution D-4
Timer Operation D-4
Enabling the Timer D-6

ADSP-2192 INTERRUPTS

Overview E-1
Peripheral Interrupts E-1
Other Interrupt Types E-4

GLOSSARY

Terms G-1

INDEX

1 INTRODUCTION

Purpose

The *ADSP-219x/2192 DSP Hardware Reference* provides architectural information on the ADSP-219x modified Harvard architecture Digital Signal Processor (DSP) core and ADSP-2192 DSP product. The architectural descriptions cover functional blocks, buses, and ports, including all features and processes they support. For programming information, see the *ADSP-219x DSP Instruction Set Reference*.

Audience

DSP system designers and programmers who are familiar with signal processing concepts are the primary audience for this manual. This manual assumes that the audience has a working knowledge of microcomputer technology and DSP-related mathematics.

DSP system designers and programmers who are unfamiliar with signal processing can use this manual, but they should supplement this manual with other texts that describe DSP techniques.

All readers, particularly system designers, should refer to the DSP's data sheet for timing, electrical, and package specifications. For additional suggested reading, see "For More Information About Analog Products" on page 1-24.

Overview—Why Fixed-Point DSP?

A digital signal processor's data format determines its ability to handle signals of differing precision, dynamic range, and signal-to-noise ratios. Because 16-bit, fixed-point DSP math is required for certain DSP coding algorithms, using a 16-bit, fixed-point DSP can provide all the features needed for certain algorithm and software development efforts. Also, a narrower bus width (16-bit as opposed to 32- or 64-bit wide) leads to reduced power consumption and other design savings. The extent to which this is true depends on the fixed-point processor's architecture. 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-2192 DSP is a highly integrated, 16-bit fixed-point DSP that provides many of these design advantages.

ADSP-219x Design Advantages

The ADSP-219x family DSPs are high-performance 16-bit DSPs for communications, instrumentation, industrial/control, voice/speech, medical, military, and other applications. These DSPs provide a DSP core that is compatible with previous ADSP-2100 family DSPs, but they also provide many additional features. The ADSP-219x core combines with on-chip peripherals to form a complete system-on-a-chip. The off-core peripherals add on-chip SRAM, integrated I/O peripherals, timer, and interrupt controller.

The ADSP-219x architecture balances a high performance processor core with high performance buses (PM, DM, DMA). In the core, every computational 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 ADSP-2192 processor.

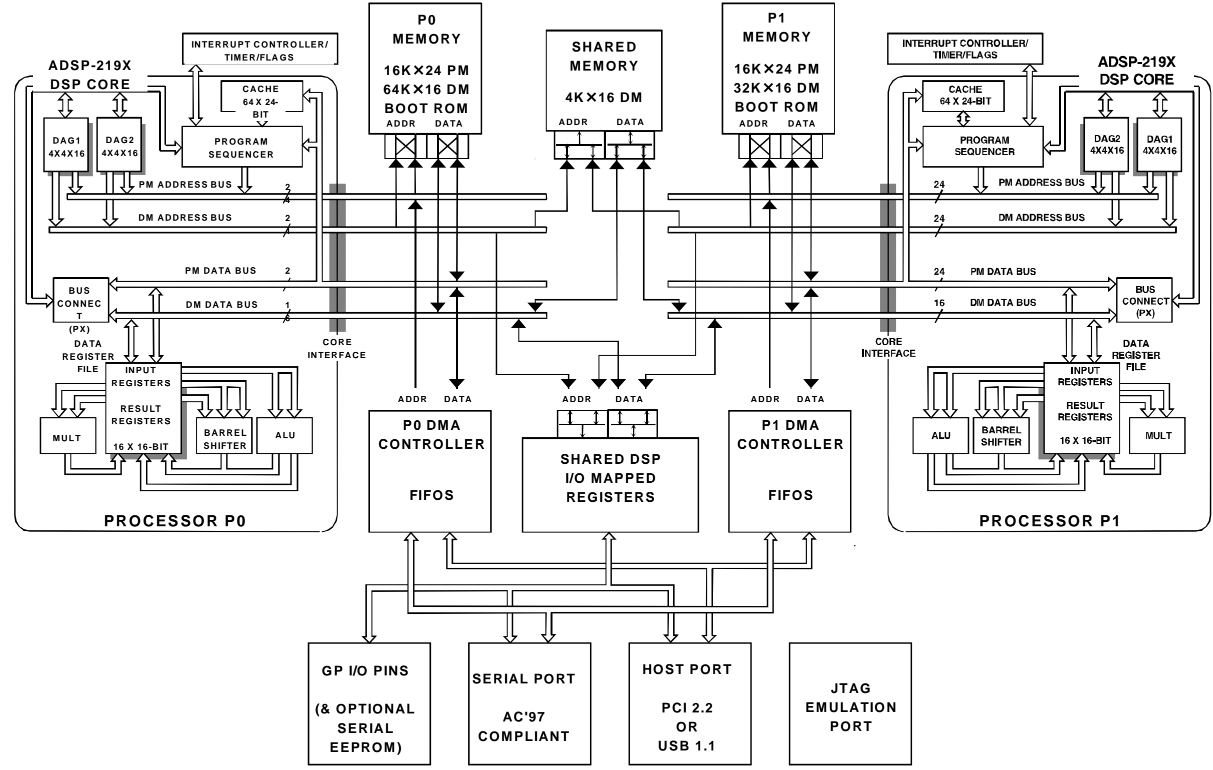


Figure 1-1. ADSP-2192 Block Diagram

This diagram illustrates the following ADSP-2192 architectural features:

- Computation units for the ADSP-219x family—multiplier, ALU, shifter, and data register file
- Program sequencer for the ADSP-219x family, with related instruction cache, interval timer, and Data Address Generators (DAG1 and DAG2)
- PCI/USB Host port

- AC'97 codec port
- SRAM for the ADSP-2192
- Input/Output (I/O) processor with integrated DMA controllers
- JTAG Test Access Port for board test and emulation on the ADSP-2192

Figure 1-1 also shows the two cores of the ADSP-2192 (processors P0 and P1). Additionally, it shows the four on-chip buses of the ADSP-2192: the Program Memory Address (PMA) bus, Program Memory Data (PMD) bus, Data Memory Address (DMA) bus, and the Data Memory Data (DMD) bus. During a single cycle, these buses let the processor access two data operands (one from PMD and one from DMD), and access an instruction (from the cache).

Further, the ADSP-219x 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

Unconstrained Data Flow. The ADSP-219x has a modified Harvard architecture combined with a data register file. In every cycle, the DSP can:

- Read two values from memory or write one value to memory
- Complete one computation
- Write up to three values back to the register file

Fast, Flexible Arithmetic. The ADSP-219x family DSPs execute all computational instructions in a single cycle. They provide both fast cycle times and a complete set of arithmetic operations.

40-Bit Extended Precision. The DSP handles 16-bit integer and fractional formats (twos-complement and unsigned). The processors carry extended precision through result registers in their computation units, limiting intermediate data truncation errors.

Dual Address Generators. The DSP has two data address generators (DAGs) that provide immediate or indirect (pre- and post-modify) addressing. Modulus and bit-reverse operations are supported with only memory page constraints on data buffer placement.

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

ADSP-219x Architecture Overview

An ADSP-219x is a single-chip microcomputer optimized for digital signal processing (DSP) and other high speed numeric processing applications. These DSPs provide 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-219x architecture, which appears in Figure 1-1 on page 1-3.

The ADSP-2192 combines the ADSP-219x family base architecture (three computational units, two data address generators, and a program sequencer) with PCI/USB interface, AC’97 serial port, a programmable timer, a DMA controller, general-purpose Programmable Flag pins, extensive interrupt capabilities, and on-chip program and data memory blocks.

The ADSP-2192 architecture is code compatible with ADSP-218x family DSPs. Though the architectures are compatible, the ADSP-2192 architecture has a number of enhancements over the ADSP-218x architecture. The enhancements to computational units, data address generators, and program sequencer make the ADSP-2192 more flexible and even easier to program than the ADSP-218x DSPs.

Indirect addressing options provide addressing flexibility—pre-modify with no update, pre- and post-modify by an immediate 8-bit, two's-complement value and base address registers for easier implementation of circular buffering.

The ADSP-2192 integrates 128K words of on-chip memory configured as 32K words (24-bit) of program RAM (16K words each on DSP P0 and DSP P1) and 96K words (16-bit) of data RAM (64K words on DSP P0 and 32K words on DSP P1). Power-down circuitry is also provided to meet the low power needs of battery operated portable equipment.

The ADSP-2192's flexible architecture and comprehensive instruction set support multiple operations in parallel. For example, in one processor cycle, each core of the ADSP-2192 can:

- Generate an address for the next instruction fetch
- Fetch the next instruction
- Perform one or two data moves
- Update one or two data address pointers
- Perform a computational operation

DSP Core Architecture

The ADSP-219x instruction set provides flexible data moves and multifunction (one or two data moves with a computation) instructions. Every single-word instruction can be executed in a single processor cycle. The ADSP-219x assembly language uses an algebraic syntax for ease of coding and readability. A comprehensive set of development tools supports program development.

Figure 1-1 on page 1-3 shows the architecture of the ADSP-219x core. It contains three independent computational units: the ALU, the multiplier/accumulator, and the shifter.

The computational units process 16-bit data from the register file and have provisions to support multiprecision computations. The ALU performs a standard set of arithmetic and logic operations; division primitives also are supported. The multiplier performs single-cycle multiply, multiply/add, and multiply/subtract operations. The multiplier now has two 40-bit accumulator results. The shifter performs logical and arithmetic shifts, normalization, denormalization, and derive exponent operations. The shifter can efficiently implement numeric format control, including multiword and block floating-point representations.

Register-usage rules influence placement of input and results within the computational units. For all unconditional, non-multi-function instructions, the computational units' data registers act as a data register file, permitting any input or result register to provide input to any unit for a computation. For feedback operations, the computational units let the output (result) of any unit be input to any unit on the next cycle. For conditional or multifunction instructions, there are restrictions limiting which data registers may provide inputs or receive results from each computational unit. For more information, see "Multifunction Computations" on page 2-60.

A powerful program sequencer controls the flow of instruction execution. The sequencer supports conditional jumps, subroutine calls, and low interrupt overhead. With internal loop counters and loop stacks, the ADSP-2192 executes looped code with zero overhead; no explicit jump instructions are required to maintain loops.

Two data address generators (DAGs) provide addresses for simultaneous dual operand fetches (from data memory and program memory). Each DAG maintains and updates four 16-bit address pointers. Whenever the pointer is used to access data (indirect addressing), it is pre- or post-modified by the value of one of four possible modify registers. A length value and base address may be associated with each pointer to implement automatic modulo addressing for circular buffers.

Page registers in the DAGs allow circular addressing within 64K word boundaries of each of the 256 memory pages, but these buffers may not cross page boundaries. Secondary registers duplicate all the primary registers in the DAGs; switching between primary and secondary registers provides a fast context switch.

Efficient data transfer in the core is achieved by using internal buses:

- Program Memory Address (PMA) Bus
- Program Memory Data (PMD) Bus
- Data Memory Address (DMA) Bus
- Data Memory Data (DMD) Bus
- IO or DMA Address Bus
- IO or DMA Data Bus

Program memory can store both instructions and data, permitting the ADSP-219x to fetch two operands in a single cycle, one from program memory and one from data memory. The DSP's dual memory buses also let the ADSP-219x core fetch an operand from data memory and the next instruction from program memory in a single cycle.

DSP Peripherals Architecture

Figure 1-1 on page 1-3 shows the DSP's on-chip peripherals, as part of a typical ADSP-2192 system with peripheral connections.The ADSP-2192 has a 16-bit PCI/USB host port. This port provides either PCI or USB functionality via the Peripheral Device Control (PDC) bus.

The ADSP-2192 can respond to up to thirteen interrupts, using a priority scheme implemented by the interrupt controller.

Memory Architecture

The ADSP-2192 integrates 128K words of on-chip memory configured as 32K words (24-bit) of program RAM (16K words each on DSP P0 and DSP P1) and 96K words (16-bit) of data RAM (64K words on DSP P0 and 32K words on DSP P1). Power-down circuitry is also provided to meet the low power needs of battery operated portable equipment.

For more information on these blocks, see the section "ADSP-2192 Memory Map" on page 5-8, which discusses the memory map in detail.

Figure 1-2 shows the ADSP-2192's memory map.

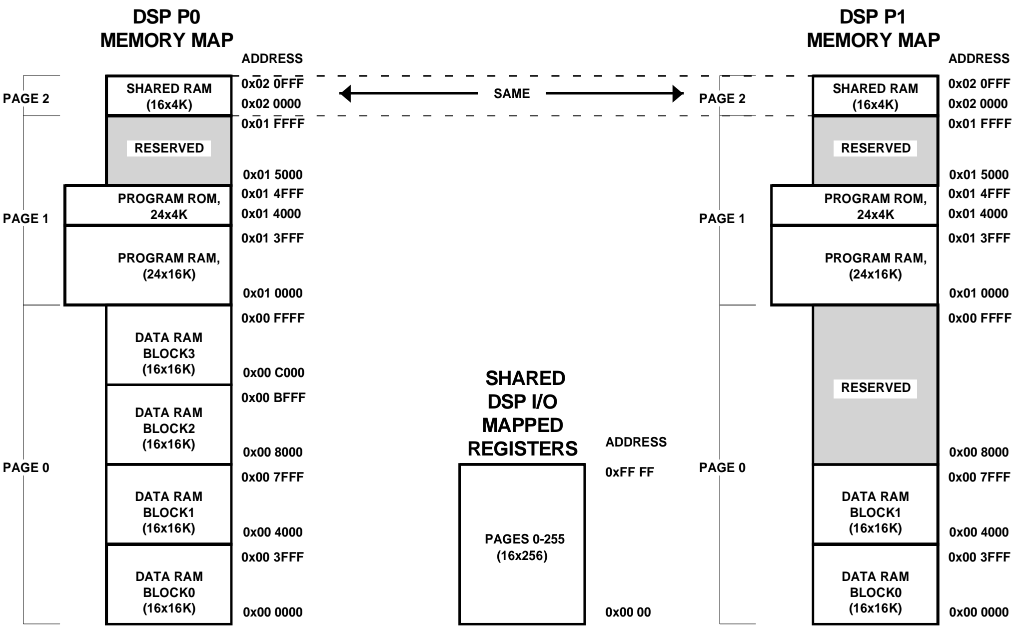


Figure 1-2. ADSP-2192 Memory Maps

Internal (On-Chip) Memory

The ADSP-2192's unified program and data memory space consists of 16M locations that are accessible through two 24-bit address buses, the PMA and DMA buses. The DSP uses slightly different mechanisms to generate a 24-bit address for each bus. The DSP has three functions that support access to the full memory map:

- The DAGs generate 24-bit addresses for data fetches from the entire DSP memory address range. Because DAG index (address) registers are 16 bits wide and hold the lower 16-bits of the address, each of the DAGs has its own 8-bit page register (`DMPGx`) to hold the most significant eight address bits. Before a DAG generates an address, the program must set the DAG's `DMPGx` register to the appropriate memory page.

- The Program Sequencer generates the addresses for instruction fetches. For relative addressing instructions, the program sequencer bases addresses for relative jumps, calls, and loops on the 24-bit Program Counter (`PC`). For direct addressing instructions (two-word instructions), the instruction provides an immediate 24-bit address value. The `PC` allows linear addressing of the full 24 bit address range.

- The Program Sequencer relies on an 8-bit Indirect Jump page (`IJPG`) register to supply the most significant eight address bits for indirect jumps and calls that use a 16-bit DAG address register for part of the branch address. Before a cross page jump or call, the program must set the program sequencer's `IJPG` register to the appropriate memory page.

The ADSP-2192 has 4K words of on-chip ROM that holds boot routines. If peripheral booting is selected, the DSP starts executing instructions from the on-chip boot ROM, which starts the boot process from the selected peripheral. For more information, see "Reset Modes" on page 1-14. The on-chip boot ROM is located on Page 255 in the DSP's memory map.

Interrupts

The interrupt controller lets the DSP respond to thirteen interrupts with minimum overhead.

DMA Controller

The ADSP-2192 has a DMA controller that supports automated data transfers with minimal overhead for the DSP core. Cycle stealing DMA transfers can occur between the ADSP-2192's internal memory and any of its DMA capable peripherals. Additionally, DMA transfers also can be accomplished between any of the DMA capable peripherals. DMA capable peripherals include the PCI, USB, and AC'97. Each individual DMA capable peripheral has one or more dedicated DMA channels. DMA sequences do not contend for bus access with the DSP core, instead DMAs "steal" cycles to access memory.

PCI Port

The ADSP-2192 can interface with a host computer through a PCI port. The PCI port accesses the DSPs via the Peripheral Device Control (PDC) bus. The PCI port connects through the internal PCI interface to the PDC bus.

USB Port

The ADSP-2192 can interface with a host computer through a USB port. The USB port accesses the DSPs via the Peripheral Device Control (PDC) bus. The USB port connects through the internal USB interface to the PDC bus.

AC’97 Interface

The ADSP-2192 includes an AC’97 interface that complies with the AC’97 specification. The AC’97 interface connects the host’s Digital Controller (DC) chip set and between one and four analog codecs.

Low Power Operation

All pins on the ADSP-2192 remain active as long as power is maintained to the chip. This chip does not have a specifically-defined powerdown state; at any time either or both of the two processors can be in a low power state, and any or all of the interfaces can be in a low power state. Additionally, each peripheral interface (USB, PCI, and AC’97) can be put into a low power mode, as described in “System Design” on page 11-1.

Clock Signals

The ADSP-2192 can be clocked by a crystal oscillator. If a crystal oscillator is used, the crystal should be connected across the `XTALI/O` pins, with two capacitors connected as shown in Figure 1-3 on page 1-14. Capacitor values are dependent on crystal type and should be specified by the crystal manufacturer. A parallel-resonant, fundamental frequency, microprocessor-grade 24.576 MHz crystal should be used for this configuration.

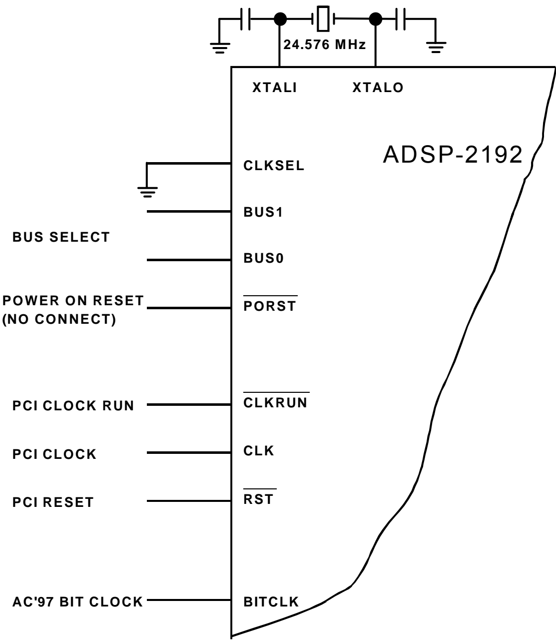


Figure 1-3. ADSP-2192 External Crystal Connections

Reset Modes

The ADSP-2192 can be reset in three ways: Power On Reset, Software Reset, or Forced Reset Via PCI or USB.

See "Resetting the Processor" on page 11-13 for more details about booting.

JTAG Port

The ADSP-2192 includes a JTAG port. Emulators use the JTAG port to monitor and control the DSP during emulation. Emulators using this port provide full-speed emulation with access to inspect and modify memory, registers, and processor stacks. JTAG-based emulation is non-intrusive and does not affect target system loading or timing. Note that the ADSP-2192 JTAG does not support boundary scan.

Development Tools

The ADSP-219x is supported by VisualDSP++®, an easy-to-use project management environment, comprised of an Integrated Development and Debugging Environment (IDDE). VisualDSP++ lets you 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.

Flexible Project Management. The IDDE provides flexible project management for the development of DSP applications. It provides you with access to all the activities necessary to create and debug DSP projects. You can create or modify source files or view listing or map files with the IDDE Editor. This powerful Editor includes multiple language syntax highlighting, OLE drag and drop, bookmarks, and standard editing operations such as undo/redo, find/replace, copy/paste/cut, and go to.

VisualDSP++ includes access to the DSP C/C++ Compiler, C Run-time Library, Assembler, Linker, Loader, Splitter, and Simulator. You specify options for these Tools through property page dialog boxes. These options control how the tools process inputs and generate outputs, and the options have a one-to-one correspondence to the tools' command-line switches. You can define these options once or modify them to meet changing development needs. You also can access the Tools from the operating system command line if you choose.

Greatly Reduced Debugging Time. VisualDSP++ has an easy-to-use, common interface for all processor simulators and emulators available through Analog Devices and third parties or custom developments. It has many features that greatly reduce debugging time. You can view C source interspersed with the resulting Assembly code. You can profile execution of a range of instructions in a program; set simulated watch points on hardware and software registers, program and data memory; and trace instruction execution and memory accesses.

These features enable you to correct coding errors, identify bottlenecks, and examine DSP performance. You can select any combination of registers to view in a single customizable window. VisualDSP++ can also generate inputs, outputs, and interrupts, so you can simulate real-world application conditions.

Software Development Tools. Software development tools, which support the ADSP-219x family, let you develop applications that take full advantage of the DSP architecture, including shared memory and memory overlays. Software development tools include the C/C++ Compiler, C Run-time Library, DSP and Math Libraries, Assembler, Linker, Loader, Splitter, and Simulator.

C/C++ Compiler and Assembler. The C/C++ Compiler generates efficient code that is optimized for both code density and execution time. The C/C++ Compiler allows you to include Assembly language statements inline. Because of this, you can program in C/C++ and still use Assembly for time-critical loops.

You can also use pretested Math, DSP, and C Run-time Library routines to help shorten your time to market. The ADSP-219x family assembly language is based on an algebraic syntax that is easy to learn, program, and debug.

Linker and Loader. The Linker provides flexible system definition through Linker Description Files (.LDF). In a single LDF, you can define different types of executables for a single processor or multiprocessor system. The Linker resolves symbols over multiple executables, maximizes memory use, and easily shares common code among multiple processors. The Loader allows multiprocessor system configuration with smaller code and faster boot time.

Differences from Previous DSPs

This section identifies differences between the ADSP-219x DSPs and previous ADSP-2100 family DSPs: ADSP-210x, ADSP-211x, ADSP-217x, and ADSP-218x. The ADSP-219x preserves much of the core ADSP-2100 family architecture, while extending performance and functionality. For background information on previous ADSP-2100 family DSPs, see the *ADSP-2100 Family User's Manual*. For background information on the ADSP-218x family DSPs, see the *ADSP-218x DSP Hardware Reference*.

The sections that follow describe key differences and enhancements of the ADSP-219x over previous ADSP-2100 family DSPs. These enhancements also lead to some differences in the instruction sets between DSP families. For more information, see the *ADSP-219x DSP Instruction Set Reference*.

Computational Units and Data Register File

The ADSP-219x DSP's computational units differ from those of the ADSP-218x's, because the ADSP-219x data registers act as a register file for unconditional, single-function instructions. In these instructions, any data register may be an input to any computational unit. For conditional and/or multifunction instructions, the ADSP-219x and ADSP-218x DSP families have the same data register usage restrictions — AX and AY for ALU, MX and MY for the multiplier, and SI for shifter inputs. For more information, see "Computational Units" on page 2-1.

Shifter Result (SR) Register as Multiplier Dual Accumulator

The ADSP-219x architecture introduces a new 16-bit register in addition to the SR0 and SR1 registers, the combination of which composes the 40-bit wide SR register on the ADSP-218x DSPs. This new register, called SR2, can be used in multiplier or shift operations (lower 8 bits) and as a full 16-bit-wide scratch register. As a result, the ADSP-219x DSP has two 40-bit-wide accumulators, MR and SR. The SR dual accumulator has replaced the multiplier feedback register MF, as shown in the following example:|

ADSP-218x Instruction	ADSP-219x Instruction (Replacement)
MF=MR+MX0*MY1(UU); IF NOT MV MR=AR*MF;	MR=MR+MX0*MY1(UU); IF NOT MV MR=AR*SR2;

Shifter Exponent (SE) Register is not Memory Accessible

The ADSP-218x DSPs use SE as a data or scratch register. The SE register of the ADSP-219x architecture is not accessible from the data or program memory buses. Therefore, the multifunction instructions of the ADSP-218x that use SE as a data or scratch register, should use one of the data file registers (DREG) as a scratch register on the ADSP-219x DSP.

ADSP-218x Instruction	ADSP-219x Instruction (Replacement)
SR=Lshift MR1(HI), SE=DM(I6,M5);	SR=Lshift MR1(HI), AX0=DM(I6,M5);

Conditions (SWCOND) and Condition Code (CCODE) Register

The ADSP-219x DSP changes support for the ALU Signed (AS) condition and supports additional arithmetic and status condition testing with the Condition Code (CCODE) register and Software Condition (SWCOND) test. The two conditions are SWCOND and Not SWCOND. The usage of the ADSP-219x's and most ADSP-218x's arithmetic conditions (EQ, NE, GE, GT, LE, LT, AV, Not AV, AC, Not AC, MV, Not MV) are compatible.

The new Shifter Overflow (SV) condition of the ADSP-219x architecture is a good example of how the CCODE register and SWCOND test work. The ADSP-219x DSP's Arithmetic Status (ASTAT) register contains a bit indicating the status of the shifter's result. The shifter is a computational unit that performs arithmetic or logical bitwise shifts on fields within a data register. The result of the operation goes into the Shifter Result (SR2, SR1, and SR0, which are combined into SR) register. If the result overflows the SR register, the Shifter Overflow (SV) bit in the ASTAT register records this overflow/underflow condition for the SR result register (0 = No overflow or underflow, 1 = Overflow or underflow).

For the most part, bits (status condition indicators) in the ASTAT register correspond to condition codes that appear in conditional instructions. For example, the AZ (ALU Zero) bit in ASTAT corresponds to the EQ (ALU result equals zero) condition and would be used in code like this:

```
IF EQ AR = AX0 + AY0;
/* if the ALU result (AR) register is zero, add AX0 and AY0 */
```

The SV status condition in the ASTAT bits does not correspond to a condition code that can be directly used in a conditional instruction. To test for this status condition, software selects a condition to test by loading a value into the Condition Code (CCODE) register and uses the Software Condition

(SWCOND) condition code in the conditional instruction. The DSP code would look like this:

```
CCODE = 0x09; Nop;              // set CCODE for SV condition
IF SWCOND SR = MR0 * SR1 (UU);  // mult unsigned X and Y
```

The Nop after loading the CCODE register accommodates the one cycle effect latency of the CCODE register.

The ADSP-218x DSP supports two conditions to detect the sign of the ALU result. On the ADSP-219x, these two conditions (Pos and Neg) are supported as AS and Not AS conditions in the CCODE register. For more information on CCODE register values and SWCOND conditions, see "Conditional Sequencing" on page 3-39.

Unified Memory Space

The ADSP-219x architecture has a unified memory space with separate memory blocks to differentiate between 24- and 16-bit memory. In the unified memory, the term *program* or *data memory* only has semantic significance; a physical address determines the "PM" or "DM" functionality. It is best to revise any code with non-symbolic addressing in order to use the new tools.

Data Memory Page (DMPG1 and DMPG2) Registers

The ADSP-219x processor introduces a paged memory architecture that uses 16-bit DAG registers to access 64K pages. The 16-bit DAG registers correspond to the lower 16 bits of the DSP's address buses, which are 24-bit wide. To store the upper 8 bits of the 24-bit address, the ADSP-219x DSP architecture uses two additional registers, DMPG1 and DMPG2. DMPG1 and DMPG2 work with the DAG registers I0-I3 and I4-I7, respectively.

Data Address Generator (DAG) Addressing Modes

The ADSP-219x architecture provides additional flexibility over the ADSP-218x DSP family in DAG addressing modes:

- Pre-modify without update addressing in addition to the post-modify with update mode of the ADSP-218x instruction set:

  ```
  DM(I0+M1) = AR;     /* pre-modify syntax */
  DM(I0+=M1) = AR;    /* post-modify syntax */
  ```

- Pre-modify and post-modify with an 8-bit two's-complement immediate modify value instead of an M register:

  ```
  AX0 = PM(I5+-4);  /* pre-modify syntax (for modifier = -4)*/
  AX0 = PM(I5+=4); /* post-modify syntax (for modifier = 4) */
  ```

- DAG modify with an 8-bit two's-complement immediate-modify value:

  ```
  Modify(I7+=0x24);
  ```

Base Registers for Circular Buffers.

The ADSP-219x processor eliminates the existing hardware restriction of the ADSP-218x DSP architecture on a circular buffer starting address. ADSP-219x enables declaration of any number of circular buffers by designating B0-B7 as the base registers for addressing circular buffers; these base registers are mapped to the "register" space on the core.

Program Sequencer, Instruction Pipeline, and Stacks

The ADSP-219x DSP core and inputs to the sequencer differ for various members of the ADSP-219x family DSPs. The main differences between the ADSP-218x and ADSP-219x sequencers are that the ADSP-219x sequencer has:

- A 6-stage instruction pipeline, which works with the sequencer's loop and PC stacks, conditional branching, interrupt processing, and instruction caching.
- A wider branch execution range, supporting:
  - 13-bit non-delayed or delayed relative conditional `Jump`
  - 16-bit non-delayed or delayed relative unconditional `Jump` or `Call`
  - Conditional non-delayed or delayed indirect `Jump` or `Call` with address pointed to by a DAG register
  - 24-bit conditional non-delayed absolute long `Jump` or `Call`
- A narrowing of the `Do`/`Until` termination conditions to Counter Expired (`CE`) and `Forever`.

Conditional Execution (Difference in Flag Input Support)

Unlike the ADSP-218x DSP family, the ADSP-219x processors do not directly support a conditional `Jump`/`Call` instruction execution based on flag input. Instead, the ADSP-219x supports this type of conditional execution with the `CCODE` register and `SWCOND` condition. For more information, see "Conditions (SWCOND) and Condition Code (CCODE) Register" on page 1-19.

The ADSP-219x architecture has 16 programmable flag pins that can be configured as either inputs or outputs. The flags can be checked by using a software condition flag.

ADSP-218x Instruction	ADSP-219x Instruction Replacement

`If Not FLAG_IN AR=MR0 And 8192;`	`CCODE=0x03;`
	`NOP;`
	`If Not SWCOND AR=MR0 And 8192;`

	`IOPG = 0x06;`
	`AX0=IO();`
	`AR=Tstbit 11 OF AX0;`
	`If EQ AR=MR0 And 8192;`

Execution Latencies (Different for JUMP Instructions)

The ADSP-219x processor has an instruction pipeline (unlike ADSP-218x DSPs) and branches execution for immediate `Jump` and `Call` instructions in four clock cycles if the branch is taken. To minimize branch latency, ADSP-219x programs can use the delayed branch option on jumps and calls, reducing branch latency by two cycles. This savings comes from executing of two instructions following the branch before the `Jump`/`Call` occurs.

Instruction Set Enhancements

ADSP-219x provides near source code compatibility with the previous family members, easing the process of porting code. All computational instructions (but not all registers) from previous ADSP-2100 family DSPs are available in ADSP-219x. New instructions, control registers, or other facilities, required to support the new feature set of ADSP-219x core are:

- Program flow control differences (pipeline execution and changes to looping)
- Memory accessing differences (DAG support and memory map)
- Peripheral I/O differences (additional ports and added DMA functionality)

For more information, see the *ADSP-219x DSP Instruction Set Reference*.

For More Information About Analog Products

Analog Devices is online on the internet at `http://www.analog.com`. Our Web pages provide information on the company and products, including access to technical information and documentation, product overviews, and product announcements.

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

- Visit our World Wide Web site at `www.analog.com`
- FAX questions or requests for information to 1(781)461-3010.
- Access the DSP Division File Transfer Protocol (FTP) site at `ftp ftp.analog.com` or `ftp 137.71.23.21` or `ftp://ftp.analog.com`.

For Technical or Customer Support

You can reach our Customer Support group in the following ways:

- E-mail questions to `dsp.support@analog.com` or `dsp.europe@analog.com` (European customer support)
- Telex questions to 924491, TWX:710/394-6577
- Cable questions to ANALOG NORWOODMASS
- Contact your local ADI sales office or an authorized ADI distributor
- Send questions by mail to:
  ```
  Analog Devices, Inc.
  DSP Division
  One Technology Way
  P.O. Box 9106
  Norwood, MA 02062-9106
  USA
  ```

What's New in This Manual

This is Revision 1.1 of the *ADSP-219x/2192 DSP Hardware Reference*. Changes to this book from the first edition include only the reporting of document errata. See Pages 11-34, A-23, B-23.

Related Documents

For more information about Analog Devices DSPs and development products, see the following documents:

- *ADSP-2192 DSP Microcomputer Data Sheet*
- *ADSP-219x DSP Instruction Set Reference*

- *VisualDSP++ 2.0 Getting Started Guide for ADSP-21xx DSPs*
- *VisualDSP++ 2.0 User's Guide for ADSP-21xx DSPs*
- *VisualDSP++ 2.0 C/C++ Compiler and Library Manual for ADSP-219x DSPs*
- *VisualDSP++ 2.0 Assembler and Preprocessor Manual for ADSP-219x DSPs*
- *VisualDSP++ 2.0 Linker and Utilities Manual for ADSP-21xx DSPs*
- *VisualDSP++ Kernel (VDK) User's Guide*

All the manuals are included in the software distribution CD-ROM. To access these documents within VisualDSP++:

1. Choose `Help Topics` from the VisualDSP++ `Help` menu.
2. Select the `Reference` book icon.
3. Select the `Online Manuals` topic.
4. Click the "`Click here to view online manuals`" button.

   A list of manuals displays.
5. Select the document you want to view.

If you are not using VisualDSP++, you can manually access these PDF files from the CD-ROM using Adobe Acrobat PDF format.

Conventions

Table 1-1 identifies and describes text conventions used in this manual.

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

Table 1-1. Notation Conventions

Example	Description
AX0, SR, PX	Register names appear in UPPERCASE and keyword font
TMR0E, $\overline{\text{RESET}}$	Pin names appear in UPPERCASE and keyword font; active low signals appear with an $\overline{\text{OVERBAR}}$.
DRx, $\overline{\text{MS3-0}}$	Register and pin names in the text may refer to groups of registers or pins. When a lowercase "x" appears in a register name (e.g., DRx), that indicates a set of registers (e.g., DR0, DR1, and DR2). A range also may be shown with a hyphen (e.g., $\overline{\text{MS3-0}}$ indicates $\overline{\text{MS3}}$, $\overline{\text{MS2}}$, $\overline{\text{MS1}}$, and $\overline{\text{MS0}}$).
If, Do/Until	Assembler instructions (mnemonics) appear in mixed-case and keyword font
[this,that] |this,that|	Assembler instruction syntax summaries show optional items in two ways. When the items are optional and none is required, the list is shown enclosed in square brackets, []. When the choices are optional, but one is required, the list is shown enclosed in vertical bars, ||.
0xabcd, b#1111	A 0x prefix indicates hexadecimal; a b# prefix indicates binary
	A note, providing information of special interest or identifying a related DSP topic.

Table 1-1. Notation Conventions

Example	Description
⃠	A caution, providing information on critical design or programming issues that influence operation of the DSP.
Click Here	In the online version of this document, a cross reference acts as a hypertext link to the item being referenced. Click on blue references (Table, Figure, or section names) to jump to the location.

2 COMPUTATIONAL UNITS

Overview

The DSP's computational units perform numeric processing for DSP algorithms. The three computational units are the arithmetic/logic unit (ALU), multiplier/accumulator (multiplier), and shifter. These units get data from registers in the data register file. Computational instructions for these units provide fixed-point operations, and each computational instruction can execute in a single cycle.

The computational units handle different types of operations. The ALU performs arithmetic and logic operations. The multiplier does multiplication and executes multiply/add and multiply/subtract operations. The shifter executes logical shifts and arithmetic shifts. Also, the shifter can derive exponents.

Data flow paths through the computational units are arranged in parallel, as shown in Figure 2-1 on page 2-3. The output of any computational unit may serve as the input of any computational unit on the next instruction cycle. Data moving in and out of the computational units goes through a data register file, consisting of sixteen primary registers and sixteen secondary registers. Two ports on the register file connect to the PM and DM data buses, allowing data transfer between the computational units and memory.

The DSP's assembly language provides access to the data register files. The syntax lets programs move data to and from these registers and specify a computation's data format at the same time. For information on the data registers, see "Data Register File" on page 2-57.

Figure 2-1 provides a graphical guide to the other topics in this chapter. First, a description of the MSTAT register shows how to set rounding, data format, and other modes for the computational units. Next, an examination of each computational unit provides details on operation and a summary of computational instructions. Looking at inputs to the computational units, details on register files, and data buses identify how to flow data for computations. Finally, details on the DSP's advanced parallelism reveal how to take advantage of conditional and multifunction instructions.

The diagrams in Figure 2-1 on page 2-3 and Figure 2-17 on page 2-62 describe the relationship between the ADSP-219x data register file and computational units: multiplier, ALU, and shifter.

Figure 2-1 shows how unconditional, single-function multiplier, ALU, and shifter instructions have unrestricted access to the data registers in the register file. Figure 2-1 also indicates that the Results Bus lets the computational units use any result registers (MR2, MR1, MR0, SR1, SR0, or AR) as an X-input for any operation. The upper part of the Shifter Results (SR) register, SR2, may not serve as feedback over the results bus.

The MR2 and SR2 registers differ from the other results registers. As a data register file register, MR2 and SR2 are 16-bit registers that may be X- or Y-inputs to the multiplier, ALU, or shifter. As result registers (part of MR or SR), only the lower 8-bits of MR2 or SR2 hold data (the upper 8-bits are sign extended). This difference (16-bit as input, 8-bit as output) influences how code can use the MR2 and SR2 registers. This sign extension appears in Figure 2-12 on page 2-30.

Using register-to-register move instructions, the data registers can load (or be loaded from) the Shifter Block (SB) and Shifter Exponent (SE) registers, but the SB and SE registers may not provide X- or Y-input to the computational units. The SB and SE registers serve as additional inputs to the shifter.

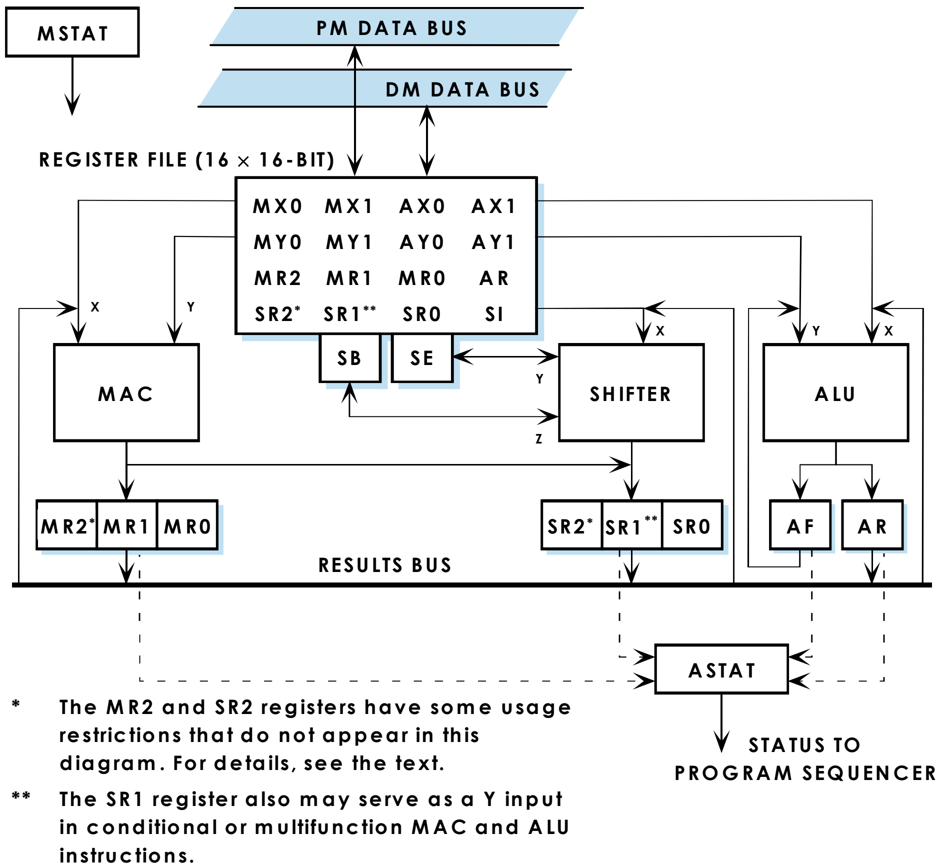


Figure 2-1. Register Access—Unconditional, Single-Function Instructions

The shaded boxes behind the data register file and the SB, SE, MR, SR, AR, and AF registers indicate that secondary registers are available for these registers. For more information, see “Secondary (Alternate) Data Registers” on page 2-59.

The Mode Status (MSTAT) register input sets arithmetic modes for the computational units, and the Arithmetic Status (ASTAT) register records status/conditions for the computation operations' results.

Using Data Formats

ADSP-219x DSPs are 16-bit, fixed-point machines. Most operations assume a two's complement number representation, while others assume unsigned numbers or simple binary strings. Special features support multi-word arithmetic and block floating-point. For detailed information on each number format, see "Numeric Formats" on page C-1.

In ADSP-219x family arithmetic, signed numbers are always in two's complement format. These DSPs do not use signed magnitude, one's complement, BCD, or excess-n formats.

Binary String

The binary string format is the least complex binary notation; sixteen bits are treated as a bit pattern. Examples of computations using this format are the logical operations: NOT, AND, OR, XOR. These ALU operations treat their operands as binary strings with no provision for sign bit or binary point placement.

Unsigned

Unsigned binary numbers may be thought of as positive, having nearly twice the magnitude of a signed number of the same length. The DSP treats the least significant words of multiple precision numbers as unsigned numbers.

Signed Numbers: Two's Complement

In ADSP-219x DSP arithmetic, the term "signed" refers to two's complement. Most ADSP-219x family operations presume or support two's complement arithmetic.

Fractional Representation: 1.15

ADSP-219x DSP arithmetic is optimized for numerical values in a fractional binary format denoted by 1.15 ("one dot fifteen"). In the 1.15 format, there is one sign bit (the MSB) and fifteen fractional bits, which represent values from –1 up to one LSB less than +1.

Figure 2-2 shows the bit weighting for 1.15 numbers. These are examples of 1.15 numbers and their decimal equivalents.

1.15 NUMBER (HEXADECIMAL)	DECIMAL EQUIVALENT
0X0001	0.000031
0X7FFF	0.999969
0XFFFF	–0.000031
0X8000	–1.000000

-2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}

Figure 2-2. Bit Weighting for 1.15 Numbers

ALU Data Types

All operations on the ALU treat operands and results as 16-bit binary strings, except the signed division primitive (`DIVS`). ALU result status bits treat the results as signed, indicating status with the overflow (`AV`) condition code and the negative (`AN`) flag.

The logic of the overflow bit (AV) is based on two's complement arithmetic. It is set if the MSB changes in a manner not predicted by the signs of the operands and the nature of the operation. For example, adding two positive numbers must generate a positive result; a change in the sign bit signifies an overflow and sets AV. Adding a negative and a positive may result in either a negative or positive result, but cannot overflow.

The logic of the carry bit (AC) is based on unsigned-magnitude arithmetic. It is set if a carry is generated from bit 16 (the MSB). The (AC) bit is most useful for the lower word portions of a multiword operation.

ALU results generate status information. For more information on using ALU status, see "ALU Status Flags" on page 2-18.

Multiplier Data Types

The multiplier produces results that are binary strings. The inputs are "interpreted" according to the information given in the instruction itself (signed times signed, unsigned times unsigned, a mixture, or a rounding operation). The 32-bit result from the multiplier is assumed to be signed, in that it is sign-extended across the full 40-bit width of the MR or SR register set.

The ADSP-219x DSPs support two modes of format adjustment: the fractional mode for fractional operands (1.15 format with 1 signed bit and 15 fractional bits) and the integer mode for integer operands (16.0 format).

When the processor multiplies two 1.15 operands, the result is a 2.30 (2 sign bits and 30 fractional bits) number. In the fractional mode, the multiplier automatically shifts the multiplier product (P) left one bit before transferring the result to the multiplier result register (MR). This shift causes the multiplier result to be in 1.31 format, which can be rounded to 1.15 format. This result format appears in Figure 2-3 on page 2-12.

In the integer mode, the left shift does not occur. For example, if the operands are in the 16.0 format, the 32-bit multiplier result would be in 32.0 format. A left shift is not needed; it would change the numerical representation. This result format appears in Figure 2-4 on page 2-13.

Multiplier results generate status information. For more information on using multiplier status, see “Multiplier Status Flags” on page 2-31.

Shifter Data Types

Many operations in the shifter are explicitly geared to signed (two’s complement) or unsigned values: logical shifts assume unsigned-magnitude or binary string values, and arithmetic shifts assume two’s complement values.

The exponent logic assumes two’s complement numbers. The exponent logic supports block floating-point, which is also based on two’s complement fractions.

Shifter results generate status information. For more information on using shifter status, see “Shifter Status Flags” on page 2-50.

Arithmetic Formats Summary

Table 2-1, Table 2-2, and Table 2-3 summarize some of the arithmetic characteristics of computational operations.

Table 2-1. ALU Arithmetic Formats

Operation	Operands Formats	Result Formats
Addition	Signed or unsigned	Interpret flags
Subtraction	Signed or unsigned	Interpret flags
Logical Operations	Binary string	same as operands
Division	Explicitly signed/unsigned	same as operands
ALU Overflow	Signed	same as operands
ALU Carry Bit	16-bit unsigned	same as operands
ALU Saturation	Signed	same as operands

Table 2-2. Multiplier Arithmetic Formats

Operation (by Mode)	Operands Formats	Result Formats
Multiplier, Fractional Mode		
Multiplication (MR/SR)	1.15 Explicitly signed/unsigned	2.30 shifted to 1.31
Mult / Add	1.15 Explicitly signed/unsigned	2.30 shifted to 1.31
Mult / Subtract	1.15 Explicitly signed/unsigned	2.30 shifted to 1.31
Multiplier Saturation	Signed	same as operands

Table 2-2. Multiplier Arithmetic Formats (Cont'd)

Operation (by Mode)	Operands Formats	Result Formats
Multiplier, Integer Mode		
Multiplication (MR/SR)	16.0 Explicitly signed/unsigned	32.0 no shift
Mult / Add	16.0 Explicitly signed/unsigned	32.0 no shift
Mult / Subtract	16.0 Explicitly signed/unsigned	32.0 no shift
Multiplier Saturation	Signed	same as operands

Table 2-3. Shifter Arithmetic Formats

Operation	Operands Formats	Result Formats
Logical Shift	Unsigned / binary string	same as operands
Arithmetic Shift	Signed	same as operands
Exponent Detection	Signed	same as operands

Setting Computational Modes

The MSTAT and ICNTL registers control the operating mode of the computational units. Table A-6 on page A-11 lists all the bits in MSTAT, and Table A-11 on page A-20 lists all the bits in ICNTL. The following bits in MSTAT and ICNTL control computational modes:

- **ALU overflow latch mode.** MSTAT Bit 2 (AV_LATCH) determines how the ALU overflow flag, AV, gets cleared (0=AV is "not-sticky", 1=AV is "sticky").
- **ALU saturation mode.** MSTAT Bit 3 (AR_SAT) determines (for signed values) whether ALU AR results that overflowed or underflowed are saturated or not (0=unsaturated, 1=saturated).
- **Multiplier result mode.** MSTAT Bit 4 (M_MODE) selects fractional 1.15 format (=0) or integer 16.0 format (=1) for all multiplier operations. The multiplier adjusts the format of the result according to the selected mode.
- **Multiplier biased rounding mode.** ICNTL Bit 7 (BIASRND) selects unbiased (=0) or biased (=1) rounding for multiplier results.

Latching ALU Result Overflow Status

The DSP supports an ALU overflow latch mode with the AV_LATCH bit in the MSTAT register. This bit determines how the ALU overflow flag, AV, gets cleared.

If AV_LATCH is disabled (=0), the AV bit is "not-sticky". When an ALU overflow sets the AV bit in the ASTAT register, the AV bit only remains set until cleared by a subsequent ALU operation that does not generate an overflow (or is explicitly cleared).

If AV_LATCH is enabled (=1), the AV bit is "sticky". When an ALU overflow sets the AV bit in the ASTAT register, the AV bit remains set until the application explicitly clears it.

Saturating ALU Results on Overflow

The DSP supports an ALU saturation mode with the AR_SAT bit in the MSTAT register. This bit determines (for signed values) whether ALU AR results that overflowed or underflowed are saturated or not. This bit enables (if set, =1) or disables (if cleared, =0) saturation for all subsequent ALU operations. If AR_SAT is disabled, AR results remain unsaturated and is returned unchanged. If AR_SAT is enabled, AR results are saturated according to the state of the AV and AC status flags in ASTAT, as shown in Table 2-4.

Table 2-4. ALU Result Saturation With AR_SAT Enabled

AV	AC	AR register
0	0	ALU output not saturated
0	1	ALU output not saturated
1	0	ALU output saturated, maximum positive 0x7FFF
1	1	ALU output saturated, maximum negative 0x8000

The AR_SAT bit in MSTAT only affects the AR register. Only the results written to the AR register are saturated. If results are written to the AF register, wraparound occurs, but the AV and AC flags reflect the saturated result.

Using Multiplier Integer and Fractional Formats

For multiply/accumulate functions, the DSP provides two modes: fractional mode for fractional numbers (1.15), and integer mode for integers (16.0).

In the fractional mode, the 32-bit Product output is format adjusted—sign-extended and shifted one bit to the left—before being added to MR. For example, bit 31 of the Product lines up with bit 32 of MR (which is bit 0 of MR2) and bit 0 of the Product lines up with bit 1 of MR (which is bit 1 of MR0). The LSB is zero-filled. The fractional multiplier result format appears in Figure 2-3.

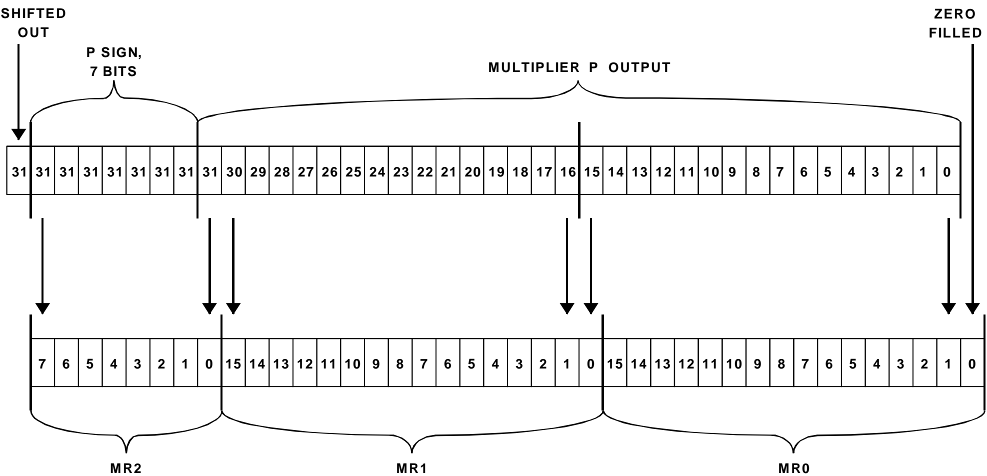


Figure 2-3. Fractional Multiplier Results Format

In the integer mode, the 32-bit Product register is not shifted before being added to MR. Figure 2-4 shows the integer-mode result placement.

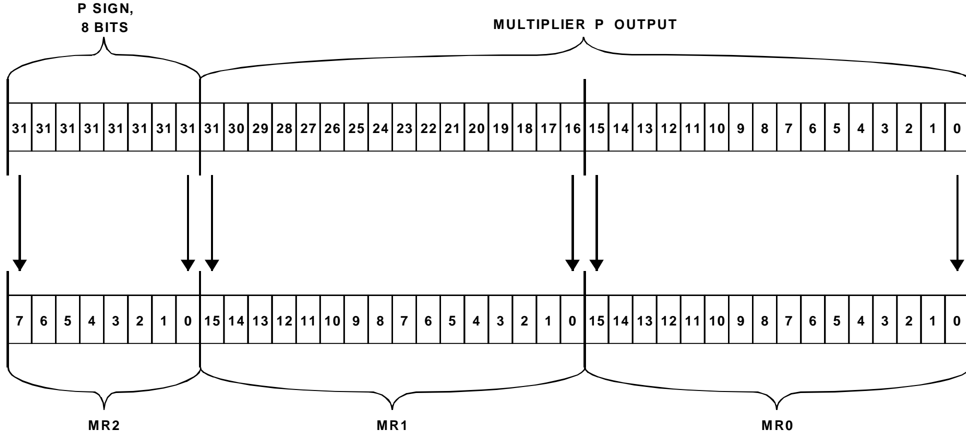


Figure 2-4. Integer Multiplier Results Format

The mode is selected by the M_MODE bit in the Mode Status (MSTAT) register. If M_MODE is set (=1), integer mode is selected. If M_MODE is cleared (=0), the fractional mode is selected. In either mode, the multiplier output Product is fed into a 40-bit adder/subtracter which adds or subtracts the new product with the current contents of the MR register to form the final 40-bit result.

Rounding Multiplier Results

The DSP supports multiplier results rounding (Rnd option) on most multiplier operations. With the Biasrnd bit in the ICNTL register, programs select whether the Rnd option provides biased or unbiased rounding.

Unbiased Rounding

Unbiased rounding uses the multiplier's capability for rounding the 40-bit result at the boundary between bit 15 and bit 16. Rounding can be specified as part of the instruction code. The rounded output is directed to either MR or SR. When rounding is selected, MR1/SR1 contains the rounded 16-bit result; the rounding effect in MR1/SR1 affects MR2/SR2 as well. The MR2/MR1 and SR2/SR1 registers represent the rounded 24-bit result.

The accumulator uses an unbiased rounding scheme. The conventional method of biased rounding is to add a 1 into bit position 15 of the adder chain. This method causes a net positive bias since the midway value (when MR0=0x8000) is always rounded upward. The accumulator eliminates this bias by forcing bit 16 in the result output to zero when it detects this midway point. This has the effect of rounding odd MR1 values upward and even MR1 values downward, yielding a zero large-sample bias assuming uniformly distributed values.

Using x to represent any bit pattern (not all zeros), here are two examples of rounding. The example in Figure 2-5 shows a typical rounding operation for MR; these also apply for SR.

```
                      ...MR2..|.......MR1......|.......MR0......
Unrounded value:  --> xxxxxxxx|xxxxxxxx00100101|1xxxxxxxxxxxxxxx
Add 1 and carry:  --> ........|................|1...............
Rounded value:    --> xxxxxxxx|xxxxxxxx00100110|0xxxxxxxxxxxxxxx
```

Figure 2-5. Typical Unbiased Multiplier Rounding Operation

The compensation to avoid net bias becomes visible when the lower 15 bits are all zero and bit 15 is one (the midpoint value) as shown in Figure 2-6.

```
                   |..MR2..MR1......|.......MR0......
Unrounded value:→ xxxxxxxx|xxxxxxxx01100110|1000000000000000
Add 1 and carry:→ ........|................|1...............
MR bit 16=1:    → xxxxxxxx|xxxxxxxx01100111|0000000000000000
Rounded value:  → xxxxxxxx|xxxxxxxx01100110|0000000000000000
```

Figure 2-6. Avoiding Net Bias in Unbiased Multiplier Rounding Operation

In Figure 2-6, MR bit 16 is forced to zero. This algorithm is employed on every rounding operation, but it is only evident when the bit patterns shown in the lower 16 bits of the last example are present.

Biased Rounding

The Biasrnd bit in the ICNTL register enables biased rounding. When the Biasrnd bit is cleared (=0), the Rnd option in multiplier instructions uses the normal unbiased rounding operation (as discussed in "Unbiased Rounding" on page 2-14). When the Biasrnd bit is set to 1, the DSP uses biased rounding instead of unbiased rounding. When operating in biased rounding mode, all rounding operations with MR0 set to 0x8000 round up, rather than only rounding odd MR1 values up. For an example, see Figure 2-7.

This mode only has an effect when the MR0 register contains 0x8000; all other rounding operations work normally. This mode allows more efficient implementation of bit-specified algorithms that use biased rounding, for example, the GSM speech compression routines. Unbiased rounding is preferred for most algorithms.

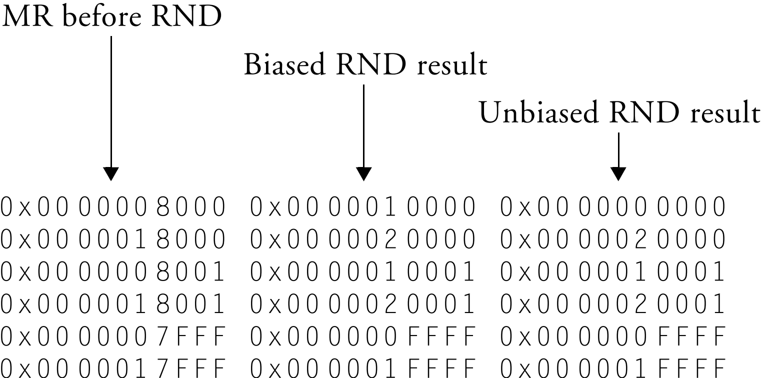


Figure 2-7. Bias Rounding in Multiplier Operation

Using Computational Status

The multiplier, ALU, and shifter update overflow and other status flags in the DSP's arithmetic status (ASTAT) register. To use status conditions from computations in program sequencing, use conditional instructions to test the exception flags in the ASTAT register after the instruction executes. This method permits monitoring each instruction's outcome.

More information on ASTAT appears in the sections that describe the computational units. For summaries relating instructions and status bits, see "ALU Status Flags" on page 2-18, "Multiplier Status Flags" on page 2-31, and "Shifter Status Flags" on page 2-50.

Arithmetic Logic Unit (ALU)

The ALU performs arithmetic and logical operations on fixed-point data. ALU fixed-point instructions operate on 16-bit fixed-point operands and output 16-bit fixed-point results. ALU instructions include:

- Fixed-point addition and subtraction
- Fixed-point add with carry, subtract with borrow, increment, decrement
- Logical AND, OR, XOR, NOT
- Functions: `Abs`, `Pass`, division primitives

ALU Operation

ALU instructions take one or two inputs: X input and Y input. For unconditional, single-function instructions, these inputs (also known as operands) can be any data registers in the register file. Most ALU operations return one result, but in `Pass` operations the ALU operation returns no result (only status flags are updated). ALU results are written to the ALU Result (`AR`) or ALU Feedback (`AF`) register.

The DSP transfers input operands from the register file during the first half of the cycle and transfers results to the result register during the second half of the cycle. With this arrangement, the ALU can read and write the `AR` register file location in a single cycle.

ALU Status Flags

ALU operations update status flags in the DSP's Arithmetic Status (ASTAT) register. Table A-5 on page A-9 lists all the bits in this register. Table 2-5 shows the bits in ASTAT that flag ALU status (a 1 indicates the condition is true) for the most recent ALU operation:

Table 2-5. ALU Status Bits in the ASTAT Register

Flag	Name	Definition
AZ	Zero	Logical NOR of all the bits in the ALU result register. True if ALU output equals zero.
AN	Negative	Sign bit of the ALU result. True if the ALU output is negative.
AV	Overflow	Exclusive-OR of the carry outputs of the two most significant adder stages. True if the ALU overflows.
AC	Carry	Carry output from the most significant adder stage.
AS	Sign	Sign bit of the ALU X input port. Affected only by the ABS instruction.
AQ	Quotient	Quotient bit generated only by the DIVS and DIVQ instructions.

Flag updates occur at the end of the cycle in which the status is generated and are available in the next cycle.

On previous 16-bit, fixed-point DSPs (ADSP-2100 family), the Pos (AS bit =1) and Neg (AS bit =0) conditions permit checking the ALU result's sign. On ADSP-219x DSPs, the CCODE register and SWCOND condition support this feature.

Unlike previous ADSP-218x DSPs, ASTAT writes on ADSP-219x DSPs have a one cycle effect latency. Code being ported from ADSP-218x to ADSP-219x DSPs that checks ALU status during the instruction following an ASTAT clear (ASTAT=0) instruction may not function as intended. Re-arranging the order of instructions to accommodate the one cycle effect latency on the ADSP-219x ASTAT register corrects this issue.

ALU Instruction Summary

Table 2-6 lists the ALU instructions and shows how they relate to ASTAT flags. As indicated in the table, the ALU handles flags in the same manner whether the result goes to the AR or AF registers. For more information on assembly language syntax, see the *ADSP-219x DSP Instruction Set Reference*. In Table 2-6, note the meaning of the following symbols:

- **Dreg, Dreg1, Dreg2** indicate any register file location
- **Xop, Yop** indicate any X- and Y-input registers, indicating a register usage restriction for conditional and/or multifunction instructions. For more information, see "Multifunction Computations" on page 2-60.
- * indicates the flag may be set or cleared, depending on the results of the instruction
- ** indicates the flag is cleared, regardless of the results of the instruction
- – indicates no effect

Table 2-6. ALU Instruction Summary

<table>
<tr><th rowspan="2">Instruction</th><th colspan="6">ASTAT Status Flags</th></tr>
<tr><th>AZ</th><th>AV</th><th>AN</th><th>AC</th><th>AS</th><th>AQ</th></tr>
<tr><td>|AR, AF| = Dreg1 + |Dreg2, Dreg2 + C, C |;</td><td>*</td><td>*</td><td>*</td><td>*</td><td>–</td><td>–</td></tr>
<tr><td>[IF Cond] |AR, AF| = Xop + |Yop, Yop + C, C, Const, Const + C|;</td><td>*</td><td>*</td><td>*</td><td>*</td><td>–</td><td>–</td></tr>
<tr><td>|AR, AF| = Dreg1 – |Dreg2, Dreg2 + C –1, +C –1|;</td><td>*</td><td>*</td><td>*</td><td>*</td><td>–</td><td>–</td></tr>
<tr><td>[IF Cond]|AR,AF| = Xop – |Yop,Yop+C–1,+C–1,Const,Const+C –1|;</td><td>*</td><td>*</td><td>*</td><td>*</td><td>–</td><td>–</td></tr>
<tr><td>|AR, AF| = Dreg2 – |Dreg1, Dreg1 + C –1|;</td><td>*</td><td>*</td><td>*</td><td>*</td><td>–</td><td>–</td></tr>
<tr><td>[IF Cond] |AR, AF| = Yop – |Xop, Xop+C–1|;</td><td>*</td><td>*</td><td>*</td><td>*</td><td>–</td><td>–</td></tr>
<tr><td>[IF Cond] |AR,AF| = – |Xop+C –1, Xop+Const, Xop+Const+C–1|;</td><td>*</td><td>*</td><td>*</td><td>*</td><td>–</td><td>–</td></tr>
<tr><td>|AR, AF| = Dreg1 |AND, OR, XOR| Dreg2;</td><td>*</td><td>**</td><td>*</td><td>**</td><td>–</td><td>–</td></tr>
<tr><td>[IF Cond] |AR, AF| = Xop |AND, OR, XOR| |Yop, Const|;</td><td>*</td><td>**</td><td>*</td><td>**</td><td>–</td><td>–</td></tr>
<tr><td>[IF Cond]|AR,AF| = |TSTBIT,SETBIT,CLRBIT,TGLBIT| n of Xop;</td><td>*</td><td>**</td><td>*</td><td>**</td><td>–</td><td>–</td></tr>
<tr><td>|AR, AF| = PASS |Dreg1, Dreg2, Const|;</td><td>*</td><td>**</td><td>*</td><td>**</td><td>–</td><td>–</td></tr>
<tr><td>|AR, AF| = PASS 0;</td><td>**</td><td>**</td><td>*</td><td>**</td><td>–</td><td>–</td></tr>
<tr><td>[IF Cond] |AR, AF| = PASS |Xop, Yop, Const|;</td><td>*</td><td>**</td><td>*</td><td>**</td><td>–</td><td>–</td></tr>
<tr><td>|AR, AF| = NOT |Dreg|;</td><td>*</td><td>**</td><td>*</td><td>**</td><td>–</td><td>–</td></tr>
<tr><td>[IF Cond] |AR, AF| = NOT |Xop, Yop|;</td><td>*</td><td>**</td><td>*</td><td>**</td><td>–</td><td>–</td></tr>
<tr><td>|AR, AF| = ABS Dreg;</td><td>*</td><td>**</td><td>**</td><td>**</td><td>*</td><td>–</td></tr>
<tr><td>[IF Cond] |AR, AF| = ABS Xop;</td><td>*</td><td>**</td><td>**</td><td>**</td><td>*</td><td>–</td></tr>
<tr><td>|AR, AF| = Dreg +1;</td><td>*</td><td>*</td><td>*</td><td>*</td><td>–</td><td>–</td></tr>
<tr><td>[IF Cond] |AR, AF| = Yop +1;</td><td>*</td><td>*</td><td>*</td><td>*</td><td>–</td><td>–</td></tr>
<tr><td>|AR, AF| = Dreg –1;</td><td>*</td><td>*</td><td>*</td><td>*</td><td>–</td><td>–</td></tr>
<tr><td>[IF Cond] |AR, AF| = Yop –1;</td><td>*</td><td>*</td><td>*</td><td>*</td><td>–</td><td>–</td></tr>
<tr><td>DIVS Yop, Xop;</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>*</td></tr>
<tr><td>DIVQ Xop;</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>*</td></tr>
</table>

ALU Data Flow Details

Figure 2-8 shows a detailed diagram of the ALU.

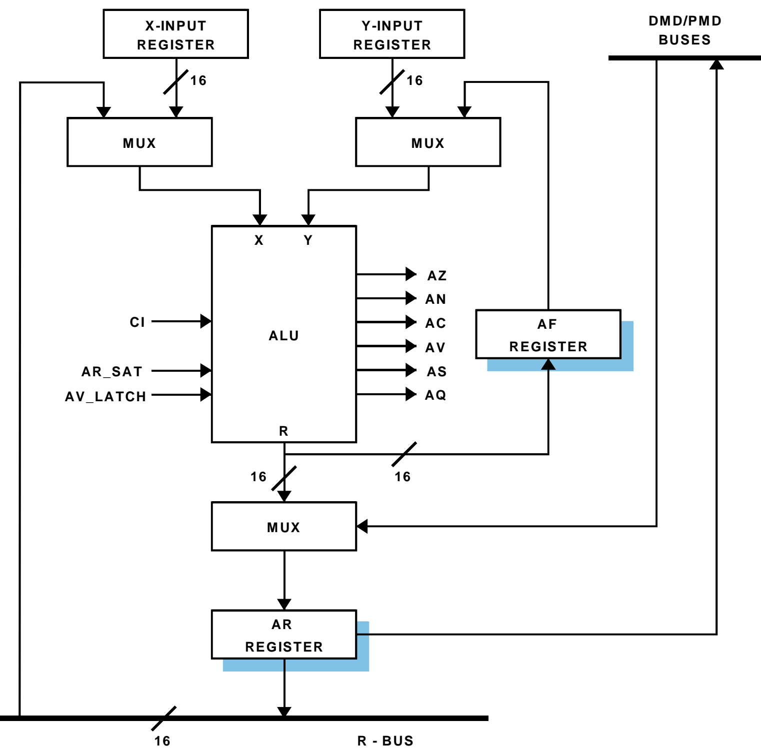


Figure 2-8. ALU Block Diagram

The ALU is 16 bits wide with two 16-bit input ports, X and Y, and one output port, R. The ALU accepts a carry-in signal (CI) which is the carry bit (AC) from the processor arithmetic status register (ASTAT). The ALU generates six status signals: the zero (AZ) status, the negative (AN) status, the carry (AC) status, the overflow (AV) status, the X-input sign (AS) status, and the quotient (AQ) status.

All arithmetic status signals are latched into the arithmetic status register (ASTAT) at the end of the cycle. For information on how each instruction affects the ALU flags, see Table 2-6 on page 2-20.

Depending on the instruction, the X input port of the ALU can accept data from two sources: the data register file (X-input registers for conditional/multifunction instructions) or the result (R) bus. The R bus connects the output registers of all the computational units, permitting them to be used as input operands directly.

Also depending on the instruction, the Y input port of the ALU can accept data from two sources: the data register file (Y-input registers for conditional/multifunction instructions) and the ALU feedback (AF) register.

For more information on register usage restrictions in conditional and multifunction instructions, see "Multifunction Computations" on page 2-60.

The output of the ALU goes into either the ALU feedback (AF) register or the ALU result (AR) register. The AF register is an ALU internal register, which lets the ALU result serve as the ALU Y input. The AR register can drive both the DMD bus and the R bus. It is also loadable directly from the DMD bus.

The ALU can read and write any of its associated registers in the same cycle. Registers are read at the beginning of the cycle and written at the end of the cycle. A register read gets the value loaded at the end of a previous cycle. A new value written to a register cannot be read out until a subsequent cycle. This read/write pattern lets an input register provide an operand to the ALU at the beginning of the cycle and be updated with the next operand from memory at the end of the same cycle. Also, this read/write pattern lets a result register be stored in memory and be updated with a new result in the same cycle.

The ALU contains a duplicate bank of registers (shown behind the primary registers in Figure 2-8 on page 2-21). There are two sets of data and results registers. Only one bank is accessible at a time. The additional bank of registers can be activated (such as during an interrupt service routine) for extremely fast context switching. A new task, such as an interrupt service routine, can be executed without transferring current states to storage. For more information, see "Secondary (Alternate) Data Registers" on page 2-59.

Multiprecision operations are supported in the ALU with the carry-in signal and ALU carry (AC) status bit. The carry-in signal is the AC status bit that was generated by a previous ALU operation. The "add with carry" (+C) operation is intended for adding the upper portions of multiprecision numbers. The "subtract with borrow" (C–1 is effectively a "borrow") operation is intended for subtracting the upper portions of multiprecision numbers.

ALU Division Support Features

The ALU supports division with two special divide primitives. These instructions (DIVS, DIVQ) let programs implement a non-restoring, conditional (error checking), add-subtract division algorithm. The division can be either signed or unsigned, but the dividend and divisor must both be of the same type. More details on using division and programming examples are available in the *ADSP-219x DSP Instruction Set Reference*.

A single-precision divide, with a 32-bit dividend (numerator) and a 16-bit divisor (denominator), yielding a 16-bit quotient, executes in 16 cycles. Higher and lower precision quotients can also be calculated. The divisor can be stored in AX0, AX1, or any of the R registers. The upper half of a signed dividend can start in either AY1 or AF. The upper half of an unsigned dividend must be in AF. The lower half of any dividend must be in AY0. At the end of the divide operation, the quotient is in AY0.

The first of the two primitive instructions "divide-sign" (DIVS) is executed at the beginning of the division when dividing signed numbers. This operation computes the sign bit of the quotient by performing an exclusive-OR of the sign bits of the divisor and the dividend. The AY0 register is shifted one place so that the computed sign bit is moved into the LSB position. The computed sign bit is also loaded into the AQ bit of the arithmetic status register. The MSB of AY0 shifts into the LSB position of AF, and the upper 15 bits of AF are loaded with the lower 15 R bits from the ALU, which simply passes the Y input value straight through to the R output. The net effect is to left shift the AF-AY0 register pair and move the quotient sign bit into the LSB position. The operation of DIVS is illustrated in Figure 2-9.

When dividing unsigned numbers, the DIVS operation is not used. Instead, the AQ bit in the arithmetic status register (ASTAT) should be initialized to zero by manually clearing it. The AQ bit indicates to the following operations that the quotient should be assumed positive.

The second division primitive is the "divide-quotient" (DIVQ) instruction, which generates one bit of quotient at a time and is executed repeatedly to compute the remaining quotient bits.

For unsigned single precision divides, the DIVQ instruction is executed 16 times to produce 16 quotient bits. For signed single precision divides, the DIVQ instruction is executed 15 times after the sign bit is computed by the DIVS operation. DIVQ instruction shifts the AY0 register left by one bit so that the new quotient bit can be moved into the LSB position.

The status of the AQ bit generated from the previous operation determines the ALU operation to calculate the partial remainder. If AQ = 1, the ALU adds the divisor to the partial remainder in AF. If AQ = 0, the ALU subtracts the divisor from the partial remainder in AF.

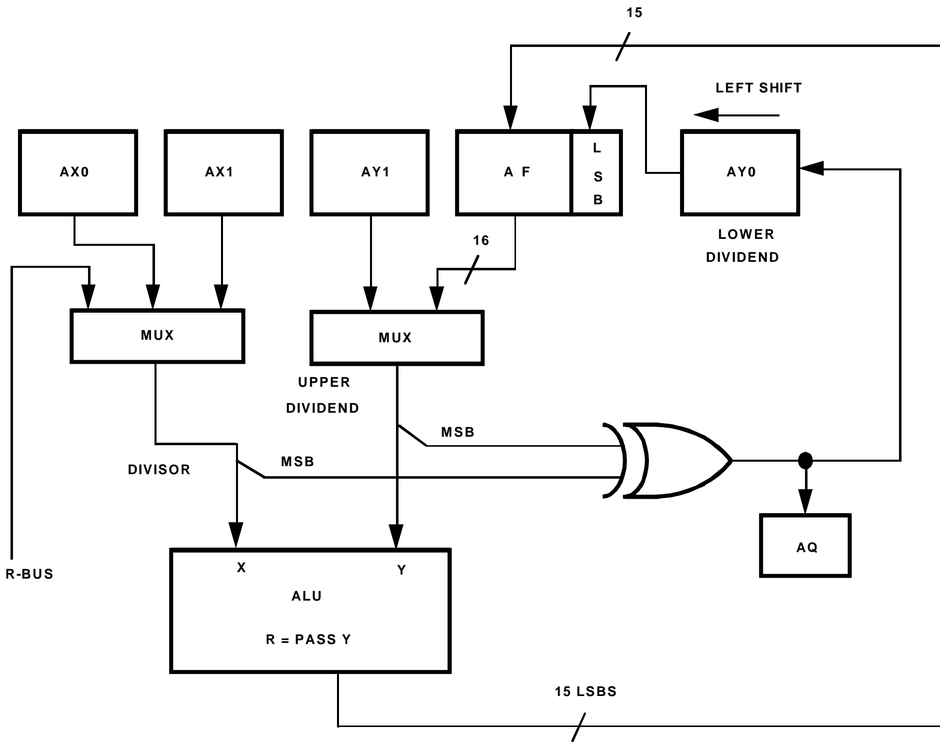


Figure 2-9. DIVS Operation

The ALU output R is offset loaded into AF just as with the DIVS operation. The AQ bit is computed as the exclusive-OR of the divisor MSB and the ALU output MSB, and the quotient bit is this value inverted. The quotient bit is loaded into the LSB of the AY0 register which is also shifted left by one bit. The DIVQ operation is illustrated in Figure 2-10.

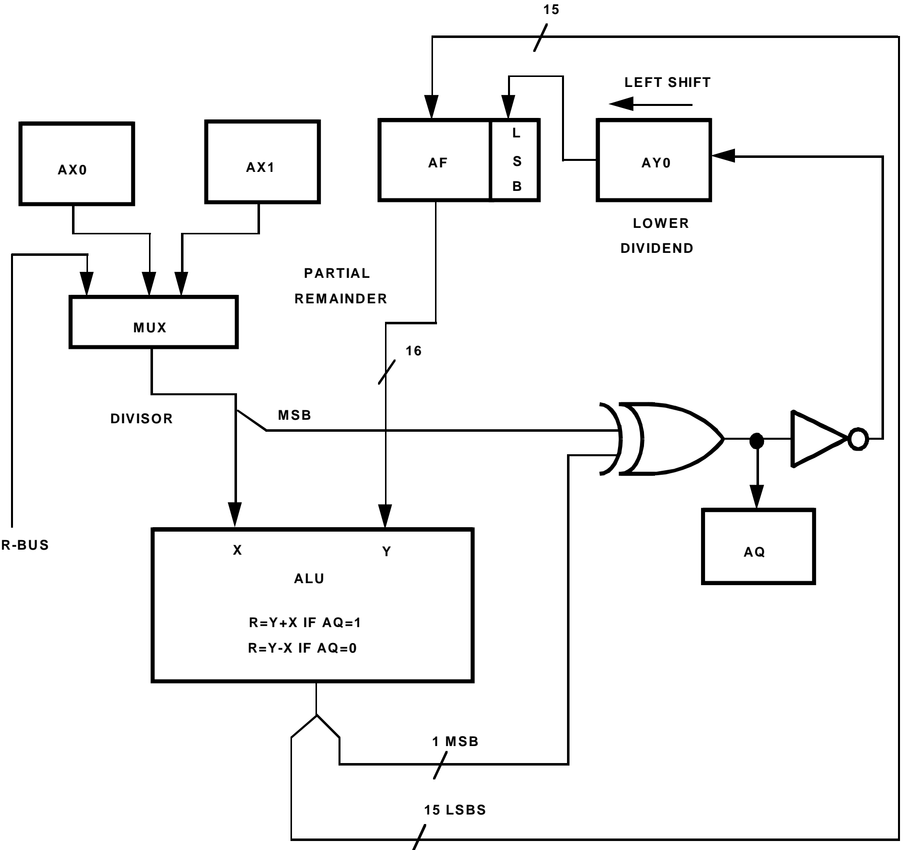


Figure 2-10. DIVQ Operation

The format of the quotient for any numeric representation can be determined by the format of the dividend and divisor as shown in Figure 2-11. Let NL represent the number of bits to the left of the binary point, let NR represent the number of bits to the right of the binary point of the dividend, let DL represent the number of bits to the left of the binary point, and let DR represent the number of bits to the right of the binary point of the divisor. Then, the quotient has NL–DL+1 bits to the left of the binary point and has NR–DR–1 bits to the right of the binary point.

```
Dividend          BBBBB .BBBBBBBBBBBBBBBBBBBBBBBBBBB
                      NL bits        NR bits

Divisor              BB .BBBBBBBBBBBBBB
                      DL bits        DR bits

Quotient           BBBB .BBBBBBBBBBBB
              (NL–DL+1) bits         (NR–DR–1) bits
```

Figure 2-11. Quotient Format

Some format manipulation may be necessary to guarantee the validity of the quotient. For example, if both operands are signed and fully fractional (dividend in 1.31 format and divisor in 1.15 format) the result is fully fractional (in 1.15 format), and the dividend must be smaller than the divisor for a valid result.

To divide two integers (dividend in 32.0 format and divisor in 16.0 format) and produce an integer quotient (in 16.0 format), the program must shift the dividend one bit to the left (into 31.1 format) before dividing. Additional discussion and code examples can be found in the *ADSP-219x DSP Instruction Set Reference.*

The algorithm overflows if the result cannot be represented in the format of the quotient as shown in the calculation in Figure 2-11 on page 2-27, or when the divisor is zero or less than the dividend in magnitude.

Multiply—Accumulator (Multiplier)

The multiplier performs fixed-point multiplication and multiply/accumulate operations. Multiply/accumulates are available with either cumulative addition or cumulative subtraction. Multiplier fixed-point instructions operate on 16-bit fixed-point data and produce 40-bit results. Inputs are treated as fractional or integer, unsigned or two's complement. Multiplier instructions include:

- Multiplication
- Multiply/accumulate with addition, rounding optional
- Multiply/accumulate with subtraction, rounding optional
- Rounding, saturating, or clearing result register

Multiplier Operation

The multiplier takes two inputs: X input and Y input. For unconditional, single-function instructions, these inputs (also known as operands) can be any data registers in the register file. The multiplier accumulates results in either the Multiplier Result (MR) or Shifter Result (SR) register. The results can also be rounded or saturated.

On previous 16-bit, fixed-point DSPs (ADSP-2100 family), only the multiplier results (MR) register can accumulate results for the multiplier. On ADSP-219x DSPs, both MR and SR registers can accumulate multiplier results.

The multiplier transfers input operands during the first half of the cycle and transfers results during the second half of the cycle. With this arrangement, the multiplier can read and write the same result register in a single cycle.

Depending on the multiplier mode (M_MODE) setting, operands are either both in integer format or both in fractional format. The format of the result matches the format of the inputs. Each operand may be either an unsigned or a two's complement value. If both inputs are fractional and signed, the multiplier automatically shifts the result left one bit to remove the redundant sign bit. Multiplier instruction options (required within the multiplier instruction) specify inputs' data format(s)—SS for signed, UU for unsigned, SU for signed X-input and unsigned Y-input, and US for unsigned X-input and signed Y-input.

Placing Multiplier Results in MR or SR Registers

As shown in Figure 2-12, the MR register is divided into three sections: MR0 (bits 0-15), MR1 (bits 16-31), and MR2 (bits 32-39). Similarly, the SR register is divided into three sections: SR0 (bits 0-15), SR1 (bits 16-31), and SR2 (bits 32-39). Each of these registers can be loaded from the DMD bus and output to the R bus or the DMD bus.

When the multiplier writes to either of the result registers, the 40-bit result goes into the lower 40 bits of the combined register (MR2, MR1, and MR0 or SR2, SR1, and SR0), and the MSB is sign extended into the upper eight bits of the uppermost register (MR2 or SR2). When an instruction explicitly loads the middle result register (MR1 or SR1), the DSP also sign extends the MSB of the data into the related uppermost register (MR2 or SR2). These sign extension operations appear in Figure 2-12.

To load the MR2 register with a value other than MR1's sign extension, programs must load MR2 after MR1 has been loaded. Loading MR0 affects neither MR1 nor MR2; no sign extension occurs in MR0 loads. This technique also applies to SR2, SR1, and SR0.

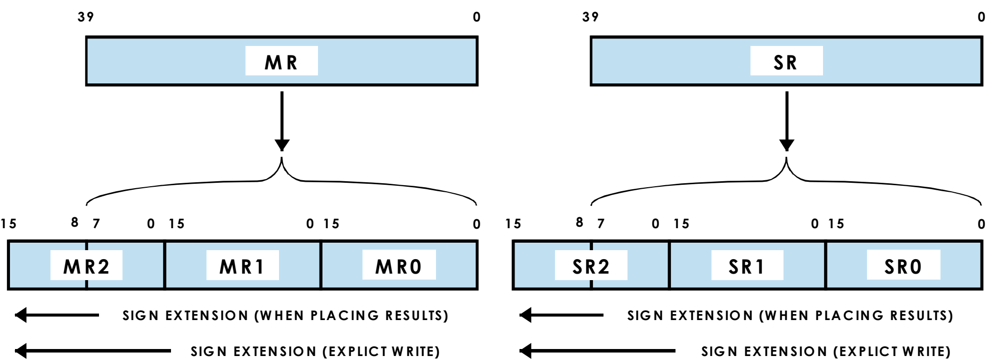


Figure 2-12. Placing Multiplier Results

Clearing, Rounding, or Saturating Multiplier Results

Besides using the results registers to accumulate, the multiplier also can clear, round, or saturate result data in the results registers. These operations work as follows:

- The clear operation—`[MR,SR]=0`—clears the specified result register to zero.
- The rounding operation—`[MR,SR]=Rnd [MR,SR]`—applies only to fractional results—integer results are not affected. This explicit rounding operation generates the same results as using the `Rnd` option in other multiplier instructions. For more information, see "Rounding Multiplier Results" on page 2-14.
- The saturate operation—`Sat [MR,SR]`—sets the specified result register to the maximum positive or negative value if an overflow or underflow has occurred. The saturation operation depends on the overflow status bit (`MV` or `SV`) and the MSB of the corresponding result register (`MR2` or `SR2`). For more information, see "Saturating Multiplier Results on Overflow" on page 2-31.

Multiplier Status Flags

Multiplier operations update two status flags in the computational unit's arithmetic status register (ASTAT). Table A-5 on page A-9 lists all the bits in these registers. The following bits in ASTAT flag multiplier status (a 1 indicates the condition) for the most recent multiplier operation:

- **Multiplier overflow.** Bit 6 (MV) records overflow/underflow condition for MR result register. If cleared (=0), no overflow or underflow has occurred. If set (=1), an overflow or underflow has occurred.
- **Shifter overflow.** Bit 8 (SV) records overflow/underflow condition for SR result register. If cleared (=0) no overflow or underflow has occurred. If set (=1), an overflow or underflow has occurred.

Flag updates occur at the end of the cycle in which the status is generated and are available on the next cycle.

Saturating Multiplier Results on Overflow

The adder/subtracter generates an overflow status signal every time a multiplier operation is executed. When the accumulator result in MR or SR, interpreted as a two's complement number, crosses the 32-bit (MR1/MR2) boundary (overflows), the multiplier sets the MV or SV bit in the ASTAT register.

The multiplier saturation instruction provides control over a multiplication result that has overflowed or underflowed. It saturates the value in the specified register only for the cycle in which it executes. It does not enable a mode that continuously saturates results until disabled, like the ALU. Used at the end of a series of multiply and accumulate operations, the saturation instruction prevents the accumulator from overflowing.

For every operation it performs, the multiplier generates an overflow status signal MV (SV when SR is the specified result register), which is recorded in the ASTAT status register. The multiplier sets MV = 1 when the upper

nine bits in MR are anything other than all 0s or all 1s, setting MV when the accumulator result—interpreted as a signed, two's complement number—crosses the 32-bit boundary and spills over from MR1 into MR2. Otherwise, the multiplier clears MV = 0.

The operation of the saturation instruction depends on the overflow status bit MV (or SV) and the MSB of the result, which appear in Table 2-7 on page 2-32. If MV/SV = 0, no saturation occurs. When MV/SV = 1, the multiplier examines the MSB of MR2 to determine whether the result has overflowed or underflowed. If the MSB = 0, the result has overflowed, and the multiplier saturates the result register, setting it to the maximum positive value. If the MSB = 1, the result has underflowed, and the multiplier saturates the MR register, setting it to the maximum negative value.

Table 2-7. Saturation Status Bits and Result Registers

MV/SV	MSB of MR2/SR2	MR/SR Results
0	0	No change.
0	1	No change.
1	0	00000000 0111111111111111 1111111111111111
1	1	11111111 1000000000000000 0000000000000000

Avoid result overflows beyond the MSB of the result register. In such a case, the true sign bit of the result is irretrievably lost, and saturation may not produce a correct result. It takes over 255 overflows to lose the sign.

Multiplier Instruction Summary

Table 2-8 lists the multiplier instructions and how they relate to ASTAT flags. For more information on assembly language syntax, see the *ADSP-219x DSP Instruction Set Reference*. In Table 2-8, note the meaning of the following symbols:

- **Dreg1, Dreg2** indicate any register file location
- **Xop, Yop** indicate any X- and Y-input registers, indicating a register usage restriction for conditional and/or multifunction instructions. For more information, see "Multifunction Computations" on page 2-60.
- * indicates the flag may be set or cleared, depending on results of instruction
- ** indicates the flag is cleared, regardless of the results of instruction
- – indicates no effect

Table 2-8. Multiplier Instruction Summary

Instruction	ASTAT Status Flags	
	MV	SV
|MR, SR| = Dreg1 * Dreg2 [(|RND, SS, SU, US, UU|)];	*	*
[IF Cond] |MR, SR| = Xop * Yop [(|RND, SS, SU, US, UU|)];	*	*
[IF Cond] |MR, SR| = Yop * Xop [(|RND, SS, SU, US, UU|)];	*	*
|MR, SR| = |MR, SR| + Dreg1 * Dreg2 [(|RND, SS, SU, US, UU|)];	*	*
[IF Cond]|MR, SR| = |MR, SR| + Xop * Yop [(|RND, SS, SU, US, UU|)];	*	*
[IF Cond] |MR, SR| = |MR, SR| + Yop * Xop [(|RND, SS, SU, US, UU|)];	*	*
|MR, SR| = |MR, SR| – Dreg1 * Dreg2 [(|RND, SS, SU, US, UU|)];	*	*
[IF Cond] |MR, SR| = |MR, SR| – Xop * Yop [(|RND, SS, SU, US, UU|)];	*	*
[IF Cond] |MR, SR| = |MR, SR| – Yop * Xop [(|RND, SS, SU, US, UU|)];	*	*

Table 2-8. Multiplier Instruction Summary (Cont'd)

Instruction	ASTAT Status Flags	
	MV	SV
[IF Cond] |MR, SR| = 0;	**	**
[IF Cond] MR = MR [(RND)];	*	–
[IF Cond] SR = SR [(RND)];	–	*
SAT [MR,SR];	–	–

Multiplier Data Flow Details

Figure 2-13 on page 2-35 shows a detailed diagram of the multiplier/accumulator.

The multiplier has two 16-bit input ports, X and Y, and a 32-bit product output port, Product. The 32-bit product is passed to a 40-bit adder/subtracter, which adds or subtracts the new product from the content of the multiplier result (MR or SR) register or passes the new product directly to the results register. For results, the MR and SR registers are 40 bits wide. These registers each consist of smaller 16-bit registers: MR0, MR1, MR2, SR0, SR1, and SR2. For more information on these registers, see Figure 2-12 on page 2-30.

The adder/subtracters are greater than 32 bits to allow for intermediate overflow in a series of multiply/accumulate operations. A multiply overflow (MV or SV) status bit is set when an accumulator has overflowed beyond the 32-bit boundary—when there are significant (non-sign) bits in the top nine bits of the MR or SR registers (based on two's complement arithmetic).

Depending on the instruction, the X input port of the multiplier can accept data from two sources: the data register file (X-input registers for conditional/multifunction instructions) or the result (R) bus. The R bus connects the output registers of all the computational units, permitting them to be used as input operands directly.

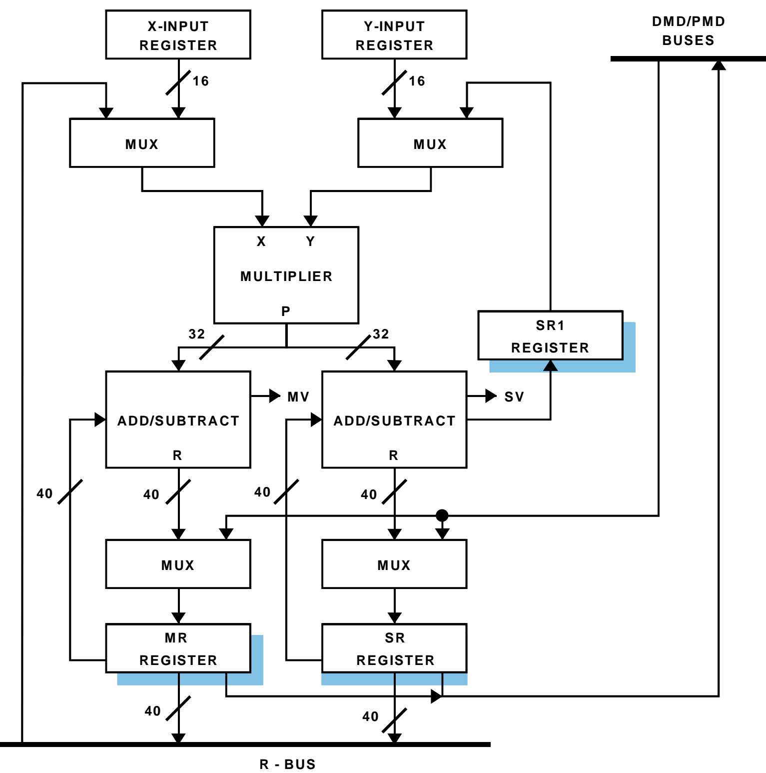


Figure 2-13. Multiplier Block Diagram

On previous 16-bit, fixed-point DSPs (ADSP-2100 family), only the dedicated input registers for each computational unit can provide inputs. On ADSP-219x DSPs, any register in the data register file can provide input to any computational unit.

Depending on the instruction, the Y input port of the multiplier can also accept data from two sources: the data register file (Y-input registers for conditional/multifunction instructions) and the multiplier feedback (SR1) register.

On previous 16-bit, fixed-point DSPs (ADSP-2100 family), a dedicated multiplier feedback (MF) register is available. On ADSP-219x DSPs, there is no MF register; therefore, code should use SR1.

For more information on register usage restrictions in conditional and multifunction instructions, see "Multifunction Computations" on page 2-60.

The output of an adder/subtracter goes to the feedback (SR) register or a results (MR or SR) register. The SR1 register is a feedback register which allows bits 16–31 of the result to be used directly as the multiplier Y input on a subsequent cycle.

The multiplier reads and writes any of its associated registers within the same cycle. Registers are read at the beginning of the cycle and written at the end of the cycle. A register read gets the value loaded at the end of a previous cycle. A new value written to a register cannot be read out until a subsequent cycle. This read/write pattern lets an input register provide an operand to the multiplier at the beginning of the cycle and be updated with the next operand from memory at the end of the same cycle. This pattern also lets a result register be stored in memory and updated with a new result in the same cycle.

The multiplier contains a duplicate bank of registers (shown behind the primary registers in Figure 2-13 on page 2-35). There are two sets of data and results registers. Only one bank is accessible at a time. The additional bank of registers can be activated (for example, during an interrupt service routine) for extremely fast context switching. A new task, such as an interrupt service routine, can be executed without transferring current states to storage. For more information, see "Secondary (Alternate) Data Registers" on page 2-59.

Barrel-Shifter (Shifter)

The shifter provides bitwise shifting functions for 16-bit inputs, yielding a 40-bit output. These functions include arithmetic shift (ASHIFT), logical shift (LSHIFT), and normalization (NORM). The shifter also performs derivation of exponent (EXP) and derivation of common exponent (EXPADJ) for an entire block of numbers. These shift functions can be combined to implement numerical format control, including full floating-point representation.

Shifter Operations

The shifter instructions (ASHIFT, LSHIFT, NORM, EXP, and EXPADJ) can be used in a variety of ways, depending on the underlying arithmetic requirements. The following sections present single and multiple precision examples for these functions:

- "Derive Block Exponent" on page 2-39
- "Immediate Shifts" on page 2-40
- "Denormalize" on page 2-42
- "Normalize, Single Precision Input" on page 2-44

The shift functions (arithmetic shift, logical shift, and normalize) can be optionally specified with [SR OR] to facilitate multiprecision operations. [SR OR] logically OR's the shift result with the current contents of SR. This option is used to join 16-bit inputs with the 40-bit value in SR. When [SR OR] is not used, the shift value is passed through to SR directly.

Almost all shifter instructions have two or three options: (Hi), (Lo), and (Hix). Each option enables a different exponent detector mode that operates only while the instruction executes. The shifter interprets and handles the input data according to the selected mode.

For the derive exponent (`EXP`) and block exponent adjust (`EXPADJ`) operations, the shifter calculates the shift code—the direction and number of bits to shift—then stores the value in `SE` (for `EXP`) or `SB` (for `EXPADJ`). For the `ASHIFT`, `LSHIFT`, and `NORM` operations, a program can supply the value of the shift code directly to the `SE` register or use the result of a previous `EXP` or `EXPADJ` operation.

For the `ASHIFT`, `LSHIFT`, and `NORM` operations:

`(Hi)` Operation references the upper half of the output field.

`(Lo)` Operation references the lower half of the output field.

For the exponent derive (`EXP`) operation:

`(Hix)` Mode used for shifts and normalization of results from ALU operations.

Input data is the result of an add or subtract operation that may have overflowed. The shifter examines the ALU overflow bit `AV`. If `AV=1`, the effective exponent of the input is `+1` (this value indicates that overflowed occurred before the `EXP` operation executed). If `AV=0`, no overflow occurred and the shifter performs the same operations as the (`Hi`) mode.

`(Hi)` Input data is a single-precision signed number or the upper half of a double-precision signed number. The number of leading sign bits in the input operand, which equals the number of sign bits minus one, determines the shift code.

(By default, the `EXPADJ` operation always operates in this mode.)

`(Lo)` Input data is the lower half of a double-precision signed number. To derive the exponent on a double-precision number, the program must perform the `EXP` operation twice, once on the upper half of the input, and once on the lower half.

Derive Block Exponent

The `EXPADJ` instruction detects the exponent of the number largest in magnitude in an array of numbers. The steps for a typical block exponent derivation are as follows:

1. **Load SB with –16.** The `SB` register contains the exponent for the entire block. The possible values at the conclusion of a series of `EXPADJ` operations range from –15 to 0. The exponent compare logic updates the `SB` register if the new value is greater than the current value. Loading the register with –16 initializes it to a value certain to be less than any actual exponents detected.

2. **Process the first array element** as follows:

Array(1) =	`11110101 10110001`
Exponent =	–3
	– 3 > `SB` (–16)
`SB` gets	–3

3. **Process next array element** as follows:

Array(2)=	`00000001 01110110`
Exponent =	–6
	–6 < –3
`SB` remains	–3

4. **Continue processing array elements.**

When and if an array element is found whose exponent is greater than `SB`, that value is loaded into `SB`. When all array elements have been processed, the `SB` register contains the exponent of the largest number in the entire block. No normalization is performed. `EXPADJ` is purely an inspection operation. The value in `SB` could be transferred to `SE` and used to normalize the block on the next pass through the shifter. Or, `SB` could be associated with that data for subsequent interpretation.

Immediate Shifts

An immediate shift shifts the input bit pattern to the right (downshift) or left (upshift) by a given number of bits. Immediate shift instructions use the data value in the instruction itself to control the amount and direction of the shifting operation. For examples using this instruction, see the *ADSP-219x DSP Instruction Set Reference*. The data value controlling the shift is an 8-bit signed number. The SE register is not used or changed by an immediate shift.

The following example shows the input value downshifted relative to the upper half of SR (SR1). This is the (Hi) version of the shift:

```
SI = 0xB6A3;
SR = LSHIFT SI By -5 (Hi);
```

Input (SI): 1011 0110 1010 0011
Shift value: –5

SR (shifted by):

```
0000 0000 0000 0101 1011 0101 0001 1000 0000 0000
---sr2---|--------sr1--------|--------sr0--------
```

This next example uses the same input value, but shifts in the other direction, referenced to the lower half (Lo) of SR:

```
SI = 0xB6A3;
SR = LSHIFT SI By 5 (LO);
```

Input (SI): 1011 0110 1010 0011
Shift value: +5

SR (shifted by):

```
0000 0000 0000 0000 0001 0110 1101 0100 0110 0000
---sr2---|--------sr1--------|--------sr0--------
```

Note that a negative shift cannot place data (except a sign extension) into SR2, but a positive shift with value greater than 16 puts data into SR2. This next example also sets the SV bit (because the MSB of SR1 does not match the value in SR2):

```
SI = 0xB6A3; SR = LSHIFT SI By 17 (Lo);
```

Input (SI): 1011 0110 1010 0011
Shift value: +17

SR (shifted by):

```
0000 0001 0110 1101 0100 0110 0000 0000 0000 0000
---sr2---|--------sr1--------|--------sr0--------
```

In addition to the direction of the shifting operation, the shift may be either arithmetic (ASHIFT) or logical (LSHIFT). For example, the following shows a logical shift, relative to the upper half of SR (Hi):

```
SI = 0xB6A3;
SR = LSHIFT SI By -5 (HI);
```

Input (SI): 10110110 10100011
Shift value: -5

SR (shifted by):

```
0000 0000 0000 0101 1011 0101 0001 1000 0000 0000
---sr2---|--------sr1--------|--------sr0--------
```

This next example uses the same input value, but performs an arithmetic shift:

```
SI = 0xB6A3;
SR = ASHIFT SI By -5 (HI);
```

Input (SI): 10110110 10100011
Shift value: -5

SR (shifted by):

```
    1111 1111 1111 1101 1011 0101 0001 1000 0000 0000
    ---sr2---|--------sr1--------|--------sr0--------
```

Denormalize

Denormalizing refers to shifting a number according to a predefined exponent. The operation is effectively a floating-point to fixed-point conversion.

Denormalizing requires a sequence of operations. First, the SE register must contain the exponent value. This value may be explicitly loaded or may be the result of some previous operation. Next, the shift itself is performed, taking its shift value from the SE register, not from an immediate data value.

Two examples of denormalizing a double-precision number follow. The first example shows a denormalization in which the upper half of the number is shifted first, followed by the lower half. Because computations may produce output in either order, the second example shows the same operation in the other order—lower half first.

This first denormalization example processes the upper half first. Some important points here are: (1) always select the arithmetic shift for the higher half (Hi) of the two's complement input (or logical for unsigned), and (2) the first half processed does not use the [SR OR] option.

```
SI = 0xB6A3;              {first input, upper half result}
SE = -3;                  {shifter exponent}
SR = ASHIFT SI By -3 (HI); {must use HI option}
```

First input (SI): `1011011010100011`

SR (shifted by):

```
1111 1111 1111 0110 1101 0100 0110 0000 0000 0000
---sr2---|--------sr1--------|--------sr0--------
```

Continuing this example, next, the lower half is processed. Some important points here are: (1) always select a logical shift for the lower half of the input, and (2) the second half processed must use the [SR OR] option to avoid overwriting the previous half of the output value.

```
SI = 0x765D;              {second input, lower half result}
                          {SE = -3 still}
SR = SR OR LSHIFT SI By -3 (Lo); {must use Lo option}
```

Second input (SI): `0111 0110 0101 1101`

SR (OR'd, shifted):

```
1111 1111 1111 0110 1101 0100 0110 1110 1100 1011
---sr2---|--------sr1--------|--------sr0--------
```

This second denormalization example uses the same input, but processes it in the opposite (lower half first) order. The same important points from before apply: (1) the high half is always arithmetically shifted, (2) the low half is logically shifted, (3) the first input is passed straight through to SR, and (4) the second half is OR'ed, creating a double-precision value in SR.

```
SI = 0x765D;              {first input, lower half result}
SE = -3;                  {shifter exponent}
SR = LSHIFT SI By -3 (LO); {must use LO option}
SI = 0xB6A3;              {second input, upper half result}
```

```
SR = SR OR ASHIFT SI By -3 (Hi); {must use Hi option}
```

First input (SI): 0111 0110 0101 1101

SR (shifted by):

```
0000 0000 0000 0000 0000 0000 0000 1110 1100 1011
---sr2---|--------sr1--------|--------sr0--------
```

Second input (SI): 1011 0110 1010 0011

SR (OR'd, shifted):

```
1111 1111 1111 0110 1101 0100 0110 1110 1100 1011
---sr2---|--------sr1--------|--------sr0--------
```

Normalize, Single Precision Input

Numbers with redundant sign bits require normalizing. Normalizing a number is the process of shifting a two's complement number within a field so that the right-most sign bit lines up with the MSB position of the field and recording how many places the number was shifted. This operation can be thought of as a fixed-point to floating-point conversion, generating an exponent and a mantissa.

Normalizing is a two-stage process. The first stage derives the exponent. The second stage does the actual shifting. The first stage uses the EXP instruction which detects the exponent value and loads it into the SE register. The EXP instruction recognizes a (Hi) and (Lo) modifier. The second stage uses the NORM instruction. NORM recognizes (Hi) and (Lo) and also has the [SR OR] option. NORM uses the negated value of the SE register as its shift control code. The negated value is used so that the shift is made in the correct direction.

This normalization example for a single precision input. First, the EXP instruction derives the exponent:

```
AR = 0xF6D4; {single precision input}
SE = EXP AR (Hi);  {Detects Exponent With Hi Modifier}
```

Input (AR): 1111 0110 1101 0100
Exponent (SE): –3

Next for this single precision example, the NORM instruction normalizes the input using the derived exponent in SE:

```
SR = NORM AR (Hi);
```

Input (AR): 1111 0110 1101 0100

SR (Normalized):

```
1111 1111 1011 0110 1010 0000 0000 0000 0000 0000
---sr2---|--------sr1--------|--------sr0--------
```

For a single precision input, the normalize operation can use either the (Hi) or (Lo) modifier, depending on whether the result is needed in SR1 or SR0.

Normalize, ALU Result Overflow

For single precision data, there is a special normalization situation—normalizing ALU results (AR) that may have overflowed—that requires the Hi-extended (Hix) modifier. When using this modifier, the shifter reads the arithmetic status word (ASTAT) overflow bit (AV) and the carry bit (AC) in conjunction with the value in AR. If AV is set (=1), an overflow has occurred. AC contains the true sign of the two's complement value.

Given the following conditions, the normalize operation would be as follows:

```
AR =          1111 1010 0011 0010
AV =          1  (indicating overflow)
AC =          0  (the true sign bit of this value)
```

```
SE=EXP AR (HIX); SR=NORM AR (HI);
```

1. Detect Exponent, Modifier = Hix

   SE gets set to: +1

2. Normalize, Modifier = Hi, SE = 1

   ```
   AR =          1111 1010 0011 0010
   ```

   SR (Normalized):

   ```
      0000 0000 0111 1101 0001 1001 0000 0000 0000 0000
      ---sr2---|--------sr1--------|--------sr0--------
   ```

The AC bit is supplied as the sign bit, MSB of SR above.

The NORM instruction differs slightly between the ADSP-219x and previous 16-bit, fixed-point DSPs in the ADSP-2100 family. The difference can only be seen when performing overflow normalization.

- On the ADSP-219x, the NORM instruction checks only that (SE == +1) for performing the shift in of the AC flag (overflow normalization).
- On previous ADSP-2100 family DSP's, the NORM instruction checks both that (SE == +1) and (AV == 1) before shifting in the AC flag.

The EXP (HIX) instruction always sets (SE = +1) when the AV flag is set, so this execution difference only appears when NORM is used without a preceding EXP instruction.

The Hix operation executes properly whether or not there has actually been an overflow, as demonstrated by this second example:

```
AR  =        1110 0011 0101 1011
AV  =        0 (indicating no overflow)
AC  =        0 (not meaningful if AV = 0)
```

1. Detect Exponent, Modifier = Hix

   SE set to   –2

2. Normalize, Modifier = Hi, SE = –2

   ```
   AR =        1110 0011 0101 1011
   ```

   SR (Normalized):

   ```
   1111 1111 1000 1101 0110 1000 0000 0000 0000 0000
   ---sr2---|--------sr1--------|--------sr0--------
   ```

The AC bit is not used as the sign bit. As Figure 2-15 shows, the Hix mode is identical to the Hi mode when AV is not set. When the NORM, Lo operation is done, the extension bit is zero; when the NORM, Hi operation is done, the extension bit is AC.

Normalize, Double Precision Input

For double precision values, the normalization process follows the same general scheme as with single precision values. The first stage detects the exponent and the second stage normalizes the two halves of the input. For normalizing double precision values, there are two operations in each stage.

For the first stage, the upper half of the input must be operated on first. This first exponent derivation loads the exponent value into SE. The second exponent derivation, operating on the lower half of the number does not alter the SE register unless SE = –15. This happens only when the first half contained all sign bits. In this case, the second operation loads a value

into SE (see Figure 2-16). This value is used to control both parts of the normalization that follows.

For the second stage (SE now contains the correct exponent value), the order of operations is immaterial. The first half (whether Hi or Lo) is normalized without the [SR OR], and the second half is normalized with [SR OR] to create one double-precision value in SR. The (Hi) and (Lo) modifiers identify which half is being processed.

The following example normalizes double precision values:

1. Detect Exponent, Modifier = Hi

   First Input: 1111 0110 1101 0100 (upper half)
   SE set to: -3

2. Detect Exponent, Modifier = Lo

   Second Input: 0110 1110 1100 1011
   SE unchanged: -3

   Normalize, Modifier=Hi, No [SR OR], SE = –3

   First Input: 1111 0110 1101 0100

   SR (Normalized):

   ```
   1111 1111 1011 0110 1010 0000 0000 0000 0000 0000
   ---sr2---|--------sr1--------|--------sr0--------
   ```

3. Normalize, Modifier=Lo, [SR OR], SE = –3

   Second Input: 0110 1110 1100 1011

   SR (Normalized):

   ```
   1111 1111 1011 0110 1010 0011 0111 0110 0101 1000
   ---sr2---|--------sr1--------|--------sr0--------
   ```

If the upper half of the double precision input contains all sign bits, the SE register value is determined by the second derive exponent operation as shown in this second double precision normalization example:

1. Detect Exponent, Modifier = Hi

   First Input: `1111 1111 1111 1111` (upper half)
   SE set to: -15

2. Detect Exponent, Modifier = Lo

   Second Input: `1111 0110 1101 0100`
   SE now set to: -19

3. Normalize, Modifier=Hi, No [SR OR], SE = –19 (negated)

   First Input: `1111 1111 1111 1111`

   SR (Normalized):

   ```
   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
   ---sr2---|--------sr1--------|--------sr0--------
   ```

   Note that all values of SE less than –15 (resulting in a shift of +16 or more) upshift the input completely off scale.

4. Normalize, Modifier=Lo, [SR OR], SE = –19 (negated)

   Second Input: `1111 0110 1101 0100`

   SR (Normalized):

   ```
   1111 1111 1011 0110 1010 0000 0000 0000 0000 0000
   ---sr2---|--------sr1--------|--------sr0--------
   ```

Shifter Status Flags

The shifter's logical shift, arithmetic shift, normalize, and derive exponent operations update status flags in the computational unit's arithmetic status register (ASTAT). Table A-5 on page A-9 lists all the bits in this register. The following bit in ASTAT flags shifter status (a 1 indicates the condition) for the most recent shifter derive exponent operation:

- **Shifter result overflow.** Bit 7 (SV) indicates overflow (if set, =1) when the MSB of SR1 does not match the eight LSBs of SR2 or indicates no overflow (if clear, =0).
- **Shifter input sign for exponent extract only.** Bit 8 (SS)

Flag updates occur at the end of the cycle in which the status is generated and is available on the next cycle.

On previous 16-bit, fixed-point DSPs (ADSP-2100 family), the Shifter Results (SR) register is 32 bits wide and has no overflow detection. On ADSP-219x DSPs, the SR register is 40 bits wide, and the SV flag indicates overflow in SR.

Shifter Instruction Summary

Table 2-9 on page 2-51 lists the shifter instructions and shows how they relate to ASTAT flags. For more information on assembly language syntax, see the *ADSP-219x DSP Instruction Set Reference*. In Table 2-9, note the meaning of the following symbols:

- **Dreg** indicates any register file location
- * indicates the flag may be set or cleared, depending on the results of the instruction
- – indicates no effect

Table 2-9. Shifter Instruction Summary

Instruction	ASTAT Status Flags	
	SV	SS
[IF Cond] SR = [SR OR] ASHIFT Dreg [(|HI, LO|)];	*	–
SR = [SR OR] ASHIFT BY <Imm8> [(|HI, LO|)];	*	–
[IF Cond] SR = [SR OR] LSHIFT Dreg [(|HI, LO|)];	*	–
SR = [SR OR] LSHIFT BY <Imm8> [(|HI, LO|)];	*	–
[IF Cond] SR = [SR OR] NORM Dreg [(|HI, LO|)];	*	–
[IF Cond] SR = [SR OR] NORM <Imm8> [(|HI, LO|)];	*	–
[IF Cond] SE = EXP Dreg [(|HIX, HI, LO|)];	–	*[1]
[IF Cond] SB = EXPADJ Dreg;	–	–

1 The SS bit is the MSB of input for the HI option and is the MSB of input (for AV=0) or inverted MSB of input (for AV=1) for the HIX option; there is no effect on SS flag for the LO option.

Shifter Data Flow Details

Figure 2-14 shows a more detailed diagram of the shifter, which appears in Figure 2-1 on page 2-3. The shifter has the following components: the shifter array, the OR/PASS logic, the exponent detector, and the exponent compare logic.

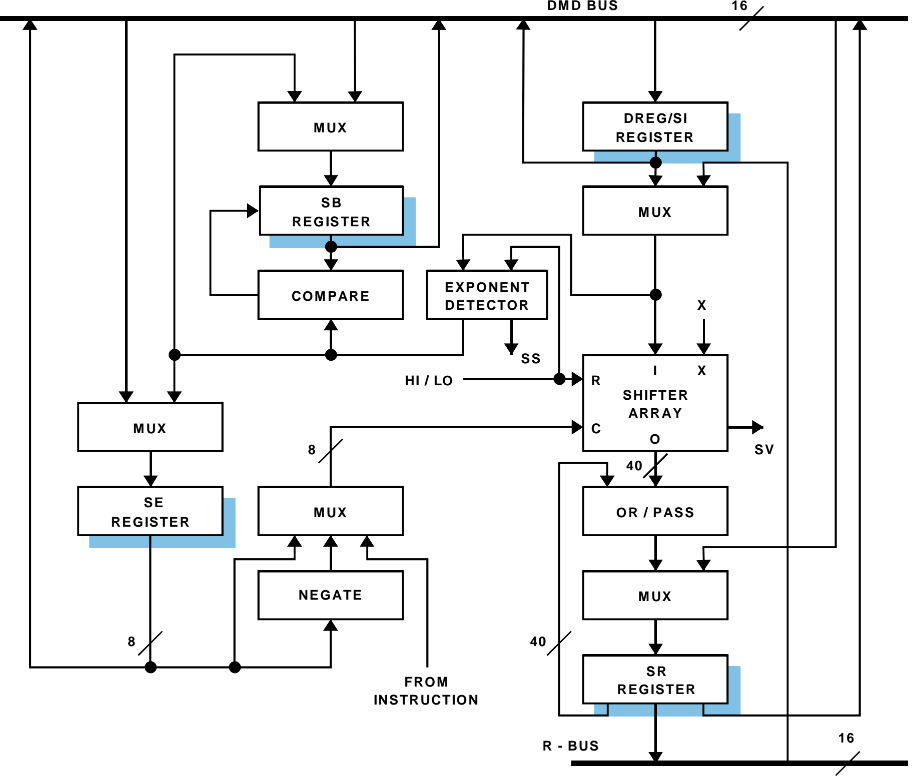


Figure 2-14. Shifter Block Diagram

The shifter array is a 16x40 barrel shifter. It accepts a 16-bit input and can place it anywhere in the 40-bit output field, from off-scale right to off-scale left, in a single cycle. This spread gives 57 possible placements

within the 40-bit field. The placement of the 16 input bits is determined by a shift control code (C) and a Hi/Lo option.

Depending on the instruction, the input port of the shifter can accept data from two sources: the data register file or the result (R) bus. Register usage for shifter input is only restricted in one instruction: the multifunction shift with memory read or write. In this instruction, only the shifter input (SI) register or result registers can provide input to the shifter array and the exponent detector.

For more information on register usage restrictions in conditional and multifunction instructions, see "Multifunction Computations" on page 2-60.

The shifter input (from register file or SI) provides input to the shifter array and the exponent detector. The SI register is 16 bits wide and is readable and writable from the DMD bus. The shifter array and the exponent detector also take as inputs AR, SR or MR via the R bus. The shifter result (SR) register is 40 bits wide and is divided into three sections: SR0, SR1, and SR2. These registers can be loaded from the DMD bus and output to either the DMD bus or the R bus. The SR register is also fed back to the OR/PASS logic to allow double-precision shift operations.

The SE register ("shifter exponent") is 8 bits wide and holds the exponent during the normalize and denormalize operations. The SE register is loadable and readable from the lower 8 bits of the DMD bus. It is a two's complement, 8.0 value.

The SB register ("shifter block") is important in block floating-point operations because it holds the block exponent value. The block exponent value is the value by which the block values must be shifted to normalize the largest value. SB is 5 bits wide and holds the most recent block exponent value. The SB register is loadable and readable from the lower 5 bits of the DMD bus. It is a two's complement, 5.0 value.

Whenever the SE or SB registers are output onto the DMD bus, they are sign-extended to form a 16-bit value.

Any of the SI, SE, or SR registers can be read and written in the same cycle. Registers are read at the beginning of the cycle and written at the end of the cycle. All register reads get values loaded at the end of a previous cycle. A new value written to a register cannot be read out until a subsequent cycle. This allows an input register to provide an operand to the shifter at the beginning of the cycle and be updated with the next operand at the end of the same cycle. It also allows a result register to be stored in memory and updated with a new result in the same cycle.

The shifter contains a duplicate bank of registers (shown behind the primary registers in Figure 2-14). There are actually two sets of SE, SB, SI, SR2, SR1, and SR0 registers. Only one bank is accessible at a time. The additional bank of registers can be activated for extremely fast context switching. A new task, such as an interrupt service routine, can then be executed without transferring current states to storage. For more information, see "Secondary (Alternate) Data Registers" on page 2-59.

The shifting of the input is determined by a control code (C) and a Hi/Lo option. The control code is an 8-bit signed value that indicates the direction and number of places the input is to be shifted. Positive codes indicate a left shift (upshift) and negative codes indicate a right shift (downshift). The control code can come from three sources: the content of the shifter exponent (SE) register, the negated content of the SE register, or an immediate value from the instruction.

The Hi/Lo option determines the reference point for the shifting. In the Hi state, all shifts are referenced to SR1 (the upper half of the output field), and in the Lo state, all shifts are referenced to SR0 (the lower half). The Hi/Lo feature is useful when shifting 32-bit values because it allows both halves of the number to be shifted with the same control code.The Hi/Lo option is selectable each time the shifter is used.

The shifter fills any bits to the right of the input value in the output field with zeros, and bits to the left are filled with the extension bit (X). The extension bit can be fed by three possible sources depending on the

instruction being performed. The three sources are the MSB of the input, the AC bit from the arithmetic status register (ASTAT), or a zero.

Figure 2-15 on page 2-56 shows the shifter array output as a function of the control code and Hi/Lo signal. In the figure, ABCDEFGHIJKLMNPR represents the 16-bit input pattern, and X stands for the extension bit.

The OR/PASS logic allows the shifted sections of a multiprecision number to be combined into a single quantity. In some shifter instructions, the shifted output may be logically OR'ed with the contents of the SR register; the shifter array is bitwise OR'ed with the current contents of the SR register before being loaded there. When the [SR OR] option is not used in the instruction, the shifter array output is passed through and loaded into the shifter result (SR) register unmodified.

The exponent detector derives an exponent for the shifter input value. The exponent detector operates in one of three ways that determine how the input value is interpreted. In the Hi state, the input is interpreted as a single precision number or the upper half of a double precision number. The exponent detector determines the number of leading sign bits and produces a code that indicates how many places the input must be up-shifted to eliminate all but one of the sign bits. The code is negative so that it can become the effective exponent for the mantissa formed by removing the redundant sign bits.

In the Hi-extend state (Hix), the input is interpreted as the result of an add or subtract performed in the ALU which may have overflowed. The exponent detector takes the arithmetic overflow (AV) status into consideration. If AV is set, then a +1 exponent is output to indicate an extra bit is needed in the normalized mantissa (the ALU Carry bit); if AV is not set, then Hi-extend functions exactly like the Hi state. When performing a derive exponent function in Hi or Hi-extend modes, the exponent detector also outputs a shifter sign (SS) bit, which is loaded into the arithmetic status register (ASTAT). The sign bit is the same as the MSB of the shifter input except when AV is set; when AV is set in Hi-extend state, the MSB is inverted to restore the sign bit of the overflowed value.

HI Reference Shift Value	LO Reference Shift Value	Shifter Results ---SR2--	-------SR1-------		-------SR0-------	
+24 to +127	**+40 to +127**	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**
+23	+39	R0000000	00000000	00000000	00000000	00000000
+22	+38	PR000000	00000000	00000000	00000000	00000000
+21	+37	NPR00000	00000000	00000000	00000000	00000000
+20	+36	MNPR0000	00000000	00000000	00000000	00000000
+19	+35	LMNPR000	00000000	00000000	00000000	00000000
+18	+34	KLMNPR00	00000000	00000000	00000000	00000000
+17	+33	JKLMNPR0	00000000	00000000	00000000	00000000
+16	**+32**	**IJKLMNPR**	**00000000**	**00000000**	**00000000**	**00000000**
+15	+31	HIJKLMNP	R0000000	00000000	00000000	00000000
+14	+30	GHIJKLMN	PR000000	00000000	00000000	00000000
+13	+29	FGHIJKLM	NPR00000	00000000	00000000	00000000
+12	+28	EFGHIJKL	MNPR0000	00000000	00000000	00000000
+11	+27	DEFGHIJK	LMNPR000	00000000	00000000	00000000
+10	+26	CDEFGHIJ	KLMNPR00	00000000	00000000	00000000
+ 9	+25	BCDEFGHI	JKLMNPR0	00000000	00000000	00000000
+ 8	**+24**	**ABCDEFGH**	**IJKLMNPR**	**00000000**	**00000000**	**00000000**
+ 7	+23	XABCDEFG	HIJKLMNP	R0000000	00000000	00000000
+ 6	+22	XXABCDEF	GHIJKLMN	PR000000	00000000	00000000
+ 5	+21	XXXABCDE	FGHIJKLM	NPR00000	00000000	00000000
+ 4	+20	XXXXABCD	EFGHIJKL	MNPR0000	00000000	00000000
+ 3	+19	XXXXXABC	DEFGHIJK	LMNPR000	00000000	00000000
+ 2	+18	XXXXXXAB	CDEFGHIJ	KLMNPR00	00000000	00000000
+ 1	+17	XXXXXXXA	BCDEFGHI	JKLMNPR0	00000000	00000000
0	**+16**	**XXXXXXXX**	**ABCDEFGH**	**IJKLMNPR**	**00000000**	**00000000**
- 1	+15	XXXXXXXX	XABCDEFG	HIJKLMNP	R0000000	00000000
- 2	+14	XXXXXXXX	XXABCDEF	GHIJKLMN	PR000000	00000000
- 3	+13	XXXXXXXX	XXXABCDE	FGHIJKLM	NPR00000	00000000
- 4	+12	XXXXXXXX	XXXXABCD	EFGHIJKL	MNPR0000	00000000
- 5	+11	XXXXXXXX	XXXXXABC	DEFGHIJK	LMNPR000	00000000
- 6	+10	XXXXXXXX	XXXXXXAB	CDEFGHIJ	KLMNPR00	00000000
- 7	+ 9	XXXXXXXX	XXXXXXXA	BCDEFGHI	JKLMNPR0	00000000
- 8	**+ 8**	**XXXXXXXX**	**XXXXXXXX**	**ABCDEFGH**	**IJKLMNPR**	**00000000**
- 9	+ 7	XXXXXXXX	XXXXXXXX	XABCDEFG	HIJKLMNP	R0000000
-10	+ 6	XXXXXXXX	XXXXXXXX	XXABCDEF	GHIJKLMN	PR000000
-11	+ 5	XXXXXXXX	XXXXXXXX	XXXABCDE	FGHIJKLM	NPR00000
-12	+ 4	XXXXXXXX	XXXXXXXX	XXXXABCD	EFGHIJKL	MNPR0000
-13	+ 3	XXXXXXXX	XXXXXXXX	XXXXXABC	DEFGHIJK	LMNPR000
-14	+ 2	XXXXXXXX	XXXXXXXX	XXXXXXAB	CDEFGHIJ	KLMNPR00
-15	+ 1	XXXXXXXX	XXXXXXXX	XXXXXXXA	BCDEFGHI	JKLMNPR0
-16	**0**	**XXXXXXXX**	**XXXXXXXX**	**XXXXXXXX**	**ABCDEFGH**	**IJKLMNPR**
-17	- 1	XXXXXXXX	XXXXXXXX	XXXXXXXX	XABCDEFG	HIJKLMNP
-18	- 2	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXABCDEF	GHIJKLMN
-19	- 3	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXABCDE	FGHIJKLM
-20	- 4	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXABCD	EFGHIJKL
-21	- 5	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXABC	DEFGHIJK
-22	- 6	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXAB	CDEFGHIJ
-23	- 7	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXA	BCDEFGHI
-24	**- 8**	**XXXXXXXX**	**XXXXXXXX**	**XXXXXXXX**	**XXXXXXXX**	**ABCDEFGH**
-25	- 9	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	XABCDEFG
-26	-10	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXABCDEF
-27	-11	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXABCDE
-28	-12	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXABCD
-29	-13	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXABC
-30	-14	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXAB
-31	-15	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXA
-32 to -128	**-16 to -128**	**XXXXXXXX**	**XXXXXXXX**	**XXXXXXXX**	**XXXXXXXX**	**XXXXXXXX**

Figure 2-15. Shifter Array Output Placement

In the Lo state, the input is interpreted as the lower half of a double precision number. In the Lo state, the exponent detector interprets the SS bit in the arithmetic status register (ASTAT) as the sign bit of the number. The SE register is loaded with the output of the exponent detector only if SE contains –15. This occurs only when the upper half (which must be processed first) contains all sign bits. The exponent detector output is also offset by –16 to account for the fact that the input is actually the lower 16 bits of a 40-bit value. Figure 2-16 on page 2-58 gives the exponent detector characteristics for all three modes.

The exponent compare logic is used to find the largest exponent value in an array of shifter input values. The exponent compare logic, in conjunction with the exponent detector, derives a block exponent. The comparator compares the exponent value derived by the exponent detector with the value stored in the shifter block exponent (SB) register and updates the SB register only when the derived exponent value is larger than the value in SB register.

Data Register File

The DSP's computational units have a data register file: a set of data registers that transfer data between the data buses and the computation units. DSP programs use these registers for local storage of operands and results.

Figure 2-1 on page 2-3 shows the register file appears. The register file consists of 16 primary registers and 16 secondary (alternate) registers. All of the data registers are 16 bits wide.

Program memory data accesses and data memory accesses to/from the register file occur on the PM data bus and DM data bus, respectively. One PM data bus access and/or one DM data bus access can occur in one cycle. Transfers between the register files and the DM or PM data buses can move up to 16-bits of valid data on each bus.

S = Sign bit
N = Non-sign bit
D = Don't care bit

HI Mode

Shifter Array Input	Output
SNDDDDDD DDDDDDDD	0
SSNDDDDD DDDDDDDD	-1
SSSNDDDD DDDDDDDD	-2
SSSSNDDD DDDDDDDD	-3
SSSSSNDD DDDDDDDD	-4
SSSSSSND DDDDDDDD	-5
SSSSSSSN DDDDDDDD	-6
SSSSSSSS NDDDDDDD	-7
SSSSSSSS SNDDDDDD	-8
SSSSSSSS SSNDDDDD	-9
SSSSSSSS SSSNDDDD	-10
SSSSSSSS SSSSNDDD	-11
SSSSSSSS SSSSSNDD	-12
SSSSSSSS SSSSSSND	-13
SSSSSSSS SSSSSSSN	-14
SSSSSSSS SSSSSSSS	-15

HIX Mode

AV	Shifter Array Input	Output
1	DDDDDDDD DDDDDDDD	+1
0	SNDDDDDD DDDDDDDD	0
0	SSNDDDDD DDDDDDDD	-1
0	SSSNDDDD DDDDDDDD	-2
0	SSSSNDDD DDDDDDDD	-3
0	SSSSSNDD DDDDDDDD	-4
0	SSSSSSND DDDDDDDD	-5
0	SSSSSSSN DDDDDDDD	-6
0	SSSSSSSS NDDDDDDD	-7
0	SSSSSSSS SNDDDDDD	-8
0	SSSSSSSS SSNDDDDD	-9
0	SSSSSSSS SSSNDDDD	-10
0	SSSSSSSS SSSSNDDD	-11
0	SSSSSSSS SSSSSNDD	-12
0	SSSSSSSS SSSSSSND	-13
0	SSSSSSSS SSSSSSSN	-14
0	SSSSSSSS SSSSSSSS	-15

LO Mode

SS	Shifter Array Input	Output
S	NDDDDDDD DDDDDDDD	-15
S	SNDDDDDD DDDDDDDD	-16
S	SSNDDDDD DDDDDDDD	-17
S	SSSNDDDD DDDDDDDD	-18
S	SSSSNDDD DDDDDDDD	-19
S	SSSSSNDD DDDDDDDD	-20
S	SSSSSSND DDDDDDDD	-21
S	SSSSSSSN DDDDDDDD	-22
S	SSSSSSSS NDDDDDDD	-23
S	SSSSSSSS SNDDDDDD	-24
S	SSSSSSSS SSNDDDDD	-25
S	SSSSSSSS SSSNDDDD	-26
S	SSSSSSSS SSSSNDDD	-27
S	SSSSSSSS SSSSSNDD	-28
S	SSSSSSSS SSSSSSND	-29
S	SSSSSSSS SSSSSSSN	-30
S	SSSSSSSS SSSSSSSS	-31

Figure 2-16. Exponent Detector Characteristics

If an operation specifies the same register file location as both an input and output, the read occurs in the first half of the cycle and the write in the second half. With this arrangement, the DSP uses the old data as the operand, before updating the location with the new result data. If writes to the same location take place in the same cycle, only the write with higher precedence actually occurs. The DSP determines precedence for the write from the type of the operation; from highest to lowest, the precedence is:

1. Move operations: register-to-register, register-to-memory, or memory-to-register

2. Compute operations: ALU, multiplier, or shifter

Secondary (Alternate) Data Registers

Computational units have a secondary register set. To facilitate fast context switching, the DSP includes secondary register sets for data, results, and data address generator registers. Bits in the MSTAT register control when secondary registers become accessible. While inaccessible, the contents of secondary registers are not affected by DSP operations. Note that there is a one cycle latency between writing to MSTAT and being able to access a secondary register set. The secondary register sets for data and results are described in this section.

For more information on secondary data address generator registers, see the “Secondary (Alternate) DAG Registers” on page 4-4.

The MSTAT register controls access to the secondary registers. Table A-6 on page A-11 lists all the bits in MSTAT. The SEC_REG bit in MSTAT controls secondary registers (a 1 enables the secondary set). When set (=1), secondary registers are enabled for the AX0, AX1, AY0, AY1, MX0, MX1, MY0, MY1, SI, SB, SE, AR, MR, and SR registers.

The following example demonstrates how code handles the one cycle of latency that occurs from the time the instruction sets the bit in MSTAT to when the secondary registers are available for accessing.

```
AR = MSTAT;
AR = Setbit SEC_REG Of AR;
MSTAT=AR;              /* activate secondary reg. file */
Nop;                   /* wait for access to secondaries */
AX0 = 7;
```

It is more efficient (no latency) to use the mode enable instruction to select secondary registers. In the following example, note that the swap to secondary registers is immediate:

```
Ena SEC_REG;           /* activate secondary reg. file */
AX0 = 7;               /* now use the secondaries */
```

Multifunction Computations

Using the many parallel data paths within its computational units, the DSP supports multiple-parallel (multifunction) computations. These instructions complete in a single cycle, and they combine parallel operation of the multiplier, ALU, or shifter with data move operations. The multiple operations perform the same as if they were in corresponding single-function computations. Multifunction computations also handle flags in the same way as the single-function computations.

To work with the available data paths, the computation units constrain which data registers may hold the input operands for multifunction computations. These constraints limit which registers may hold the X-input and Y-input for the ALU, multiplier, and shifter.

Figure 2-17 on page 2-62 shows how some register access restrictions apply to conditional and/or multifunction instructions. The boxes around the X- and Y-inputs within the register file only apply for conditional and/or multifunction instructions. For unconditional, single-function instructions, any of the registers within the register file may serve as X- or

Y-inputs (see Figure 2-1 on page 2-3). The following code example shows the differences between conditional versus unconditional instructions and single-function versus multifunction instructions.

```
/* Conditional computation instructions begin with an IF
clause. The DSP tests whether the condition is true before
executing the instruction. */

AR = AX0 + AY0; /*unconditional: add X and Y ops*/
If EQ AR = AX0 + AY0; /*conditional: if AR=0, add X and Y ops*/

/* Multifunction instructions are sets of instruction that
execute in a single cycle. The instructions are delimited with
commas, and the combined multifunction instruction is
terminated with a semicolon. */

AR = AX0-AY0;   /* single function ALU subtract */
AX0 = MR1;     /* single function register-to-register move */
AR = AX0-AY0, AX0 = MR1; /* multifunction, both in 1 cycle */
```

The data paths over the Results Bus from the results registers let the result registers (MR2, MR1, MR0, SR1, SR0, or AR) serve as X-inputs to the ALU and multiplier in both the conditional/multifunction and unconditional/single-function cases. The upper part of the Shifter Results (SR) register, SR2, may not serve as feedback over the results bus. For information on the SR2, SB, SE, MSTAT, and ASTAT registers in Figure 2-17, see the discussion on page 2-2.

Only the ALU and multiplier X- and Y-operand registers (MX0, MX1, MY0, MY1, AX0, AY1) have memory data bus access in dual memory read multifunction instructions.

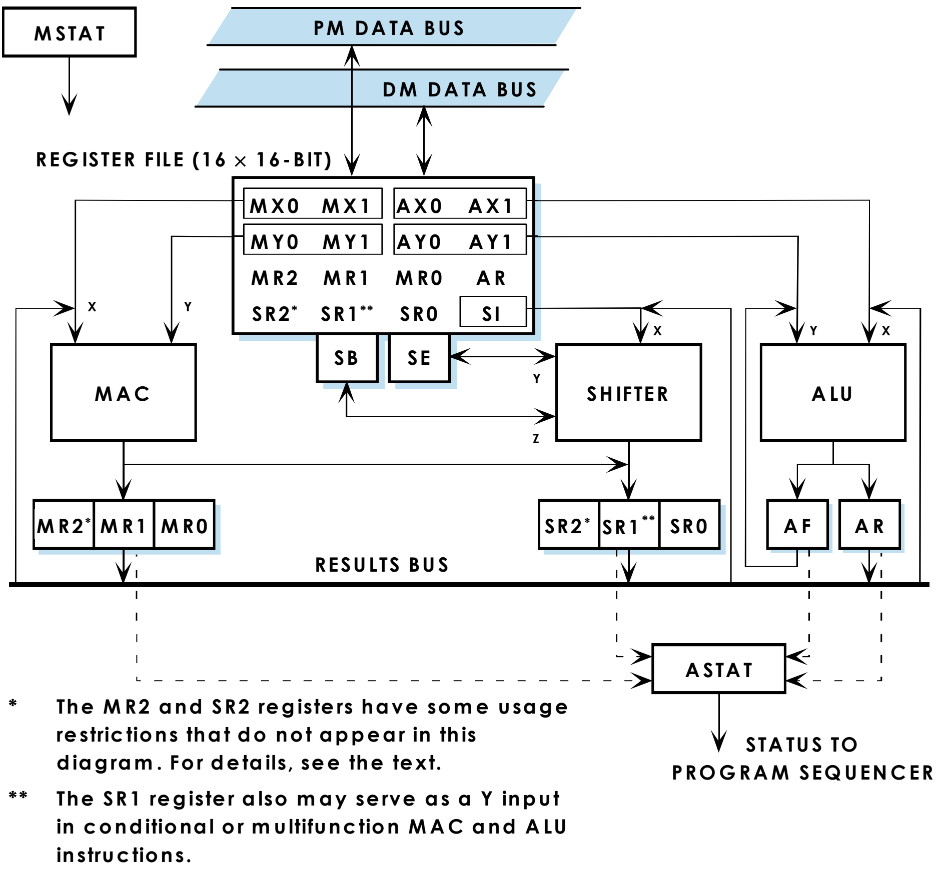


Figure 2-17. Register Access for Conditional/Multifunction Instructions

Table 2-10 lists the multifunction instructions. For more information on assembly language syntax, see the *ADSP-219x DSP Instruction Set Reference*. In these tables, note the meaning of the following symbols:

- **ALU, MAC, SHIFT** indicate any ALU, multiplier, or shifter instruction
- **Dreg** indicates any register file location
- **Xop, Yop** indicate any X- and Y-input registers, indicating a register usage restriction for conditional and/or multifunction instructions.

Table 2-10. ADSP-219x Multifunction Instruction Summary

Instruction[1]
|<ALU>, <MAC>|, Xop = DM(Ia += Mb), Yop = PM(Ic += Md);
Xop = DM(Ia += Mb), Yop = PM(Ic += Md);
|<ALU>, <MAC>,<SHIFT> |, Dreg = |DM(Ia += Mb), PM(Ic += Md)|;
|<ALU>, <MAC>, <SHIFT>|, |DM(Ia += Mb), PM(Ic += Md)| = Dreg;
|<ALU>, <MAC>, <SHIFT>|, Dreg = Dreg;

1 Multifunction instructions are sets of instruction that execute in a single cycle. The instructions are delimited with commas, and the combined multifunction instruction is terminated with a semicolon.

3 PROGRAM SEQUENCER

Overview

The DSP's program sequencer controls program flow by providing the address of the next instruction to be executed by other parts of the DSP. Program flow in the DSP is mostly linear, with the processor executing program instructions sequentially. This linear flow varies occasionally when the program uses non-sequential program structures, such as those illustrated in Figure 3-1 on page 3-2. Non-sequential structures direct the DSP to execute an instruction that is not at the next sequential address. These structures include:

- **Loops.** One sequence of instructions executes several times with near-zero overhead.
- **Subroutines.** The processor temporarily interrupts sequential flow to execute instructions from another part of program memory.
- **Jumps.** Program flow transfers permanently to another part of program memory.
- **Interrupts.** Subroutines in which a runtime event (not an instruction) triggers the execution of the routine.
- **Idle.** An instruction that causes the processor to cease operations, holding its current state until an interrupt occurs. Then, the processor services the interrupt and continues normal execution.

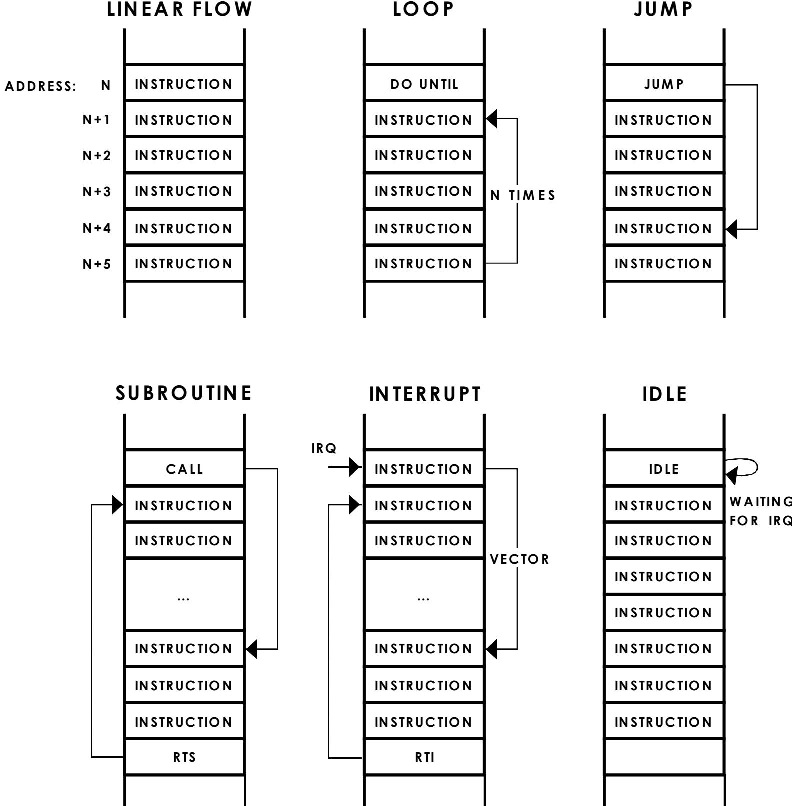


Figure 3-1. Program Flow Variations

The sequencer manages execution of these program structures by selecting the address of the next instruction to execute. As part of this process, the sequencer handles the following tasks:

- Increments the fetch address
- Maintains stacks
- Evaluates conditions
- Decrements the loop counter

- Calculates new addresses
- Maintains an instruction cache
- Handles interrupts

To accomplish these tasks, the sequencer uses the blocks shown in Figure 3-2 on page 3-4. The sequencer's address multiplexer selects the value of the next fetch address from several possible sources. The fetched address enters the instruction pipeline, ending with the program counter (PC). The pipeline contains the 24-bit addresses of the instructions currently being fetched, decoded, and executed. The PC couples with the PC stack, which stores return addresses. All addresses generated by the sequencer are 24-bit program memory instruction addresses.

Figure 3-2 uses the following abbreviations: ADDR=address, BRAN=branch, IND=indirect, DIR=direct, RT=return, RB=rollback, INCR=increment, PC-REL=PC relative, PC=program counter.

The diagram in Figure 3-2 also describes the relationship between the program sequencer in the ADSP-219x DSP core and inputs to that sequencer that differ for various members of the ADSP-219x family DSPs.

To manage events, the sequencer's interrupt controller handles interrupt processing, determines whether an interrupt is masked, and generates the appropriate interrupt vector address.

With selective caching, the instruction cache lets the DSP access data in program memory and fetch an instruction (from the cache) in the same cycle. The program sequencer uses the cache if there are two data accesses in a single cycle or if a single data access uses the same 16K block of memory as the current instruction fetch.

In addition to providing data addresses, the Data Address Generators (DAGs) provide instruction addresses for the sequencer's indirect branches.

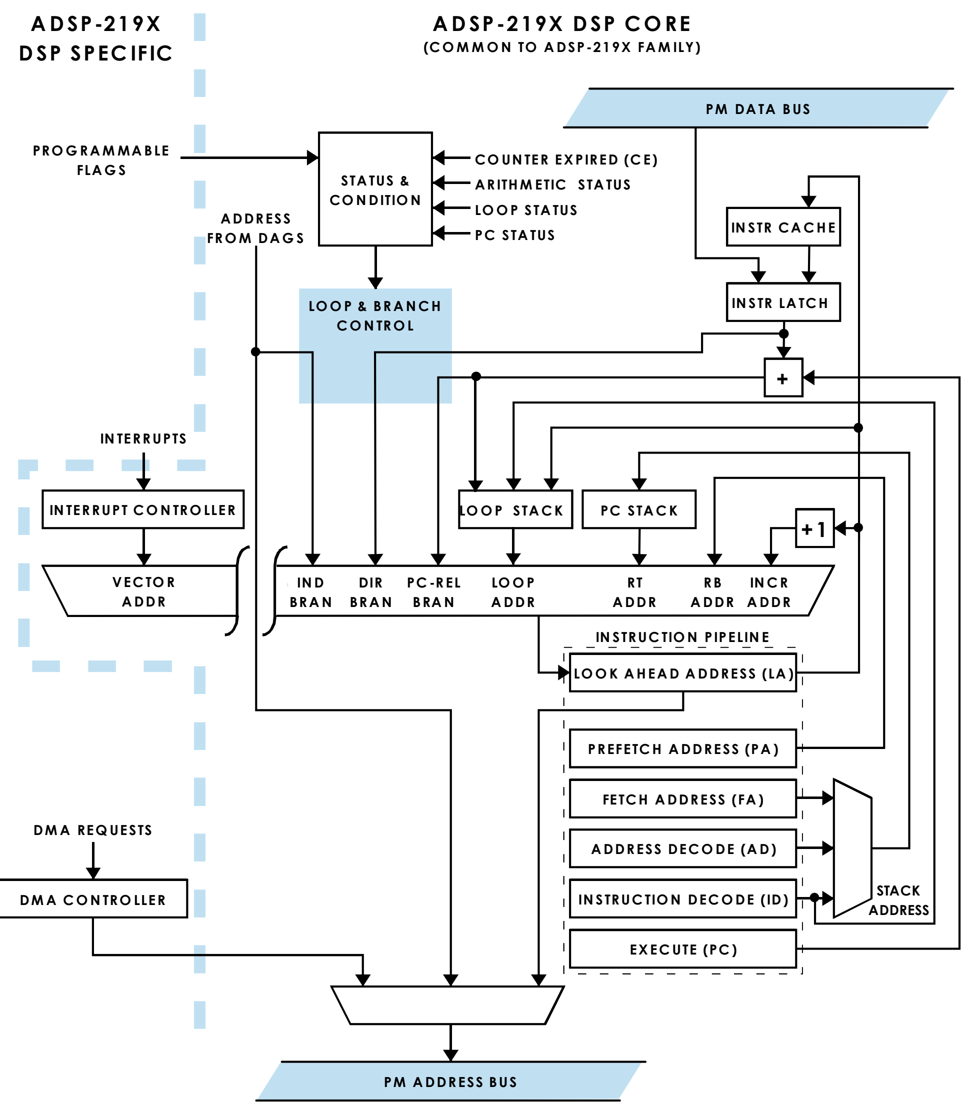


Figure 3-2. Program Sequencer Block Diagram

The sequencer evaluates conditional instructions and loop termination conditions using information from the status registers. The loop stacks support nested loops. The status stack stores status registers for implementing interrupt service routines.

Table 3-1 and Table 3-2 list the registers within and related to the program sequencer. All registers in the program sequencer are Register Group 1 (Reg1), 2 (Reg2), or 3 (Reg3) registers, so they are accessible to other data (Dreg) registers and to memory. All the sequencer's registers are directly readable and writable, except for the PC. Manual pushing or popping the PC stack is done using explicit instructions and the PC stack page (STACKP) and address (STACKA) registers, which are readable and writable. Pushing or popping the loop stacks and status stack also requires explicit instructions. For information on using these stacks, see "Stacks and Sequencing" on page 3-34.

A set of system control registers configures or provides input to the sequencer. These registers include ASTAT, MSTAT, CCODE, IMASK, IRPTL, and ICNTL. Writes to some of these registers do not take effect on the next cycle. For example, after a write to the MSTAT register to enable ALU saturation mode, the change does not take effect until one cycle after the write. Table 3-1 and Table 3-2 summarize the number of extra cycles (latency) for a write to take effect (effect latency) and for a new value to appear in the register (read latency). A "0" indicates that the write takes effect or appears in the register on the next cycle after the write instruction is executed, and a "1" indicates one extra cycle.

Table 3-1. Program Sequencer Registers

Register	Contents	Bits	Effect Latency
CNTR	loop count loaded on next Do/Until loop	16	1[1]
IJPG	Jump Page (upper eight bits of address)	8	1
IOPG	I/O Page (upper eight bits of address)	8	1
DMPG1	DAG1 Page (upper eight bits of address)	8	1
DMPG2	DAG2 Page (upper eight bits of address)	8	1

1 CNTR has a one-cycle latency before an If Not CE instruction, but has zero latency otherwise.

Table 3-2. System Registers

Register	Contents	Bits	Effect Latency
ASTAT	Arithmetic status	9	1
MSTAT	Mode status	7	0[1]
SSTAT	System status	8	n/a
CCODE	Condition Code	16	1
IRPTL	Interrupt latch	16	1
IMASK	Interrupt mask	16	1
ICNTL	Interrupt control	16	1
CACTL	Cache control	3	5[2]

1 Changing MSTAT bits with the Ena or Dis mode instruction has a 0 effect latency; when writing to MSTAT or performing a Pop Sts the effect latencies vary based on the altered bits.
2 Except for the CFZ bit, which has an effect latency of four cycles.

Instruction Pipeline

The program sequencer determines the next instruction address by examining both the current instruction being executed and the current state of the processor. If no conditions require otherwise, the DSP executes instructions from program memory in sequential order by incrementing the look-ahead address. Using its instruction pipeline, the DSP processes instructions in six clock cycles:

- **Look-Ahead Instruction (LA).** The DSP determines the source for the instruction from inputs to the look-ahead address multiplexer.
- **Prefetch Instruction (PA)** and **Fetch Instruction (FA).** The DSP reads the instruction from either the on-chip instruction cache or from program memory.
- **Address Decode (AD)** and **Instruction Decode (ID).** The DSP decodes the instruction, generating conditions that control instruction execution.
- **Execute (PC).** The DSP executes the instruction; the operations specified by the instruction complete in a single cycle.

These cycles overlap in the pipeline, as shown in Table 3-3 on page 3-8. In sequential program flow, when one instruction is being fetched, the instruction fetched three cycles previously is being executed. With few exceptions, sequential program flow has a throughput of one instruction per cycle. The exceptions are the two-cycle instructions: 16- or 24-bit immediate data write to memory with indirect addressing, long jump (`Ljump`), and long call (`Lcall`).

Any non-sequential program flow can potentially decrease the DSP's instruction throughput. Non-sequential program operations include:

- Program memory data accesses that conflict with instruction fetches
- Jumps

Table 3-3. Pipelined Execution Cycles

Cycles	LA	PA	FA	AD	ID	PC
1	0x08 ↘					
2	0x09 ↘	0x08 ↘				
3	0x0A ↘	0x09 ↘	0x08 ↘			
4	0x0B ↘	0x0A ↘	0x09 ↘	0x08 ↘		
5	0x0C ↘	0x0B ↘	0x0A ↘	0x09 ↘	0x08 ↘	
6	0x0D ↘	0x0C ↘	0x0B ↘	0x0A ↘	0x09 ↘	0x08
7	0x0E ↘	0x0D ↘	0x0C ↘	0x0B ↘	0x0A ↘	0x09
8	0x0F	0x0E	0x0D	0x0C	0x0B	0x0A
Look Ahead Address (LA). Prefetch Address (PA). Fetch Address (FA). Address Decode (AD). Instruction Decode (ID). Execute (PC).						

- Subroutine calls and returns
- Interrupts and return
- Loops (of less than five instructions)

Instruction Cache

Usually, the sequencer fetches an instruction from memory on each cycle. Occasionally, bus constraints prevent some of the data and instructions from being fetched in a single cycle. To minimize the impact of these bus constraints on data flow, the DSP has an instruction cache, as shown in Figure 3-3. When the DSP executes an instruction that requires data access over the PM data bus, there is a bus conflict because the sequencer also uses the PM data bus for fetching instructions. To avoid these conflicts and reduce delays, the DSP caches instructions.

When the DSP encounters a fetch conflict, it must wait to fetch the instruction on the following cycle, causing a delay. The DSP automatically writes the fetched instruction to the cache to prevent the same delay from happening again. The sequencer checks the instruction cache on every program memory data access. If the appropriate instruction is in the cache, the instruction fetch (from the cache) occurs in parallel with the data access (from the PM data bus), without incurring a delay.

Because of the six-stage instruction pipeline, as the DSP executes an instruction (at address n) that requires a program memory data access, this execution creates a conflict with the instruction fetch (at address n+3). The cache stores the fetched instruction (n+3), not the instruction requiring the program memory data access.

If the instruction needed to avoid a conflict is in the cache, the cache provides the instruction while the program memory data access is performed. If the needed instruction is not in the cache, the instruction fetch from memory takes place in the cycle following the program memory data access, incurring one cycle of overhead. If the cache is enabled and not frozen, the fetched instruction is loaded into the cache, so that it is available the next time the same conflict occurs.

Figure 3-3 shows a block diagram of the instruction cache. The cache holds 64 instruction-address pairs. These pairs (or cache entries) are arranged into 32 (31-0) cache sets according to the instruction address' five least significant bits (4-0). The two entries in each set (entry 0 and entry 1) have a valid bit, indicating whether the entry contains a valid instruction. The least recently used (LRU) bit for each set indicates which entry was not used last (0=entry 0 and 1=entry 1).

Figure 3-3. Instruction Cache Architecture

The cache places instructions in entries according to the five LSBs of the instruction's address. When the sequencer checks for an instruction to fetch from the cache, it uses the five address LSBs as an index to a cache set. Within that set, the sequencer checks the addresses and valid bits of the two entries, looking for the needed instruction. If the cache contains the instruction, the sequencer uses the entry and updates the LRU bit to indicate the entry did not contain the needed instruction.

When the cache does not contain a needed instruction, the cache loads a new instruction and its address, placing these in the least recently used entry of the appropriate cache set and toggling the LRU bit.

Using The Cache

After a DSP reset, the cache starts cleared (containing no instructions), unfrozen, and enabled. From then on, the CACTL register controls the operating mode of the instruction cache. All of the bits in the CACTL register are listed in "ADSP-219x DSP Core Registers" on page A-1. The following bits in the CACTL register control cache modes:

- **Cache DM access Enable.** Bit 5 (CDE) directs the sequencer to cache conflicting DM bus accesses (if 1) or not to cache conflicting DM bus accesses (if 0).
- **Cache Freeze.** Bit 6 (CFZ) directs the sequencer to freeze the contents of the cache (if 1) or let new entries displace the entries in the cache (if 0).
- **Cache PM access Enable.** Bit 7 (CPE) directs the sequencer to cache conflicting PM bus accesses (if 1) or not to cache conflicting PM bus accesses (if 0).

When changing the cache mode or flushing the cache, the instruction containing a program memory data access must not be placed directly after a cache enable or cache disable instruction, because the DSP must wait at least one cycle before executing the PM data access. A program should have a Nop inserted after the cache enable instruction if necessary.

When program memory changes, the programs should resynchronize the instruction cache with the program memory using the Flush Cache instruction. This instruction flushes the instruction cache, invalidating all instructions currently cached, so that the next instruction fetch results in a memory access.

Optimizing Cache Usage

Usually, cache operation is efficient and requires no intervention, but certain ordering of instructions can work against the cache architecture and can degrade cache efficiency. When the order of PM data accesses and instruction fetches continuously displaces cache entries and loads new entries, the cache is not being efficient. Rearranging the order of these instructions can improve efficiency.

An example of code that works against cache efficiency appears in Table 3-4 on page 3-13. The program memory data access at address 0x0100 in the loop, `Outer`, causes the cache to load the instruction at 0x0103 (into set 19). Each time the program calls the subroutine, `Inner`, the program memory data accesses at 0x0300 and 0x500 displace the instruction at 0x0103 by loading the instructions at 0x0303 and 0x0503 (also into set 19). If the program only calls the `Inner` subroutine rarely during the `Outer` loop execution, the repeated cache loads do not greatly influence performance. If the program frequently calls the subroutine while in the loop, the cache inefficiency has a noticeable effect on performance. To improve cache efficiency on this code (if for instance, execution of the `Outer` loop is time-critical), it would be good to rearrange the order of some instructions. Moving the subroutine call up one location (starting at 0x02FE) would work here, because with that order the two cached instructions end up in cache set 18 instead of set 19.

Because the least significant five address bits determine which cache set store an instruction, instructions in the same cache set are multiples of 64 address locations apart. As demonstrated in the optimization example, it is a rare combination of instruction sequences that can lead to “cache thrashing”—iterative swapping of cache entries.

Table 3-4. Cache-Inefficient Code

Address	Instruction
0x00FE	`CNTR=1024;`
0x00FF	`Do Outer Until CE;`
0x0100	`AX0=DM(I0+=M0), AY0=PM(I4,M4);`
...	
0x0103	`If EQ Call Inner;`
0x0104	`AR=AX1 + AY1;`
0x0105	`MR=MX0*MY0 (SS);`
0x0106	`Outer: SR=MX1*MY1(SS);`
0x0107	`PM(I7+=M7)=SR1;`
...	
0x02FF	`Inner: SR0=AY0;`
0x0300	`AY0=PM(I5+=M5);`
...	
0x0500	`PM(I5+=M5)=AY1;`
...	
0x05FF	`Rts;`

Branches and Sequencing

One of the types of non-sequential program flow that the sequencer supports is branching. A branch occurs when a `Jump` or `Call`/return instruction begins execution at a new location, other than the next sequential address. For descriptions on how to use the `Jump` and `Call`/return instructions, see the ADSP-219x *DSP Instruction Set Reference*. Briefly, these instructions operate as follows:

- A `Jump` or a `Call`/return instruction transfers program flow to another memory location. The difference between a `Jump` and a `Call` is that a `Call` automatically pushes the return address (the next sequential address after the `Call` instruction) onto the `PC` stack. This push makes the address available for the `Call` instruction's matching return instruction, allowing easy return from the subroutine.

- A return instruction causes the sequencer to fetch the instruction at the return address, which is stored at the top of the `PC` stack. The two types of return instructions are return from subroutine (`Rts`) and return from interrupt (`Rti`). While the return from subroutine (`Rts`) only pops the return address off the `PC` stack, the return from interrupt (`Rti`) pops the return address and pops the status stack.

There are a number of parameters that programs can specify for branches:

- `Jump` and `Call`/return instructions can be conditional. The program sequencer can evaluate status conditions to decide whether to execute a branch. If no condition is specified, the branch is always taken. For more information on these conditions, see "Conditional Sequencing" on page 3-39.

- `Jump` and `Call`/return instructions can be immediate or delayed. Because of the instructions pipeline, an immediate branch incurs four lost (overhead) cycles. A delayed branch incurs two cycles of overhead. For more information, see "Delayed Branches" on page 3-16.

- Jump and Call/return instructions can be used within Do/Until counter (CE) or infinite (Forever) loops, but a Jump or Call instruction may not be the last instruction in the loop. For information, see "Restrictions On Ending Loops" on page 3-24.

The sequencer block diagram in Figure 3-2 on page 3-4 shows that branches can be direct or indirect. The difference is that the sequencer generates the address for a direct branch, and the PM data address generator (DAG2) produces the address for an indirect branch.

Direct branches are Jump or Call/return instructions that use an absolute—not changing at runtime—address (such as a program label) or use a PC-relative 16-bit address. To branch farther, the Ljump or Lcall instructions use a 24-bit address. Some instruction examples that cause a direct branch are:

```
Jump fft1024; {where fft1024 is an address label}
Call 10; {where 10 a PC-relative address}
```

Indirect branches are Jump or Call/return instructions that use a dynamic—changes at runtime—address that comes from either data address generator. For more information on the data address generator, see "DAG Operations" on page 4-9. Some instruction examples that cause an indirect branch are:

```
Jump (I6); {where (i6) is a DAG1 or DAG2 register}
Call (I7); {where (i7) is a DAG1 or DAG2 register}
```

Indirect Jump Page (IJPG) Register

The IJPG register provides the upper eight address bits for indirect Jump and Call instructions. When performing an indirect branch, the sequencer gets the lower 16 bits of the branch address from the I register specified in the Jump or Call instruction and uses the IJPG register to complete the address.

At power up, the DSP initializes the `IJPG` register to `0x0`. Initializing the page register is only necessary when the instruction is located on a page other than the current page.

Changing the contents of the sequencer page register is not automatic and requires explicit programming.

Conditional Branches

The sequencer supports conditional branches. These are `Jump` or `Call`/return instructions whose execution is based on testing an `If` condition. For more information on condition types in `If` condition instructions, see "Conditional Sequencing" on page 3-39.

Delayed Branches

The instruction pipeline influences how the sequencer handles branches. For immediate branches—`Jump` and `Call`/return instructions not specified as delayed branches (`DB`), four instruction cycles are lost (`Nop`s) as the pipeline empties and refills with instructions from the new branch.

As shown in Table 3-5 and Table 3-6, the DSP does not execute the four instructions after the branch, which are in the fetch and decode stages. For a `Call`, the next instruction (the instruction after the `Call`) is the return address. During the four lost (no-operation) cycles, the pipeline fetches and decodes the first instruction at the branch address.

For delayed branches—`Jump` and `Call`/return instructions with the delayed branches (`DB`) modifier, only two instruction cycles are lost in the pipeline, because the DSP executes the two instructions after the branch while the pipeline fills with instructions from the new branch.

Table 3-5. Pipelined Execution Cycles For Immediate Branch (Jump/Call)

Cycles	LA	PA	FA	AD	ID	PC
1	j	n+4→nop[1]	n+3→nop[1]	n+2→nop[1]	n+1→nop[1]	n
2	j+1	j	n+4→nop[1]	n+3→nop[1]	n+2→nop[1]	Nop[2]
3	j+2	j+1	j	n+4→nop[1]	n+3→nop[1]	Nop
4	j+3	j+2	j+1	j	n+4→nop[1]	Nop
5	j+4	j+3	j+2	j+1	j	Nop
6	j+5	j+4	j+3	j+2	j+1	j

Note that n is the branching instruction, and j is the instruction branch address. Notes: (1) n+1, n+2, n+3, and n+4 are suppressed. (2) For call, return address (n+1) is pushed on PC stack.

Table 3-6. Pipelined Execution Cycles For Immediate Branch (Return)

Cycles	LA	PA	FA	AD	ID	PC
1	r	n+4→nop[1]	n+3→nop[1]	n+2→nop[1]	n+1→nop[1]	n
2	r+1	r	n+4→nop[1]	n+3→nop[1]	n+2→nop[1]	Nop[2]
3	r+2	r+1	r	n+4→nop[1]	n+3→nop[1]	Nop
4	r+3	r+2	r+1	r	n+4→nop[1]	Nop
5	r+4	r+3	r+2	r+1	r	Nop
6	r+5	r+4	r+3	r+2	r+1	r

Note that n is the branching instruction, and r is the instruction branch address. Notes: (1) n+1, n+2, n+3, and n+4 are suppressed. (2) r (n+1 in Table 3-5) the return address is popped from PC stack.

As shown in Table 3-7 and Table 3-8, the DSP executes the two instructions after the branch, while the instruction at the branch address is fetched and decoded. In the case of a `Call`, the return address is the third instruction after the branch instruction. While delayed branches use the instruction pipeline more efficiently than immediate branches, it is important to note that delayed branch code can be harder to understand because of the instructions between the branch instruction and the actual branch.

Table 3-7. Pipelined Execution Cycles For Delayed Branch (Jump/Call)

Cycles	LA	PA	FA	AD	ID	PC
1	j	n+4→nop[1]	n+3→nop[1]	n+2	n+1	n
2	j+1	j	n+4→nop[1]	n+3→nop[1]	n+2	n+1[2]
3	j+2	j+1	j	n+4→nop[1]	n+3→nop[1]	n+2[2]
4	j+3	j+2	j+1	j	n+4→nop[1]	Nop[3]
5	j+4	j+3	j+2	j+1	j	Nop
6	j+5	j+4	j+3	j+2	j+1	j

Note that n is the branching instruction, and j is the instruction branch address. Notes: (1) n+3 and n+4 are suppressed. (2)Delayed branch slots. (3) For call, return address (n+3) is pushed on PC stack.

Table 3-8. Pipelined Execution Cycles For Delayed Branch (Return)

Cycles	LA	PA	FA	AD	ID	PC
1	r[1]	n+4→nop[2]	n+3→nop[2]	n+2	n+1	n
2	r+1	r	n+4→nop[2]	n+3→nop[2]	n+2	n+1[3]
3	r+2	r+1	r	n+4→nop[2]	n+3→nop[2]	n+2[3]
4	r+3	r+2	r+1	r	n+4→nop[2]	Nop
5	r+4	r+3	r+2	r+1	r	Nop
6	r+5	r+4	r+3	r+2	r+1	r

Note that n is the branching instruction, and r is the instruction branch address. Notes: (1) r (n+1 in Table 3-7) the return address is popped from PC. (2) stackn+3 and n+4 are suppressed. (3) Delayed branch slots.

Besides being somewhat more challenging to code, there are also some limitations on delayed branches that stem from the instruction pipeline architecture. Because the delayed branch instruction and the two instructions that follow it must execute sequentially, the instructions in the two locations (delayed branch slots) that follow a delayed branch instruction may not be any of the following:

- Other branches (no `Jump`, `Call`, or `Rti/Rts` instructions)
- Any stack manipulations (no `Push` or `Pop` instructions or writes to the `PC` stack)
- Any loops or other breaks in sequential operation (no `Do/Until` or `Idle` instructions)
- Two-cycle instructions may not appear in the second delay branch slot; these instructions may appear in the first delay branch slot.

Interrupt processing is also influenced by delayed branches and the instruction pipeline. Because the delayed branch instruction and the two instructions that follow it must execute sequentially, the DSP does not immediately process an interrupt that occurs in between a delayed branch instruction and either of the two instructions that follow. Any interrupt that occurs during these instructions is latched, but not processed until the branch is complete.

Loops and Sequencing

Another type of non-sequential program flow that the sequencer supports is looping. A loop occurs when a Do/Until instruction causes the DSP to repeat a sequence of instructions infinitely (Forever) or until the counter expires (CE).

The condition for terminating a loop with the Do/Until logic is loop Counter Expired (CE). This condition tests whether the loop has completed the number of iterations loaded from the CNTR register. Loops that exit with conditions other than CE (using a conditional Jump) have some additional restrictions. For more information, see "Restrictions On Ending Loops" on page 3-24. For more information on condition types in Do/Until instructions, see "Conditional Sequencing" on page 3-39.

The Do/Until instruction uses the sequencer's loop and condition features, which appear in Figure 3-2 on page 3-4. These features provide efficient software loops, without the overhead of additional instructions to branch, test a condition, or decrement a counter. The following code example shows a Do/Until loop that contains three instructions and iterates 30 times.

```
CNTR=30; Do the_end Until CE; {loop iterates 30 times}
AX0=DM(I0+=M0), AY0=PM(I4+=M4);
AR=AX0-AY0;
the_end: DM(I1+=M0)=AR;       {last instruction in loop}
```

When executing a Do/Until instruction, the program sequencer pushes the address of the loop's last instruction and loop's termination condition onto the loop-end stack. The sequencer also pushes the loop-begin address—address of the instruction following the Do/Until instruction—onto the loop-begin stack.

The sequencer's instruction pipeline architecture influences loop termination. Because instructions are pipelined, the sequencer must test the termination condition and decrement the counter at the end of the loop. Based on the test's outcome, the next fetch either exits the loop or returns to the beginning of the loop.

The Do/Until instruction supports infinite loops, using the Forever condition instead of CE. Software can use a conditional Jump instruction to exit such an infinite loop.

When using a conditional Jump to exit any Do/Until loop, software must perform some loop stack maintenance (Pop Loop). For more information, see "Stacks and Sequencing" on page 3-34.

The condition test occurs when the DSP is executing the last instruction in the loop (at location e, where e is the end-of-loop address). If the condition tests false, the sequencer repeats the loop, fetching the instruction from the loop-begin address, which is stored on the loop-begin stack. If the condition tests true, the sequencer terminates the loop, fetching the next instruction after the end of the loop and popping the loop stacks. For more information, see "Stacks and Sequencing" on page 3-34.

Table 3-9 and Table 3-10 show the pipeline states for loop iteration and termination.

Table 3-9. Pipelined Execution Cycles For Loop Back (Iteration)

Cycles	LA	PA	FA	AD	ID	PC
1	e[1]	e–1	e–2	e–3	e–4	e–5
2	b[2]	e	e–1	e–2	e–3	e–4
3	b+1	b	e	e–1	e–2	e–3
4	b+2	b+1	b	e	e–1	e–2
5	b+3	b+2	b+1	b	e	e–1
6	b+4[3]	b+3[3]	b+2[3]	b+1[3]	b[3]	e[3]
7	b+5	b+4	b+3	b+2	b+1	b

Note that e is the loop end instruction, and b is the loop begin instruction.
1. Termination condition tests false.
2. Loop start address is top of loop-begin stack.
3. For loops of less than six instructions (shorter than the pipeline), the pipeline retains the instructions in the loop (e through b+4). On the first iteration of such a short loop, there is a branch penalty of four Nops while the pipeline sets up for the short loop.

Table 3-10. Pipelined Execution Cycles For Loop Termination

Cycles	LA	PA	FA	AD	ID	PC
1	e[1]	e–1	e–2	e–3	e–4	e–5
2	e+1	e	e–1	e–2	e–3	e–4
3	e+2	e+1	e	e–1	e–2	e–3
4	e+3	e+2	e+1	e	e–1	e–2
5	e+4	e+3	e+2	e+1	e	e–1
6	e+5	e+4	e+3	e+2	e+1	e
7	e+6	e+5	e+4	e+3	e+2	e+1[2]

Note that e is the loop end instruction.
1. Termination condition tests true.
2. Loop aborts and loop stacks pop.

Managing Loop Stacks

To support low overhead looping, the DSP stores information for loop processing in three stacks: loop-begin stack, loop-end stack, and counter stack. The sequencer manages these stacks for loops that terminate when the counter expires (`Do/Until CE`), but does not manage these stacks for loops that terminate with a conditional `Jump`. For information on managing loop stacks, see "Stacks and Sequencing" on page 3-34.

Restrictions On Ending Loops

The sequencer's loop features (which optimize performance in many ways) limit the types of instructions that may appear at or near the end of the loop. The only absolute restriction is that the last instruction in a loop (at the loop end label) may not be a `Call`/return, a `Jump (DB)`, or a two cycle instruction.

There are restrictions on placing nested loops. For example, nested loops may not use the same end-of-loop instruction address.

Use care if using `Push Loop` or `Pop Loop` instruction inside loops. It is best to perform any stack maintenance outside of loops.

Interrupts and Sequencing

Another type of non-sequential program flow that the sequencer supports is interrupt processing. Interrupts may stem from a variety of conditions, both internal and external to the processor. In response to an interrupt, the sequencer processes a subroutine call to a predefined address, the interrupt vector. The DSP assigns a unique vector to each interrupt.

The ADSP-219x DSP core supports 16 prioritized interrupts. The four highest priority interrupts (reset, powerdown, stack, and kernel) are part of the DSP core and are common to all ADSP-219x DSPs. The rest of the interrupt levels are assignable to peripherals off the DSP core and vary with the particular DSP. For information on working with peripheral interrupts, see "ADSP-2192 Interrupts" on page E-1 and "Interrupts" on page 11-22.

The DSP supports a number of prioritized, individually-maskable external interrupts, each of which can be either level- or edge-sensitive. External interrupts occur when another device asserts one of the DSP's interrupt inputs. The DSP also supports internal interrupts. An internal interrupt can stem from stack overflows or a program writing to the interrupt's bit in the `IRPTL` register. Several factors control the DSP's response to an interrupt. The DSP responds to an interrupt request if:

- The DSP is executing instructions or is in an `Idle` state
- The interrupt is not masked
- Interrupts are globally enabled
- A higher priority request is not pending

When the DSP responds to an interrupt, the sequencer branches program execution with a call to the corresponding interrupt vector address. Within the DSP's program memory, the interrupt vectors are grouped in an area called the interrupt vector table. The interrupt vectors in this table are spaced at intervals that permit placing most interrupt service routines at the vector location—instead of branching to the actual interrupt service routine from the vector location. For a list of interrupt vector addresses and their associated latch and mask bits, see "ADSP-2192 Interrupts" on page E-1. Each interrupt vector has associated latch and mask bits. Table A-10 on page A-19 lists the latch and mask bits.

To process an interrupt, the DSP's program sequencer does the following:

1. Outputs the appropriate interrupt vector address
2. Pushes the next PC value (the return address) on to the PC stack
3. Pushes the current value of the ASTAT and MSTAT registers onto the status stack
4. Clears the appropriate bit in the interrupt latch register (IRPTL)

At the end of the interrupt service routine, the sequencer processes the return from interrupt (Rti) instruction and does following:

1. Returns to the address stored at the top of the PC stack
2. Pops this value off of the PC stack
3. Pops the status stack

All interrupt service routines should end with a return-from-interrupt (Rti) instruction. Although the interrupt vector table holds space for a reset service routine, it is important to note that DSP reset/startup routines do not operate the same as other interrupt service routines. After reset, the PC stack is empty, so there is no return address. The last instruction of the reset service routine should be a Jump to the start of the program.

If software writes to a bit in IRPTL, forcing an interrupt, the processor recognizes the interrupt in the following cycle. Four cycles of branching to the interrupt vector then follow the recognition cycle.

The DSP responds to interrupts in three stages: synchronization and latching (one cycle), recognition (one cycle), and branching to the interrupt vector (four cycles). Table 3-11, Table 3-12, and Table 3-13 show the pipelined execution cycles for interrupt processing.

Table 3-11. Pipelined Execution Cycles For Interrupt During Single-cycle Instruction

Cycles	LA	PA	FA	AD	ID	PC
1	n+4	n+3	n+2	n+1	n	n–1[1]
2	v	n+4→nop[3]	n+3→nop[3]	n+2→nop[3]	n+1→nop[3]	n[2]
3	v+1	v	n+4→nop[3]	n+3→nop[3]	n+2→nop[3]	Nop[3]
4	v+2	v+1	v	n+4→nop[3]	n+3→nop[3]	Nop
5	v+3	v+2	v+1	v	n+4→nop[3]	Nop
6	v+4	v+3	v+2	v+1	v	Nop
7	v+5	v+4	v+3	v+2	v+1	v[4]

Note that n is the single-cycle instruction, and v is the interrupt vector instruction.
1. Interrupt occurs.
2. Interrupt recognized.
3. n+1 pushed on PC stack; ASTAT/MSTAT pushed onto status stack; n+1 suppressed.
4. Interrupt vector output.

Table 3-12. Pipelined Execution Cycles For Interrupt During Instruction With Conflicting PM Data Access (Instruction Not Cached)

Cycles	LA	PA	FA	AD	ID	PC
1	n+4	n+3	n+2	n+1	n	n–1[1]
2	—	n+4	n+3	n+2	n+1	n[2]
3	v[3]	n+5→nop[4]	n+4→nop[4]	n+3→nop[4]	n+2→nop[4]	Nop[4]
4	v+1	v	n+5→nop[4]	n+4→nop[4]	n+3→nop[4]	Nop[4]
5	v+2	v+1	v	n+5→nop[4]	n+4→nop[4]	Nop
6	v+3	v+2	v+1	v	n+5→nop[4]	Nop
7	v+4	v+3	v+2	v+1	v	Nop
8	v+5	v+4	v+3	v+2	v+1	v[5]

Note that n is the single-cycle instruction, and v is the interrupt vector instruction.
1. Interrupt occurs.
2. Interrupt recognized, but not processed; PM data access.
3. Interrupt processed.
4. n+1 pushed on PC stack; ASTAT/MSTAT pushed onto status stack; n+1 suppressed.
5. Interrupt vector output.

Table 3-13. Pipelined Execution Cycles For Interrupt During Delayed Branch Instruction

Cycles	LA	PA	FA	AD	ID	PC
1	n+4	n+3	n+2	n+1	n	n–1[1]
2	j	n+4→nop	n+3→nop	n+2	n+1	n[2]
3	j+1	j	n+4→nop	n+3→nop	n+2	n+1
4	v[3]	j+1→nop[4]	j→nop[4]	n+4→nop[4]	n+3→nop[4]	n+2
5	v+1	v	j+1→nop[4]	j→nop[4]	n+4→nop[4]	Nop[3]
6	v+2	v+1	v	j+1→nop[4]	j→nop[4]	Nop[4]
7	v+3	v+2	v+1	v	j+1→nop[4]	Nop[4]
8	v+4	v+3	v+2	v+1	v	Nop[5]
9	v+5	v+4	v+3	v+2	v+1	v[6]

Note that n is the delayed branch instruction, j is the instruction at the branch address, and v is the interrupt vector instruction.

1. Interrupt occurs.
2. Interrupt recognized, but not processed.
3. Interrupt processed.
4. ASTAT/MSTAT pushed onto status stack; n+3 suppressed.
5. j pushed on PC stack; j+1 suppressed.
6. Interrupt vector output.

For most interrupts, internal and external, only one instruction is executed after the interrupt occurs (and before the two instructions are aborted) while the processor fetches and decodes the first instruction of the service routine. For more information on interrupt latency, see "ADSP-2192 Interrupts" on page E-1.

If nesting is enabled and a higher priority interrupt occurs immediately after a lower priority interrupt, the service routine of the higher priority interrupt is delayed by at least three additional cycles. For more information, see "Nesting Interrupts" on page 3-32.

Certain DSP operations that span more than one cycle hold off interrupt processing. If an interrupt occurs during one of these operations, the DSP latches the interrupt, but delays processing the interrupt. The operations that delay interrupt processing are as follows:

- A branch (`Jump` or `Call`/return) instruction and the following cycle, whether it is an instruction (in a delayed branch) or a `Nop` (in a non-delayed branch)
- The first of the two cycles used to perform a program memory data access and an instruction fetch
- The set up cycles for loops shorter than the instruction pipeline (<5 instructions).
- Any waitstates for external memory accesses
- Any external memory access that is required when the DSP does not have control of the external bus or during a host bus grant

Sensing Interrupts

The DSP supports two types of interrupt sensitivity—the signal shape that triggers the interrupt. On interrupt pins, either the input signal's edge or level can trigger an external interrupt. For more information on interrupt sensitivity and timing, see "ADSP-2192 Interrupts" on page E-1.

Masking Interrupts

The sequencer supports interrupt masking—latching an interrupt, but not responding to it. Except for the emulator (EMU), reset (RESET), and power-down interrupts, all interrupts are maskable. If a masked interrupt is latched, the DSP responds to the latched interrupt if it is later unmasked.

Interrupts can be masked globally or selectively. Bits in the ICNTL and IMASK registers control interrupt masking. Table A-11 on page A-20 lists the bits in ICNTL, and Table A-10 on page A-19 lists the bits in IMASK. These bits control interrupt masking as follows:

- **Global interrupt enable.** ICNTL, Bit 5 (GIE) directs the DSP to enable (if 1) or disable (if 0) all interrupts
- **Interrupt mask.** IMASK, Bits 15-0 direct the DSP to enable (if 1) or disable/mask (if 0) the corresponding interrupt

Except for the non-maskable interrupts and boot interrupts, all interrupts are masked at reset. For booting, the DSP automatically unmasks and uses the selected peripheral as the source for boot data.

Latching Interrupts

When the DSP recognizes an interrupt, the DSP's interrupt latch (IRPTL) register latches the interrupts and sets a bit to record that the interrupt occurred. The bits in this register indicate all interrupts that are currently pending or are being serviced. Because these registers are readable and writable, any interrupt can be set or cleared in software.

When responding to an interrupt, the sequencer clears the corresponding bit in IRPTL. During execution of the interrupt's service routine, the DSP can latch the same interrupt again while the service routine is executing.

The interrupt latch bits in IRPTL correspond to interrupt mask bits in the IMASK register. In both registers, the interrupt bits are arranged in order of priority. The interrupt priority is from 0 (highest) to 15 (lowest). Interrupt priority determines which interrupt is serviced first when more than one occurs in the same cycle. Priority also determines which interrupts are nested when the DSP has interrupt nesting enabled. For more information, see "Nesting Interrupts" on page 3-32.

Depending on the assignment of interrupts to peripherals, one event can cause multiple interrupts, and multiple events can trigger the same interrupt. For more information, see "ADSP-2192 Interrupts" on page E-1.

Stacking Status During Interrupts

To run in an interrupt driven system, programs depend on the DSP being restored to its pre-interrupt state after an interrupt is serviced. The sequencer's status stack eases the return from interrupt process by eliminating some interrupt service overhead, such as register saves and restores. For a description of stack operations, see "Stacks and Sequencing" on page 3-34.

Nesting Interrupts

The sequencer supports interrupt nesting—responding to another interrupt while a previous interrupt is being serviced. Bits in the ICNTL, IMASK, and IRPTL registers control interrupt nesting. Table A-11 on page A-20 lists the bits in ICNTL, Table A-10 on page A-19 lists the bits in IMASK and IRPTL. These bits control interrupt nesting as follows:

- **Interrupt nesting enable.** ICNTL, Bit 4 (INE), directs the DSP to enable (if 1) or disable (if 0) interrupt nesting.

- **Interrupt Mask.** IMASK, 16 Bits, selectively mask the interrupts. For each bit's corresponding interrupt, these bits direct the DSP to unmask (1=enable) or mask (0=disable) the matching interrupt.
- **Interrupt Latch.** IRPTL, 16 Bits, latch interrupts. For each corresponding interrupt, these bits indicate that the DSP has latched (1=pending) or not latched (0=pending) the matching interrupt.

When interrupt nesting is disabled, a higher priority interrupt cannot interrupt a lower priority interrupt's service routine. Other interrupts are latched as they occur, but the DSP processes them after the active routine finishes.

When interrupt nesting is enabled, a higher priority interrupt can interrupt a lower priority interrupt's service routine. Lower interrupts are latched as they occur, but the DSP processes them after the nested routines finish.

Programs should only change the interrupt nesting enable (INE) bit while outside of an interrupt service routine.

If nesting is enabled and a higher priority interrupt occurs immediately after a lower priority interrupt, the service routine of the higher priority interrupt is delayed by up to several cycles. This delay allows the first instruction of the lower priority interrupt routine to be executed, before it is interrupted.

If an interrupt re-occurs while its service routine is running and nesting is enabled, the DSP does not latch the re-occurrence in IRPTL. The DSP waits until the return from interrupt (Rti) completes, before permitting the interrupt to latch again.

Interrupting Idle

The sequencer supports placing the DSP in `Idle`—until an interrupt occurs. When executing an Idle instruction, the sequencer fetches one more instruction at the current fetch address and then suspends operation. The DSP's I/O processor is not affected by the `Idle` instruction. DMA transfers to or from internal memory continue uninterrupted.

The processor's on-chip peripherals continue to run during `Idle`. When an interrupt occurs, the processor responds normally. After two cycles used to fetch and decode the first instruction of the interrupt service routine, the processor resumes execution with the service routine.

Stacks and Sequencing

The sequencer includes five stacks: `PC` stack, loop-begin stack, loop-end stack, counter stack, and status stack. These stacks preserve information about program flow during execution branches. Figure 3-4 shows how these stacks relate to each other and to the registers that load (push) or are loaded from (pop) these stacks. Besides showing the operations that occur during explicit push and pop instructions, Figure 3-4 also indicates which stacks the DSP automatically pushes and pops when processing different types of branches: loops (`Do`/`Until`), calls (`Call`/return), and interrupts.

These stacks have differing depths. The `PC` stack is 33 locations deep; the status stack is 16 locations deep; and the loop begin, loop end, and counter stacks are 8 locations deep. A stack is full when all entries are occupied. Bits in the `SSTAT` register indicate the stack status.

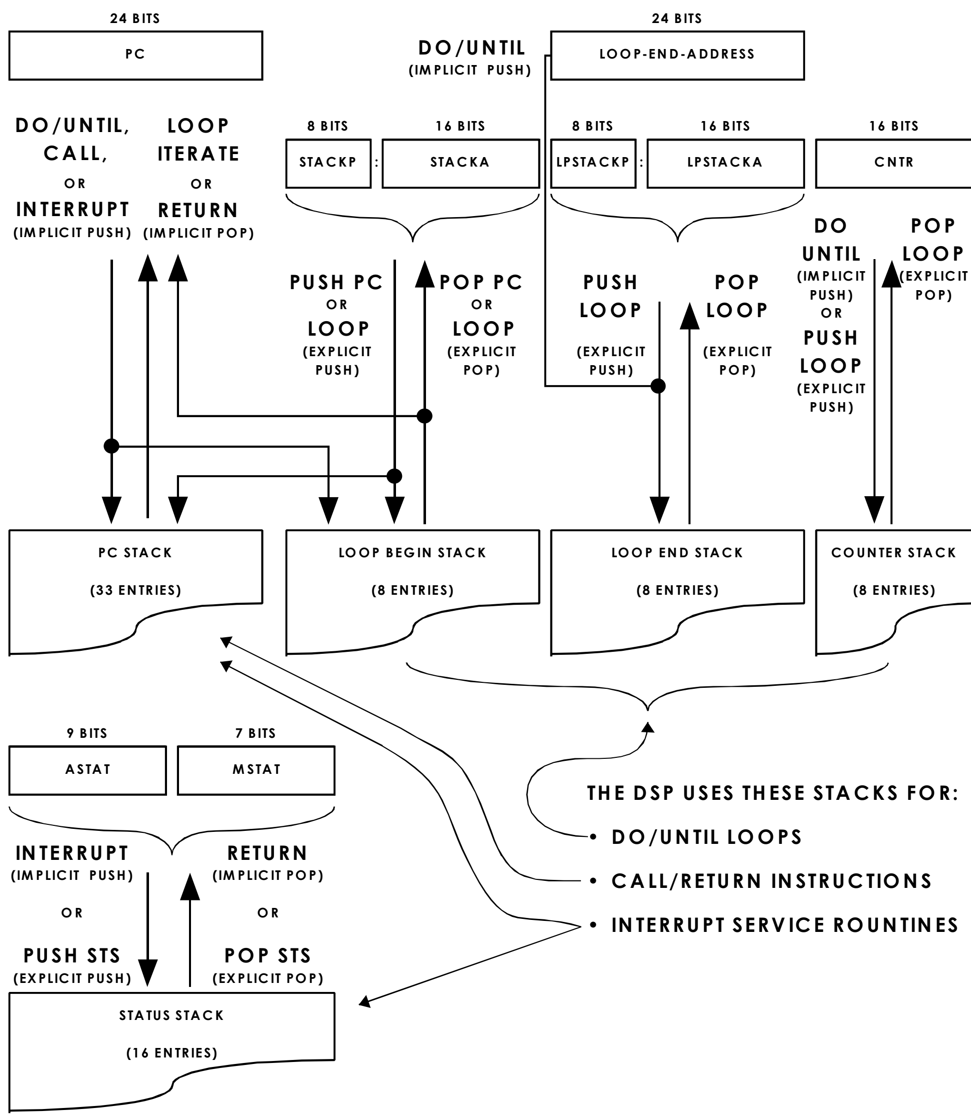


Figure 3-4. Program Sequencer Stacks

Table A-7 on page A-14 lists the bits in the SSTAT register. The SSTAT bits that indicate stack status are:

- **PC stack empty.** Bit 0 (PCSTKEMPTY) indicates that the PC stack contains at least one pushed address (if 0) or PC stack is empty (if 1).
- **PC stack full.** Bit 1 (PCSTKFULL) indicates that the PC stack contains at least one empty location (if 0) or PC stack is full (if 1).
- **PC stack level.** Bit 2 (PCSTKLVL) indicates that the PC stack contains between 3 and 28 pushed addresses (if 0) or PC stack is at or above the high-water mark—28 pushed addresses, or it is at or below the low-water mark—3 pushed addresses (if 1).
- **Loop stack empty.** Bit 4 (LPSTKEMPTY) indicates that the Loop stack contains at least one pushed address (if 0) or Loop stack is empty (if 1).
- **Loop stack full.** Bit 5 (LPSTKFULL) indicates that the Loop stack contains at least one empty location (if 0) or Loop stack is full (if 1).
- **Status stack empty.** Bit 6 (STSSTKEMPTY) indicates that the Status stack contains at least one pushed status (if 0) or Status stack is empty (if 1).
- **Stacks overflowed.** Bit 7 (STKOVERFLOW) indicates that an Overflow/underflow has not occurred (if 0) or indicates that at least one of the stacks (PC, loop, counter, status) has overflowed, or the PC or status stack has underflowed (if 1). Note that STKOVERFLOW is only cleared on reset. Loop stack underflow is not detected because it occurs only as a result of a `Pop Loop` operation.

Stack status conditions can cause a `STACK` interrupt. The stack interrupt always is generated by a stack overflow condition, but also can be generated by OR'ing together the stack overflow status (`STKOVERFLOW`) bit and stack high/low level status (`PCSTKLVL`) bit. The level bit is set when:

- The PC stack is pushed and the resulting level is at or above the high water-mark.
- The PC stack is popped and the resulting level is at or below the low water-mark.

This spill-fill mode (using the stacks' status to generate a stack interrupt) is disabled on reset. Bits in the `ICNTL` register control whether the DSP generates this interrupt based on stack status. Table A-11 on page A-20 lists the bits in the `ICNTL` register. The bits in `ICNTL` that enable the `STACK` interrupt are:

- **Global interrupt enable.** Bit 5 (`GIE`) globally disables (if 0) or enables (if 1) unmasked interrupts
- **PC stack interrupt enable.** Bit 10 (`PCSTKE`) directs the DSP to disable (if 0) or enable (if 1) spill-fill mode—OR'ing of stack status—to generate the `STACK` interrupt.

When switching on spill-fill mode, a spurious (low) stack level interrupt may occur (depending on the level of the stack). In this case, the interrupt handler should push some values on the stack to raise the level above the low level threshold.

Values move on (push) or off (pop) the stacks through implicit and explicit operations. Implicit stack operations are stack accesses that the DSP performs while executing a branch instruction (`Call`/return, `Do`/`Until`) or while responding to an interrupt. Explicit stack operations are stack accesses that the DSP performs while executing the stack instructions (`Push`, `Pop`).

As shown in Figure 3-4, the source for the pushed values and destination for the pop value differs depending on whether the stack operations is implicit or explicit.

In implicit stack operations, the DSP places values on the stacks from registers (PC, CNTR, ASTAT, MSTAT) and from calculated addresses (end-of-loop, PC+1). For example a Call/return instruction directs the DSP to branch execution to the called subroutine and push the return address (PC+1) onto the PC stack. The matching return from subroutine instruction (Rts) causes the DSP to pop the return address off of the PC stack and branch execution to the address following the Call.

A second instruction that makes the DSP perform implicit stack operations is the Do/Until instruction. It takes the following steps to set up a Do/Until loop:

- Load the loop count into the CNTR register
- Initiate the loop with a Do/Until instruction
- Terminate the loop with an end-of-loop label

When executing a Do/Until instruction, the DSP performs the following implicit stack operations:

- Pushes the loop count from the CNTR register onto the counter stack
- Pushes the start-of-loop address from the PC onto the loop start stack
- Pushes the end-of-loop address from the end-of-loop label onto the loop-end stack

When the count in the top location of the counter stack expires, the loop terminates, and the DSP pops the three loop stacks, resuming execution at the address after the end of the loop. The count is decremented on the stack, *not* in the CNTR register.

A third condition/instruction that makes the DSP perform implicit stack operations is an interrupt/return instruction. When interrupted, the DSP pushes the PC onto the PC stack, pushes the ASTAT and MSTAT registers onto the status stack, and branches execution to the interrupt service routine's location (vector). At the end of the routine, the return from interrupt instruction directs the DSP to pop these stacks and branch execution to the instruction after the interrupt (PC+1).

In explicit stack operations, a program's access to the stacks goes through a set of registers: STACKP, STACKA, LPSTACKP, LPSTACKA, CNTR, ASTAT, and MSTAT. A Pop instruction retrieves the value or address from the corresponding stack (PC, Loop, or Sts) and places that value in the corresponding register (as shown in Figure 3-4 on page 3-35). A Push instruction takes the value or address from the register and puts it on the corresponding stack. Programs should use explicit stack operations for stack maintenance, such as managing the stacks when exiting a Do/Until loop with a conditional Jump.

Conditional Sequencing

The sequencer supports conditional execution with conditional logic that appears in Figure 3-4 on page 3-35. This logic evaluates conditions for conditional (If) instructions and loop (Do/Until) terminations. The conditions are based on information from the arithmetic status registers (ASTAT), the condition code register (CCODE), the flag inputs, and the loop counter. For more information on arithmetic status, see "Using Computational Status" on page 2-16.

Each condition that the DSP evaluates has an assembler mnemonic. The condition mnemonics for conditional instructions appear in Table 3-14. For most conditions, the sequencer can test both true and false states. For example, the sequencer can evaluate ALU equal-to-zero (EQ) and ALU not-equal-to-zero (NE).

To test conditions that do not appear in Table 3-14, a program can use the Test Bit (`Tstbit`) instruction to test bit values loaded from status registers. For more information, see the ADSP-219x *DSP Instruction Set Reference*.

Table 3-14. If Condition and Do/Until Termination Condition Logic

Syntax	Status Condition	True If:	Do/Until	If cond
EQ	Equal Zero	AZ = 1	✘	✓
NE	Not Equal Zero	AZ = 0	✘	✓
LT	Less Than Zero	AN .XOR. AV = 1	✘	✓
GE	Greater Than or Equal Zero	AN .XOR. AV = 0	✘	✓
LE	Less Than or Equal Zero	(AN .XOR. AV) .OR. AZ = 1	✘	✓
GT	Greater Than Zero	(AN .XOR. AV) .OR. AZ = 0	✘	✓
AC	ALU Carry	AC = 1	✘	✓
Not AC	Not ALU Carry	AC = 0	✘	✓
AV	ALU Overflow	AV = 1	✘	✓
Not AV	Not ALU Overflow	AV = 0	✘	✓
MV	MAC Overflow	MV = 1	✘	✓
Not MV	Not MAC Overflow	MV = 0	✘	✓

Table 3-14. If Condition and Do/Until Termination Condition Logic

Syntax	Status Condition	True If:	Do/Until	If cond
SWCOND	Compares value in CCODE register with following DSP conditions: PF0-13 inputs Hi, AS, SV	CCODE=SWCOND	✘	✓
Not SWCOND	Compares value in CCODE register with following DSP conditions: PF0-13 inputs Lo, Not AS, Not SV	CCODE= Not SWCOND	✘	✓
CE	Counter Expired	loop counter = 0	✓	✘
Not CE[1]	Counter Not Expired	loop counter = Not 0	✘	✓
Forever	Always (Do)		✓	✘
True	Always (If)		✘	✓

1 Executing this instruction decrements the CNTR register.

The two conditions that do not have complements are `CE`/`Not CE` (loop counter expired/not expired) and `True`/`Forever`. The context of these condition codes determines their interpretation. Programs should use `True` and `Not CE` in conditional (`If`) instructions. Programs should use `Forever` and `CE` to specify loop (`Do`/`Until`) termination. A `Do Forever` instruction executes a loop indefinitely, until an interrupt, jump, or reset intervenes.

There are some restrictions on how programs may use conditions in `Do`/`Until` loops. For more information, see "Restrictions On Ending Loops" on page 3-24.

Sequencer Instruction Summary

Table 3-15 lists the program sequencer instructions and how they relate to SSTAT flags. For more information on assembly language syntax, see the ADSP-219x *DSP Instruction Set Reference*. In Table 3-15, note the meaning of the following symbols:

- **Reladdr#** indicates a PC-relative address of #number of bits
- **Addr24** indicates an absolute 24-bit address.
- **Ireg** indicates an Index (I) register in either DAG.
- **Imm4** indicates an immediate 4-bit value.
- **Addr24** indicates an absolute 24-bit address.
- * indicates the flag may be set or cleared, depending on results of instruction.
- – indicates no effect.

Table 3-15. Sequencer Instruction Summary

Instruction	SSTAT Status Flags						
	LE	LF	PE	PF	PL	SE	SO
Do <Reladdr12> Until [CE, Forever];	*	*	–	–	–	–	*
[If Cond] Jump <Reladdr13> [(DB)];	–	–	–	–	–	–	–
Call <Reladdr16> [(DB)];	–	–	*	*	*	–	*
Jump <Reladdr16> [(DB)];	–	–	–	–	–	–	–
[If Cond] Lcall <Addr24>;	–	–	*	*	*	–	*
[If Cond] Ljump <Addr24>;	–	–	–	–	–	–	–
[If Cond] Call <Ireg> [(DB)];	–	–	*	*	*	–	*
[If Cond] Jump <Ireg> [(DB)];	–	–	–	–	–	–	–
[If Cond] Rti [(DB)];	–	–	*	*	*	*	–
[If Cond] Rts [(DB)];	–	–	*	*	*	–	–
Push |PC, Loop, Sts|;	*	*	*	*	*	*	*
Pop |PC, Loop, Sts|;	*	*	*	*	*	*	*
Flush Cache;	–	–	–	–	–	–	–
Setint <Imm4>;	–	–	*	*	*	*	*
Clrint <Imm4>;	–	–	–	–	–	–	–
Nop;	–	–	–	–	–	–	–
Idle;	–	–	–	–	–	–	–
Ena | TI, MM, AS, OL, BR, SR, BSR, INT | ;	–	–	–	–	–	–	–
Dis | TI, MM, AS, OL, BR, SR, BSR, INT | ;	–	–	–	–	–	–	–
Abbreviations for SSTAT Flags: LE=LPSTKEMPTY, LF=LPSTKFULL, PE=PCSTKEMPTY, PF=PCSTKFULL, PL=PCSTKLVL, SE=STSSTKEMPTY, SO=STKOVERFLOW							

4 DATA ADDRESS GENERATORS

Overview

The DSP's Data Address Generators (DAGs) generate addresses for data moves to and from Data Memory (DM) and Program Memory (PM). By generating addresses, the DAGs let programs refer to addresses indirectly, using a DAG register instead of an absolute address. The DAG architecture, which appears in Figure 4-1, supports several functions that minimize overhead in data access routines. These functions include:

- **Supply address and post-modify**—provides an address during a data move and auto-increments the stored address for the next move.
- **Supply pre-modified address**—provides a modified address during a data move without incrementing the stored address.
- **Modify address**—increments the stored address without performing a data move.
- **Bit-reverse address**—provides a bit-reversed address during a data move without reversing the stored address.

As shown in Figure 4-1, each DAG has five types of registers. These registers hold the values that the DAG uses for generating addresses. The types of registers are:

- **Index registers (I0-I3 for DAG1 and I4-I7 for DAG2).** An index register holds an address and acts as a pointer to memory. For example, the DAG interprets `DM(I0)` and `PM(I4)` syntax in an instruction as addresses.

- **Modify registers (M0-M3 for DAG1 and M4-M7 for DAG2).** A modify register provides the increment or step size by which an index register is pre- or post-modified during a register move. For example, the `dm(I0+=M1)` instruction directs the DAG to output the address in register `I0` then modify the contents of `I0` using the `M1` register.

- **Length and Base registers (L0-L3 and B0-B3 for DAG1 and L4-L7 and B4-B7 for DAG2).** Length and base registers setup the range of addresses and the starting address for a circular buffer. For more information on circular buffers, see "Addressing Circular Buffers" on page 4-11.

- **DAG Memory Page registers (DMPG1 for DAG1 and DMPG2 for DAG2).** Page registers set the upper eight bits address for DAG memory accesses; the 16-bit Index and Base registers hold the lower 16 bits. For more information on about DAG page registers and addresses from the DAGs, see "DAG Page Registers (DMPGx)" on page 4-6.

Do not assume that the L registers are automatically initialized to zero for linear addressing. The I, M, L, and B registers contain random values following DSP reset. For each I register used, programs must initialize the corresponding L registers to the appropriate value—either 0 for linear addressing or the buffer length for circular buffer addressing.

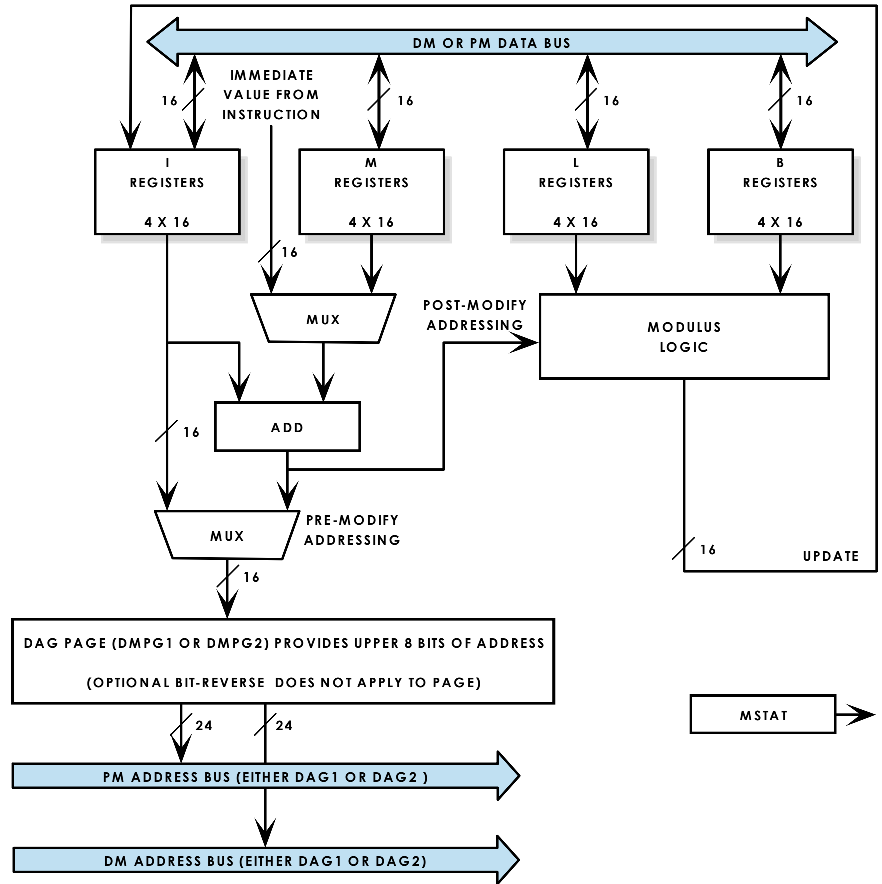


Figure 4-1. Data Address Generator (DAG) Block Diagram

On previous 16-bit, fixed-point DSPs (ADSP-218x family), the DAG registers are 14-bits wide. Because the ADSP-219x DAG registers are 16-bits wide, the DAGs do not need to perform the zero padding on I and L register writes to memory or the sign extension on M register writes to memory that is required for previous ADSP-218x family DSPs.

Setting DAG Modes

The MSTAT register controls the operating mode of the DAGs. Table A-6 on page A-11 lists all the bits in MSTAT. The following bits in MSTAT control Data Address Generator modes:

- **Bit-reverse addressing enable.** Bit 1 (BIT_REV) enables bit-reversed addressing (if 1) or disables bit-reversed addressing (if 0) for DAG1 Index (I0-I3) registers.
- **Secondary registers for DAG.** Bit 6 (SEC_DAG) selects the corresponding secondary register set (if 1) or selects the corresponding primary register set—the set that is available at reset—(if 0).

Secondary (Alternate) DAG Registers

Each DAG has an secondary register set. To facilitate fast context switching, the DSP includes secondary register sets for data, results, and data address generator registers. The SEC_DAG bit in the MSTAT register controls when secondary DAG registers become accessible. While inaccessible, the contents of secondary registers are not affected by DSP operations. Figure 4-2 on page 4-5 shows the DAG's primary and secondary register sets.

ⓘ The secondary register sets for the DAGs are described in this section. For more information on secondary data and results registers, see "Secondary (Alternate) Data Registers" on page 2-59.

ⓘ There are no secondary DMPGx registers. Changing between primary and secondary DAG registers does not affect the DMPGx register settings.

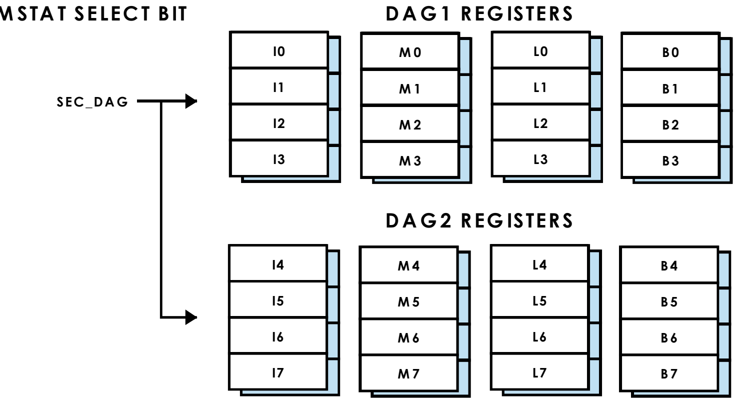


Figure 4-2. Data Address Generator Primary and Alternate Registers

System power-up and reset enable the primary set of DAG address registers. To enable or disable the secondary address registers, programs set or clear the `SEC_DAG` bit in `MSTAT`. The instruction set provides three methods for swapping the active set. Each method incurs a latency, which is the delay between the time the instruction affecting the change executes until the time the change takes effect and is available to other instructions. Table A-3 on page A-5 shows the latencies associated with each method.

When switching between primary and secondary DAG registers, the program needs to account for the latency associated with the method used. For example, after the `MSTAT = data12;` instruction, a minimum of three cycles of latency occur before the mode change takes effect. So for this method, the program must issue at least three instructions after `MSTAT = 0x20;` before attempting to use the other set of DAG registers.

The Ena/Dis mode instructions are more efficient for enabling and disabling DSP modes because these instructions incur no cycles of effect latency. For example:

```
CCODE = 0x9; Nop;
If SWCOND Jump do_data;/* Jump to do_data */
do_data:
   Ena SEC_REG;        /* Switch to 2nd Dregs */
   Ena SEC_DAG;        /* Switch to 2nd DAGs */
   AX0 = DM(buffer);   /* if buffer empty, go */
   AR = Pass AX0;      /* right to fill and */
   If EQ Jump fill;    /* get new data */
   Rti;
fill:                  /* fill routine */
   Nop;
buffer:                /* buffer data */
   Nop;
```

On previous 16-bit, fixed-point DSPs (ADSP-218x family), there are no secondary DAG registers.

Bit-Reverse Addressing Mode

The BIT_REV bit in the MSTAT register enables bit-reverse addressing mode—outputting addresses in bit-reversed order. When BIT_REV is set (1), the DAG bit-reverses 16-bit addresses output from DAG1 index registers—I0, I1, I2, and I3. Bit-reverse addressing mode affects post-modify operations. For more information, see “Addressing With Bit-Reversed Addresses” on page 4-15.

DAG Page Registers (DMPGx)

The DAGs and their associated page registers generate 24-bit addresses for accessing the data needed by instructions. For data accesses, the DSP’s unified memory space is organized into 256 pages, with 64K locations per page. The page registers provide the eight MSBs of the 24-bit address, specifying the page on which the data is located.

The DAGs provide the sixteen LSBs of the 24-bit address, specifying the exact location of the data on the page.

- The DMPG1 page register is associated with DAG1 (registers I0–I3) indirect memory accesses and immediate addressing.
- The DMPG2 page register is associated with DAG2 (registers I4–I7) indirect memory accesses.

At power up, the DSP initializes both page registers to 0x0. Initializing page registers is only necessary when the data is located on a page other than the current page. Programs should set the corresponding page register when initializing a DAG index register to set up a data buffer.

For example,

```
DMPG1 = 0x12; /* set page register */
              /* or the syntax: DMPG1 = page(data_buffer);
                 for relative addressing */
I2 = 0x3456;  /* init data buffer; 24b addr=0x123456 */
L2 = 0;       /* define linear buffer */
M2 = 1;       /* increment address by one */
              /* two stall cycles inserted here */
DM(I2 += M2) = AX0;  /* write data to buffer and update I2 */
```

DAG register (DMPGx, Ix, Mx, Lx, Bx) loads can incur up to two stall cycles when a memory access based on the initialized register immediately follows the initialization.

To avoid stall cycles, programs could perform the following memory access sequence:

```
I2 = 0x3456;   /* init data buffer; 24b addr=0x123456 */
L2 = 0;        /* define linear buffer */
M2 = 1;        /* increment address by one */
DMPG1 = 0x12;  /* set page register */
               /* or use the syntax: DMPG1 = page(data_buffer);
                  for relative addressing */
AX0 = 0xAAAA;
AR = AX0 - 1;
DM(I2 += M2) = AR;  /* write data to buffer and update I2 */
```

Typically, programs load both page registers with the same page value (0-255), but programs can increase memory flexibility by loading each with a different page value. For example, by loading the page registers with different page values, programs could perform high-speed data transfers between pages.

Changing the contents of the DAG page registers is not automatic and requires explicit programming.

Using DAG Status

As described in "Addressing Circular Buffers" on page 4-11, the DAGs can provide addressing for a constrained range of addresses, repeatedly cycling through this data (or buffer). A buffer overflow (or wrap around) occurs each time the DAG circles past the buffer's base address.

Unlike the computational units and program sequencer, the DAGs do not generate status information. So, the DAGs do not provide buffer overflow information when executing circular buffer addressing. If a program requires status information for the circular buffer overflow condition, the program should implement an address range checking routine to trap this condition.

DAG Operations

The DSP's DAGs perform several types of operations to generate data addresses. As shown in Figure 4-1 on page 4-3, the DAG registers and the MSTAT register control DAG operations. The following sections provide details on DAG operations:

- "Addressing with DAGs" on page 4-9
- "Addressing Circular Buffers" on page 4-11
- "Modifying DAG Registers" on page 4-19

An important item to note from Figure 4-1 is that each DAG automatically uses its DAG memory page (DMPGx) register to include the page number as part of the output address. By including the page, DAGs can generate addresses for the DSP's entire memory map. For details on these address adjustments, see "DAG Page Registers (DMPGx)" on page 4-6.

Addressing with DAGs

The DAGs support two types of modified addressing—generating an address that is incremented by a value or a register. In pre-modify addressing, the DAG adds an offset (modifier), either an M register or an immediate value, to an I register and outputs the resulting address. Pre-modify addressing does not change (or update) the I register. The other type of modified addressing is post-modify addressing. In post-modify addressing, the DAG outputs the I register value unchanged, then the DAG adds an M register or immediate value, updating the I register value. Figure 4-3 compares pre- and post-modify addressing.

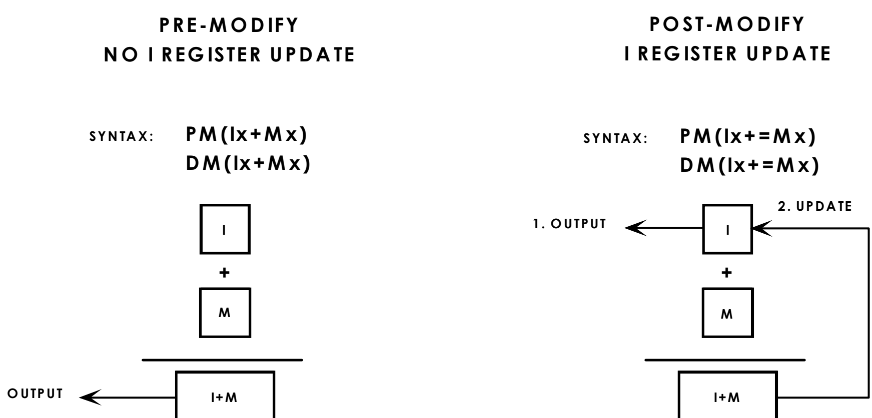


Figure 4-3. Pre-Modify and Post-Modify Operations

The difference between pre-modify and post-modify instructions in the DSP's assembly syntax is the operator that appears between the index and modify registers in the instruction. If the operator between the I and M registers is += (plus-equals), the instruction is a post-modify operation. If the operator between the I and M registers is + (plus), the instruction is a pre-modify without update operation. The following instruction accesses the program memory location indicated by the value in I7 and writes the value I7 plus M6 to the I7 register:

```
AX0 = PM(I7+=M6); /* Post-modify addressing with update */
```

By comparison, the following instruction accesses the program memory location indicated by the value I7 plus M6 and does not change the value in I7:

```
AX0 = PM(I7+M6); /* Pre-modify addressing without update */
```

Modify (M) registers can work with any index (I) register in the same DAG (DAG1 or DAG2). For a list of I and M registers and their DAGs, see Figure 4-2 on page 4-5.

On previous 16-bit, fixed-point DSPs (ADSP-2180 family), the assembly syntax uses a comma between the DAG registers (`I,M` indicates post-modify) to select the DAG operation. While the legacy support in the ADSP-219x assembler permits this syntax, updating ported code to use the ADSP-219x syntax (`I+M` for premodify and `I+=M` for post-modify) is advised.

Instructions can use a signed 8-bit number (immediate value), instead of an M register, as the modifier. For all single data access operations, modify values can be from an M register or an 8-bit immediate value. The following example instruction accepts up to 8-bit modifiers:

```
AX0=DM(I1+0x40);     /* DM address = I1+0x40 */
```

Instructions that combine DAG addressing with computations do not accept immediate values for the modifier. In these instructions (multifunction computations), the modify value must come from an M register:

```
AR=AX0+AY0,PM(I4+=m5)=AR; /* PM address = I4, I4=I4+M5 */
```

Note that pre- and post-modify addressing operations do not change the memory page of the address. For more information, see "DAG Page Registers (DMPGx)" on page 4-6.

Addressing Circular Buffers

The DAGs support addressing circular buffers. The DAG steps repeatedly through a range of addresses containing data, "wrapping around" in a circular pattern. To address a circular buffer, the DAG steps the index pointer (I register) through the buffer, post-modifying and updating the index on each access with a positive or negative modify value (M register or immediate value). If the index pointer falls outside the buffer, the DAG subtracts or adds the length of the buffer from or to the value, wrapping the index pointer back to the start of the buffer.

The DAG's support for circular buffer addressing appears in Figure 4-1 on page 4-3. An example of circular buffer addressing appears in Figure 4-4.

The starting address that the DAG wraps around is called the buffer's base address (B register). There are no restrictions on the value of the base address for a circular buffer.

Circular buffering may only use post-modify addressing. The DAG's architecture, as shown in Figure 4-1 on page 4-3, cannot support pre-modify addressing for circular buffering, because circular buffering requires that the index be updated on each access.

Do not place the index pointer for a circular buffer such that it crosses a memory page boundary during post-modify addressing. All memory locations in a circular buffer must reside on the same memory page. For more information on the DSP's memory map, see "Memory" on page 5-1.

As shown in Figure 4-4, programs use the following steps to set up a circular buffer:

1. Load the memory page address into the selected DAG's `DMPGx` register. This operation is needed only once per page change in a program.

2. Load the starting address within the buffer into an I register in the selected DAG.

3. Load the modify value (step size) into an M register in the corresponding DAG as the I register. For corresponding registers list, see Figure 4-2 on page 4-5.

4. Load the buffer's length into the L register that corresponds to the I register. For example, `L0` corresponds to `I0`.

5. Load the buffer's base address into the B register that corresponds to the I register. For example, `B0` corresponds to `I0`.

After this setup, the DAGs use the modulus logic in Figure 4-1 on page 4-3 to process circular buffer addressing.

```
.section/dm seg_data;
.var coeff_buffer[11] = 0,1,2,3,4,5,6,7,8,9,10;
.section/pm seg_code;
dmpg1 = page(coeff_buffer);/* set the memory page */
i0 = coeff_buffer;         /* set the current addr */
m1 = 4;                    /* set the modify value */
l0 = length(coeff_buffer); /* if l = 0 buffer is linear */
ax0 = i0;                  /* copy the base addr into ax0 */
reg(b0) = ax0;             /* set the buffer's base addr */
ar = ax1 and ay0;
ar = dm(i0 += m1);         /* read 1st buffer location */

cntr = 11; do my_cir_buffer until ce;
                           /* sets up a loop accessing the buffer */
ax0 = dm(i0,m1);           /* access using post modify addressing */
nop;                       /* other instructions in the loop */
my_cir_buffer: nop;        /* end of my_cir_buffer loop */
```


Figure 4-4. Circular Data Buffers

On the first post-modify access to the buffer, the DAG outputs the I register value on the address bus then modifies the address by adding the modify value. If the updated index value is within the buffer length, the DAG writes the value to the I register. If the updated value is outside the buffer length, the DAG subtracts (positive) or adds (negative) the L register value before writing the updated index value to the I register.

In equation form, these post-modify and wrap around operations work as follows:

- If M is positive:

  $\mathbf{I_{new} = I_{old} + M}$ if $I_{old} + M <$ Buffer base + length (end of buffer)

  $\mathbf{I_{new} = I_{old} + M - L}$ if $I_{old} + M \geq$ Buffer base + length (end of buffer)

- If M is negative:

  $\mathbf{I_{new} = I_{old} + M}$ if $I_{old} + M \geq$ Buffer base (start of buffer)

  $\mathbf{I_{new} = I_{old} + M + L}$ if $I_{old} + M <$ Buffer base (start of buffer)

The DAGs use all types of DAG registers for addressing circular buffers. These registers operate as follows for circular buffering:

- The index (I) register contains the value that the DAG outputs on the address bus.
- The modify (M) register contains the post-modify amount (positive or negative) that the DAG adds to the I register at the end of each memory access. The M register can be any M register in the same DAG as the I register. The modify value also can be an immediate value instead of an M register. The size of the modify value, whether from an M register or immediate, must be less than the length (L register) of the circular buffer.

- The length (L) register sets the size of the circular buffer and the address range that the DAG circulates the I register through. L is positive and cannot have a value greater than $2^{16} - 1$. If an L register's value is zero, its circular buffer operation is disabled.
- The base (B) register, or the B register plus the L register, is the value that the DAG compares the modified I value with after each access.

On previous 16-bit, fixed-point DSPs (ADSP-218x family), the DAGs do not have B registers. When porting code that uses circular buffer addressing, add the instructions needed for loading the ADSP-219x B register that is associated with the corresponding circular buffer.

Addressing With Bit-Reversed Addresses

Programs need bit-reversed addressing for some algorithms (particularly FFT calculations) to obtain results in sequential order. To meet the needs of these algorithms, the DAG's bit-reverse addressing feature permits repeatedly subdividing data sequences and storing this data in bit-reversed order.

Bit-reversed address output is available on DAG1, while DAG2 always outputs its address bits in normal, Big Endian format. Because the two DAGs operate independently, programs can use them in tandem, with one generating sequentially ordered addresses and the other generating bit-reversed addresses, to perform memory reads and writes of the same data.

To use bit-reversed addressing, programs set the `BIT_REV` bit in `MSTAT` (`Ena BIT_REV`). When enabled, DAG1 outputs all addresses generated by its index registers (`I0`–`I3`) in bit-reversed order. The reversal applies only to the address value DAG1 outputs, not to the address value stored in the index register, so the address value is stored in Big Endian format. Bit-reversed mode remains in effect until disabled (`Dis BIT_REV`).

Bit reversal operates on the binary number that represents the position of a sample within an array of samples. Using 3-bit addresses, Table 4-1 shows the position of each sample within an array before and after the bit-reverse operation. Sample 0x4 occupies position b#100 in sequential order and position b#001 in bit-reversed order. Bit reversing transposes the bits of a binary number about its midpoint, so b#001 becomes b#100, b#011 becomes b#110, and so on. Some numbers, like b#000, b#111, and b#101, remain unchanged and retain their original position within the array.

Table 4-1. 8-point array sequence before and after bit reversal

Sequential Order		Bit Reversed Order	
Sample (hexadecimal)	Binary	Binary	Sample (hexadecimal)
0x0	b#000	b#000	0x0
0x1	b#001	b#100	0x4
0x2	b#010	b#010	0x2
0x3	b#011	b#110	0x6
0x4	b#100	b#001	0x1
0x5	b#101	b#101	0x5
0x6	b#110	b#011	0x3
0x7	b#111	b#111	0x7

Bit-reversing the samples in a sequentially ordered array scrambles their positions within the array. Bit-reversing the samples in a scrambled array restores their sequential order within the array.

In full 16-bit reversed addressing, bits 7 and 8 of the 16-bit address are the pivot points for the reversal:

Normal	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit-reversed	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Fast Fourier Transform (FFT) algorithm is a special case for bit-reversal. FFT operations often need only a few address bits reversed. For example, a a 16-point sequence requires four reversed bits, and a 1024-bit sequence requires ten reversed bits. Programs can bit-reverse address values less than 16-bits—which reverses a specified number of LSBs only. Bit-reversing less than the full 16-bit index register value requires that the program adds the correct modify value to the index pointer after each memory access to generate the correct bit-reversed addresses.

To set up bit-reversed addressing for address values < 16 bits, determine:

1. The **number of bits to reverse** (`N`)—permits calculating the modify value
2. The **starting address of the linear data buffer**—this address must be zero or an integer multiple of the number of bits to reverse (starting address = `0, N, 2N, …`)
3. The **first bit-reversed address** that the DAG outputs—the buffer's starting address with the `N` LSBs bit-reversed
4. The **initialization value for the index register**—the bit-reversed value of the first bit-reversed address the DAG outputs
5. The **modify register value** for updating (correcting) the index pointer after each memory access—calculated from the formula: $Mreg = 2^{(16-N)}$.

The following example, sets up bit-reversed addressing that reverses the eight address LSBs (`N = 8`) of a data buffer with a starting address of `0x0020` (`4N`). Following the described steps, the factors to determine are:

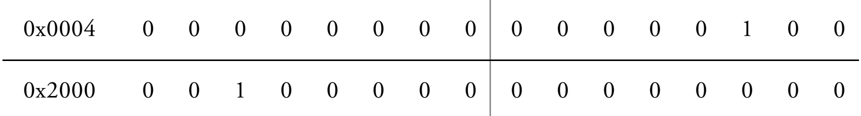
1. The **number of bits to reverse** (`N`)—eight bits (from description)

2. The **starting address of the linear data buffer**—`0x0020` (`4N`) (from description)

3. The **first bit-reversed address** that the DAG outputs—This value is the buffer's starting address (`0x0020`) with bits 7–0 reversed: `0x0004`.

0x0020	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0x0004	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

4. The **initialization value for the index register**—This is the first bit-reversed address DAG1 outputs (`0x0004`) with bits 15–0 reversed: `0x2000`.



0x0004	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0x2000	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

5. The **modify register value** for updating (correcting) the index pointer after each memory access—This is 2^{16-N} which evaluates to 2^8 or `0x0100`.

Listing 4-1 implements this example in assembly code.

Listing 4-1. Bit-reversed addressing, 8 LSBs

```
br_adds: I4=read_in;        /* DAG2 pointer to input samples */
   I0=0x0200;             /* Base address of bit_rev output */
   M4=1;                    /* DAG2 increment by 1 */
   M0=0x0100;             /* DAG1 increment for 8-bit rev. */
   L4=0;                    /* Linear data buffer */
   L0=0;                    /* Linear data buffer */
   CNTR=8;                  /* 8 samples */
   Ena BIT_REV;            /* Enable DAG1 bit reverse mode */
   Do brev Until CE;
      AY1=DM(I4+=M4);       /* Read samples sequentially */
     brev: DM(I0+=M0)=AY1; /* Write results nonsequentially */
   Dis BIT_REV;           /* Disable DAG1 bit reverse mode */
   Rts;                     /* Return to calling routine */
read_in:                  /* input buffer, could be .extern */
  Nop;
```

Modifying DAG Registers

The DAGs support an operation that modifies an address value in an index register without outputting an address. The operation, address modify, is useful for maintaining pointers.

The `Modify` instruction modifies addresses in any DAG index register (`I0-I7`) without accessing memory. If the I register's corresponding B and L registers are set up for circular buffering, a `Modify` instruction performs the specified buffer wrap around (if needed). The syntax for `Modify` is similar to post-modify addressing (index+=modifier). `Modify` accepts either a signed 8-bit immediate values or an M register as the modifier. The following example adds 4 to `I1` and updates `I1` with the new value:

```
Modify(I1+=4);
```

DAG Register Transfer Restrictions

DAG I, M, and L registers are part of the DSP's Register Group 1 (`Reg1`), 2 (`Reg2`), and 3 (`Reg3`) register sets; the B registers are in register memory space. Programs may load the DAG registers from memory, from another data register, or with an immediate value. Programs may store DAG registers' contents to memory or to another data register.

While instructions to load and use DAG registers may be sequential, the DAGs insert stall cycles for sequences of instructions that cause instruction pipeline conflicts. The two types of conflicts are:

- Using an I register (or its corresponding L or B registers) within two cycles of loading the I register (or its corresponding L or B registers)
- Using an M register within two cycles of loading the M register
- Using an I register within two cycles of performing the `modify` instruction

The following code examples and comments demonstrate the conditions under which the DAG inserts stall cycles. These examples also show how to avoid these stall conditions.

```
/* The following sequence of loading and using the DAG
   registers does NOT force the DAG to insert stall cycles. */
I0=0x1000;
M0=1;
L0=0xF;
Reg(B0)=AX0;
AR = AX0 +AY0;
MR = MX0 * MY0 (SS);
AX1=DM(I0+=M0);

/* This sequence of loading and using the DAG registers
   FORCES the DAG to insert two stall cycles. */
M0=1;
L0=0xF;
Reg(b0)=ax0;
I0=0x1000;
```

```
AX1=DM(I0+=M0);  /* DAG inserts two stall cycles here
                    until i0 can be used */

/* This sequence of loading and using the DAG registers
   FORCES the DAG to insert two stall cycles. */
I0=0x1000;
L0=0xF;
Reg(B0)=AX0;
M0=1;
AX1=DM(I0+=M0);  /* DAG inserts two stall cycles here
                    until m0 can be used */

/* This sequence of loading and using the DAG registers
   FORCES the DAG to insert one stall cycle. */
M0=1;
L0=0xF;
I0=0x1000;
Reg(B0)=AX0;
AR = AX0 + AY0;
AX1=DM(I0+=M0);  /* DAG inserts one stall cycle here
                   until i0 (corresponds to b0) can be used */
```

DAG Instruction Summary

Table 4-2 lists the DAG instructions. For more information on assembly language syntax, see the ADSP-219x *DSP Instruction Set Reference*. In Table 4-2, note the meaning of the following symbols:

- **Dreg, Dreg1, Dreg2** indicate any register file location (Register Group).
- **Reg1, Reg2, Reg3, or Reg** indicate Register Group 1, 2, 3 or any register.
- **Ia and Mb** indicate DAG1 I and M registers.
- **Ic and Md** indicate DAG2 I and M registers.
- **Ireg and Mreg** indicate I and M registers in either DAG.
- **Imm# and Data#** indicate immediate values or data of the # of bits.

Table 4-2. DAG Instruction Summary

<table>
<tr><th>Instruction</th></tr>
<tr><td>|DM(Ia += Mb), DM(Ic += Md)| = Reg;</td></tr>
<tr><td>Reg = |DM(Ia += Mb), DM(Ic += Md)|;</td></tr>
<tr><td>|DM(Ia + Mb), DM(Ic + Md)| = Reg;</td></tr>
<tr><td>Reg = |DM (Ia + Mb), DM (Ic + Md)|;</td></tr>
<tr><td>|PM(Ia += Mb), PM(Ic += Md)| = Reg;</td></tr>
<tr><td>Reg = |PM(Ia += Mb), PM(Ic += Md)|;</td></tr>
<tr><td>|PM(Ia + Mb), PM(Ic + Md)| = Reg;</td></tr>
<tr><td>Reg = |PM(Ia + Mb), PM(Ic + Md)|;</td></tr>
<tr><td>DM(Ireg1 += Mreg1) = |Ireg2, Mreg2, Lreg2|, |Ireg2, Mreg2, Lreg2| = Ireg1;</td></tr>
<tr><td>Dreg = DM(Ireg += <Imm8>);</td></tr>
<tr><td>DM(Ireg += <Imm8>) = Dreg;</td></tr>
<tr><td>Dreg = DM(Ireg + <Imm8>);</td></tr>
<tr><td>DM(Ireg + <Imm8>) = Dreg;</td></tr>
<tr><td>|DM(Ia += Mb), DM (Ic += Md)| = <Data16>;</td></tr>
<tr><td>|PM (Ia += Mb), PM (Ic += Md)| = <Data24>:24;</td></tr>
<tr><td>|Modify (Ia += Mb), Modify (Ic += Md)|;</td></tr>
<tr><td>Modify (Ireg += <Imm8>);</td></tr>
</table>

5 MEMORY

Overview

Each DSP core in the ADSP-2192 contains large internal memory. This chapter describes the DSP's memory and how to use it. The two DSP cores also have shared memory and memory-mapped registers. For information on using the shared memory, see "Dual DSP Cores" on page 6-1.

There are 140K words of internal memory on the ADSP-2192. Within this space, the P0 DSP core has 80K words of SRAM and 4K words of ROM, and the P1 DSP core has 48K words of SRAM and 4K words of ROM. The P0 and P1 DSP cores also have 4K words of shared memory space. The memory is divided into 16K word blocks for access. For more information on these blocks, see "ADSP-2192 Memory Map" on page 5-8.

Most microprocessors use a single address and data bus for memory access. This type of memory architecture is called Von Neumann architecture. But, DSPs require greater data throughput than Von Neumann architecture provides, so many DSPs use memory architectures that have separate buses for program and data transfer. The two buses let the DSP get a data word and an instruction simultaneously. This type of memory architecture is called Harvard architecture.

ADSP-219x family DSPs go a step farther by using a modified Harvard architecture. This architecture has program and data buses, but provides a single, unified address space for program and data storage. While the Data Memory (DM) bus only carries data, the Program Memory (PM) bus handles instructions or data, allowing dual-data accesses.

DSP core and I/O processor share accesses to internal memory. Each block of memory can be accessed by the DSP core or I/O processor in every cycle, but the DSP is held off if contending with the I/O processor core for accesses to the same block.

A memory access conflict can occur when the DSP core attempts two accesses to the same internal memory block in the same cycle. When this conflict happens, an extra cycle is incurred. The DM bus access completes first and the PM bus access completes in the following (extra) cycle.

During a single-cycle, dual-data access, the DSP core uses the independent PM and DM buses to simultaneously access data from two separate memory blocks. Though dual-data accesses provide maximum data throughput, it is important to note some limitations on how programs may use them. The limitations on single-cycle, dual-data accesses are:

- The two pieces of data must come from different memory blocks.

  If the core tries to access two words from the same memory block for a single instruction, an extra cycle is needed. For more information on how the buses access these blocks, see "Internal Data Bus Exchange" on page 5-5.

- The PM data access execution may not conflict with an instruction fetch operation.

  If the cache contains the conflicting instruction, the data access completes in a single-cycle and the sequencer uses the cached instruction. If the conflicting instruction is not in the cache, an extra cycle is needed to complete the data access and cache the conflicting instruction. For more information, see "Instruction Cache" on page 3-9.

Efficient memory usage relies on how the program and data are arranged in memory and how the program accesses the data. For more information, see "Arranging Data in Memory" on page 5-13.

As shown in Figure 5-1, the DSP has three internal buses connected to its internal memory, the Program Memory (PM) bus, Data Memory (DM) bus, and I/O Processor (IO) bus. The PM bus, DM bus, and IO bus share two memory ports; one for each block. Memory accesses from the DSP's core (computational units, data address generators, or program sequencer) use the PM or DM buses, while the I/O processor uses the IO bus for memory accesses. Using the IO bus and cycle-stealing DMA, the I/O processor can provide data transfers between internal memory and the DSP's communication ports (host PCI/USB port or AC'97 port) without hindering the DSP core's access to memory (except for stealing a cycle).

There are some bus arbitration issues involved in memory accesses. A DSP core's PM and DM buses can try to access the same block of memory in the same cycle. Also, the DSP cores' DM buses can try to access shared memory space in the same cycle. The ADSP-2192 has an arbitration system to handle this conflicting access. Arbitration for accesses to a DSP core's internal memory is fixed at the following priority: (highest priority) I/O processor accesses over the DMA bus, core accesses over the DM bus, and core accesses over the PM bus (lowest priority). Also, I/O processor accesses may not be sequential, so the DSP core's buses are never held off for more than one cycle. Arbitration for accesses to shared memory is fixed with the highest priority for DSP P0 and the lowest priority for DSP P1.

Internal Address and Data Buses

Each DAG is associated with a particular data bus and memory page. From settings at reset, DAG1 supplies addresses over the DM bus for memory page 0 and DAG2 supplies addresses over the PM bus for memory page 0. These selections can be changed using the DMPGx registers. For more information on address generation, see "Program Sequencer" on page 3-1 or "Data Address Generators" on page 4-1.

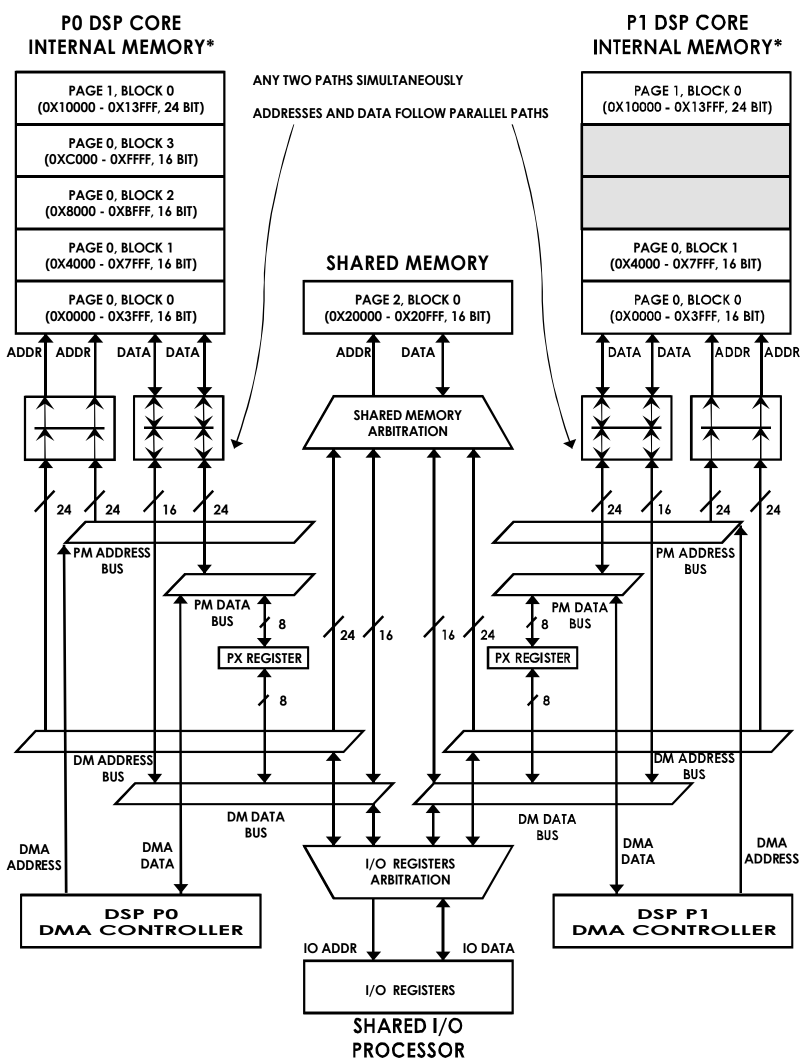


Figure 5-1. ADSP-2192 Memory and Internal Buses Block Diagram

Because the DSP's blocks of internal memory have different widths, placing 16-bit data in a Program Memory block leaves some space unused. For more information on how the DSP works with memory words, see "P0 DSP Core Internal Memory Space" on page 5-10.

The PM data bus is 24 bits wide, and the DM data bus is 16 bits wide. Both data buses can handle data words (16-bit), but only the PM data bus carries instruction words (24-bit).

Internal Data Bus Exchange

The data buses let programs transfer the contents of one register to another or to any internal memory location in a single cycle. As shown in Figure 5-1, the PM Bus Exchange (PX) register permits data to flow between the PM and DM data buses. The PX register holds the lower 8 bits during transfers between the PM and DM buses. The alignment of PX register to the buses appears in Figure 5-2.

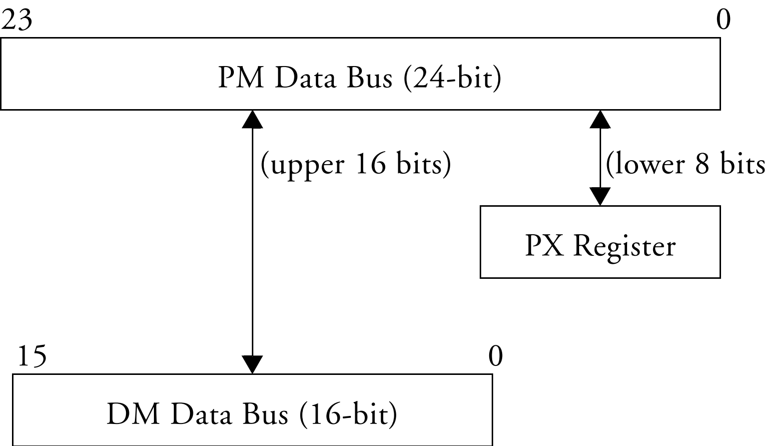


Figure 5-2. PM Bus Exchange (PX) Registers

The PX register is a Register Group 3 (REG3) registers and is accessible for register-to-register transfers.

When reading data from program memory and data memory simultaneously, there is a dedicated path from the upper 16 bits of the PM data bus to the Y registers of the computational units. This read-only path does not use the bus exchange circuit.

For transferring data from the PM data bus, the PX register is:

1. Loaded automatically whenever data (not an instruction) is read from program memory to any register. For example:

   ```
   AX0 = PM(I4,M4);
   ```

   In this example, the upper 16 bits of a 24-bit program memory word are loaded into AX0 and the lower 8 bits are automatically loaded into PX.

2. Read out automatically as the lower 8 bits when data is written to program memory. For example:

   ```
   PM(I4,M4) = AX0;
   ```

   In this example, the 16 bits of AX0 are stored into the upper 16 bits of a 24-bit program memory word. The 8 bits of PX are automatically stored to the 8 lower bits of the memory word.

For transferring data from the DM data bus, the PX register may be:

1. Loaded with a data move instruction, explicitly specifying the PX register as the destination. The lower 8 bits of the data value are used and the upper 8 are discarded.

   ```
   PX = AX0;
   ```

2. Read with a data move instruction, explicitly specifying the PX register as a source. The upper 8 bits of the value read from the register are all zeroes.

   ```
   AX0 = PX;
   ```

Whenever any register is written out to program memory, the source register supplies the upper 16 bits. The contents of the PX register are added as the lower 8 bits for instructions (such as the Type-1 and Type-32) that use the PX register, but the PX register is not used for other instructions (such as the Type-4, Type-12, and Type-29). If these lower 8 bits of data to be transferred to program memory (through the PM data bus) are important, and any instructions will be using the PX register, you should load the PX register from DM data bus before the program memory write operation.

ADSP-2192 Memory Map

The ADSP-2192's memory map appears in Figure 5-3. This memory has multiple memory spaces: internal memory space, shared memory space, system control registers, and shared DSP I/O mapped registers.

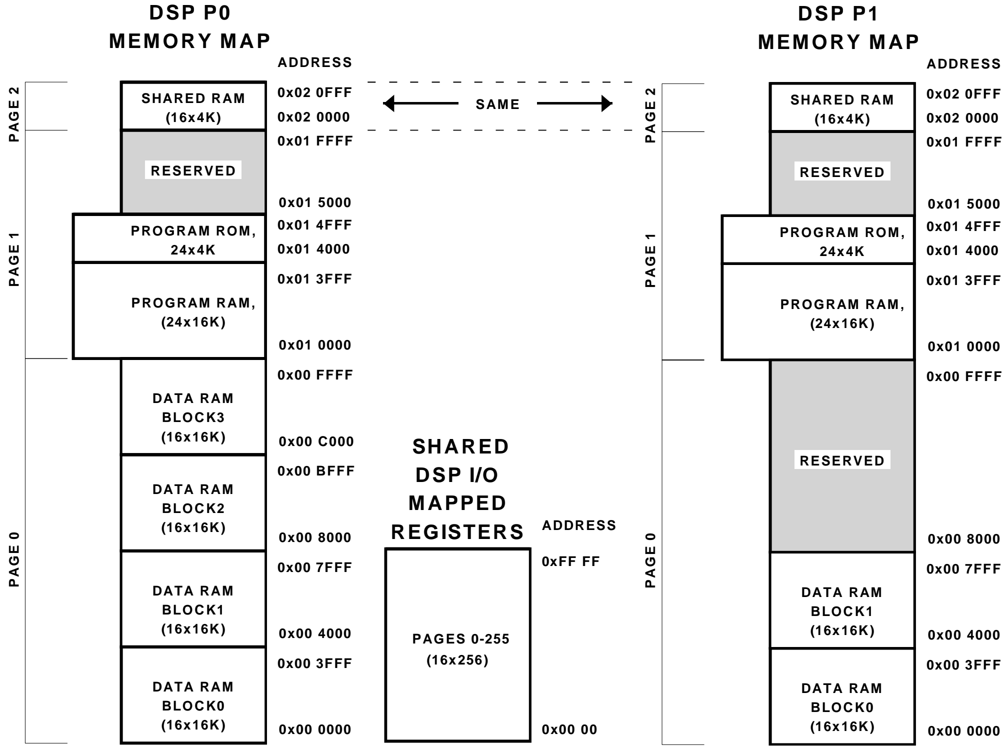


Figure 5-3. ADSP-2192 Memory Map

These memory spaces have the following definitions:

- **Internal memory space.** The internal RAM space ranges from address 0x00 0000 through 0x01 3FFF on the P0 DSP and 0x00 0000 through 0x00 7FFF plus 0x01 0000 through 0x01 3FFF on the P1 DSP. The internal (boot kernel) ROM space ranges from address 0x01 4000 through 0x01 4FFF on both DSP cores. Internal memory space refers to the DSP's on-chip SRAM and boot kernel ROM.

- **Shared memory space.** This space ranges from address 0x02 0000 through 0x02 0FFF on both DSP cores. Shared memory space refers to on-chip memory that is accessed through data move instructions and permits communications between the two cores. Accesses to shared memory space are arbitrated, with the highest priority for DSP P0 and the lowest priority for DSP P1.

- **System control registers.** This space is separate from other memory spaces and has 256 locations. (This space does not appear in Figure 5-3.) Each DSP core has its own system control register space. These locations are reserved for core-based controls and are accessed through system control register read/write instructions (`REG()`).

- **Shared I/O memory-mapped registers.** This space is separate from other memory spaces and has an address range from address 0x00 00 through 0xFF FF. The DSP cores share access to the I/O registers. The I/O registers setup and control memory-mapped peripherals. These registers are accessed through I/O port read/write instructions (`IO()`). Access to I/O memory-mapped registers is arbitrated. The priorities (from highest to lowest) are as follows: DSP P0, DSP P1, PCI interface, USB interface.

P0 DSP Core Internal Memory Space

The P0 DSP's internal memory space contains four 16K word blocks of Data Memory on Page 0 and one 16K word block of Program Memory on Page 1 on the DSP's memory map. The memory map has a unified, continuous address range.

Some features of the DSP's architecture lead to block and page distinctions within the map. These distinctions include:

- **Internal memory block width.** Blocks 0, 1, 2, and 3 reside on Page 0, are 16 bits wide, and can contain data only. The block on Page 1 (Program Memory) is 24 bits wide and can contain instructions or data.
- **Internal bus width.** The PM data bus is 24 bits wide, and the DM data bus is 16 bits wide. While either bus can access any internal memory block for data, only the PM bus can fetch instructions. The PM address bus and DM address bus are each 24 bits wide.
- **Data Address Generators.** For Type-1 instructions, DAG1 generates addresses for DM bus, and DAG2 generates addresses for the PM bus; however, for most instructions, both DAGs can access either bus. At reset, the DAGs generate addresses for Page 0; the page selections are configurable with the `DMPGx` registers.
- **Page size.** Architectural features (which are described in "Program Sequencer" on page 3-1 and "Data Address Generators" on page 4-1) lead to 64K word page segmentation of memory—a 16-bit address range per page. To move beyond a page range requires changing a value in a page register. These registers hold the upper 8 bits of the 24-bit address. There are page registers associated with I/O memory space.

To execute programs and use data in internal memory, the ADSP-2192 operates very similarly to previous ADSP-218x DSPs. For most internal memory operations, paging is not required, and the page registers remain at their reset values (Page 0).

The DSP's memory architecture permits either bus to access either internal memory block and also permits dual accesses—a single cycle operation where each bus accesses a block of memory. To arbitrate simultaneous access, the memory interface:

- Processes a memory read before memory write
- Processes a DM bus access before a PM bus access

Also on-chip, the DSP has an internal boot kernel ROM in the upper part of Page 1. Programs should treat this area as reserved and should not access this area at runtime.

P1 DSP Core Internal Memory Space

The P1 DSP's internal memory space is identical to the P0 except that it contains two 16K word blocks of Data Memory on Page 0 and contains one 16K word block of Program Memory on Page 1 on the DSP's memory map.

Shared Memory

The ADSP-2192's shared memory space contains one 4K word block of memory. Because this memory is outside of each DSP core and because access is arbitrated between the two cores, access to shared memory has core stall and latency issues. Some points on these issues include:

- Every access to shared memory incurs at least one cycle of stall (to perform synchronization), therefore minimum latency is 2 cycles.
- Arbitrated access leads to stalls for the loser of the arbitration.

- When accessing shared memory, a DSP locks out the other DSP for several cycles. A DSP can completely lock out the other DSP from shared memory by performing back to back or nearly back to back cycles to shared memory.
- Once a particular DSP "owns" the shared memory, it takes two cycles of inactivity to shared memory from that DSP to relinquish the interface.

If, for example, both DSP's are accessing the shared memory with the following code loop:

```
ar = dm(shared_memory);
nop;
nop;
/* REPEAT */
```

Each DSP gets a single shared access every 6 cycles.

The best way to get good bandwidth from shared memory is to do bursts of accesses. Each access after the first takes 2 cycles, which is the maximum throughput.

Host (PCI/USB) and DSP Internal Memory Space

PCI and USB hosts can access both DSP cores' internal memory. These accesses occur as DMA processes and are executed by the DSP core's DMA controller. To a host, a DSP core's memory appears as a memory peripheral and is accessible through a set of addresses. For more information, see Host Memory Maps in "Host (PCI/USB) Port" on page 8-1.

System Control Registers

Each DSP core has a separate memory space for system control registers. These registers support parts of the core (for example, DAGs and program sequencer) for controls. For information on using system control registers, see "ADSP-219x DSP Core Registers" on page A-1. To access system control registers, programs use the system control register read/write instructions (`REG()`).

Shared I/O Memory-mapped Registers

The DSP cores share I/O memory spaces for I/O memory-mapped registers. Similar to internal memory, the addressing for I/O memory is divided into pages. Programs select a page with the `IOPG` registers. The I/O registers control and contain status information from DSP peripherals (Host port or AC'97 port) and peripheral DMA.

Arranging Data in Memory

Each DSP core's memory is divided into 16K word blocks of program and data memory. Although the memory map is unified (either bus can access any address), programs can achieve efficient operation only by minimizing data access conflicts. The following guidelines provide an overview of how programs should interleave data in memory locations. For more information and examples, see the ADSP-219x *DSP Instruction Set Reference*:

- If two pieces of data are needed simultaneously (a dual-read), put them in different memory blocks and uses the DM bus to fetch one and the PM bus to fetch the other.

- If instruction and data fetch combinations repeatedly cause cached conflicts (repeatedly empty and fill cache), re-order the instruction to minimize these conflicts. For more information, see "Instruction Cache" on page 3-9.

Data Move Instruction Summary

Table 5-1 lists the data move instructions. For more information on assembly language syntax, see the *ADSP-219x DSP Instruction Set Reference*. In Table 5-1, note the meaning of the following symbols:

- **Dreg, Dreg1, Dreg2** indicate any register file location (Register Group)
- **Reg1, Reg2, Reg3, or Reg** indicate Register Group 1, 2, 3 or any register.
- **Ia and Mb** indicate DAG1 I and M registers.
- **Ic and Md** indicate DAG2 I and M registers.
- **Ireg and Mreg** indicate I and M registers in either DAG.
- **Imm# and Data#** indicate immediate values or data of the # of bits.

Table 5-1. Data/Register Move Instruction Summary

Instruction
Reg = Reg;
DM(<Addr16>) = |Dreg, Ireg, Mreg|;
|Dreg, Ireg, Mreg| = DM(<Addr16>);
|<Dreg>, <Reg1>, <Reg2>| = <Data16>;
Reg3 = <Data12>;
IO(<Addr10>) = Dreg;
Dreg = IO (<Addr10>);
REG(<Addr8>) = Dreg;
Dreg = REG(<Addr8>);

6 DUAL DSP CORES

Overview

The ADSP-2192 contains two ADSP-219x DSP cores. The two cores are independent, but the ADSP-2192's architecture provides a number of interactive DSP core features. These features include shared memory, shared I/O mapped registers, inter-core flags and interrupts, and mailbox registers. For information on shared memory, see "Memory" on page 5-1.

Although code execution is independent on the two cores, some ADSP-2192 settings must apply for both cores. These settings include clock and reset modes, some power down features, and general-purpose I/O settings.

Shared Dual DSP Core Settings

Settings for dual core features are contained in shared I/O mapped registers. I/O registers are accessible through the combination of a 8-bit address and an 8-bit memory page selection. To select the memory page for I/O register accesses, a program can load the IOPG register using either of the following instructions:

```
IOPG = eight_bit_immediate_value;
IOPG = register_name;
```

The I/O registers are grouped by related function onto pages to minimize the need for frequent changes of the IOPG register. To access I/O registers, programs use the following instructions:

```
IO(eight_bit_address) = Dreg; {write access}
Dreg = IO(eight_bit_address); {read access}
```

A Dreg is one of the 16 data registers within the DSP computation unit.

I/O accesses take one or more cycles within the ADSP-2192. The additional cycles often occur because the access must cross a clock boundary within the part to reach these registers. Additional cycles may also occur due to latencies for getting ownership of the internal register access bus. These additional cycles are transparent to the program.

Because each DSP core may access I/O registers simultaneously, these accesses are arbitrated. Also, the PCI interface and the USB interface can access the I/O registers using the same bus as the DSP cores. The prioritization for I/O register access among these possible masters is fixed. The priorities from highest to lowest are: DSP core P0, DSP core P1, PCI interface, and USB interface.

Unique DSP Core Settings

Settings for DSP core features (unique to one core) are contained in system control registers. To access system control registers, programs use the following instructions:

```
REG(eight_bit_address) = Dreg; {write access}
Dreg = REG(eight_bit_address); {read access}
```

A Dreg is one of the 16 data registers within the DSP computation unit.

There are two sets of system control registers on the ADSP-2192, one for each core. For a list of system registers, see "ADSP-219x DSP Core Registers" on page A-1.

Setting Dual DSP Core Features

The SYSCON, PWRCFGx, PWRPx, PLLCTL, and GPIOxxx I/O registers control the ADSP-2192's dual DSP core features.

System Control

The following bits in SYSCON control PCI bus, I/O voltage, and reset modes for the ADSP-2192. Table B-4 on page B-14 lists all the bits in SYSCON. These modes affect operations for both DSP cores:

- **PCI Reset.** SYSCON Bit 15 (PCIRST) This bit indicates the $\overline{\text{RST}}$ pin is asserted (if =0) or is not asserted (if =1).
- **Vaux Present.** SYSCON Bit 14 (VAUX) This bit indicates the VAUX supply is not powered (if =0) or is powered (if =1). (read only)
- **PCI 5V level.** SYSCON Bit 13 (PCI_5V) This bit indicates the PCI, ISA, and Card Bus interface supply (PCIVDD pins) is powered from nominal 3.3V (if =0) or is powered from nominal 5V (if =1). (read only)
- **Bus Mode.** SYSCON Bits 11–10 (BUS1-0) These bits indicate the state of the BUSMODE1-0 pins when sampled at power-on reset as: 00=PCI, 01=CardBus, 10=USB, or 11= Sub-ISA. (read only)
- **Chip Reset Source.** SYSCON Bits 9–8 (CRST1-0) These bits indicate the source of the last chip reset as: 00=power-on reset; 01=reserved; 10=PCI, ISA, CardBus, or USB bus interface hard reset; or 11=Soft Reset from the RST bit in SYSCON. A fifth possible reset source—PWRPx Soft Reset—is indicated by a PWRPx register's RD=1. Each DSP must check its PWRPx register's RD bit and clear it to zero upon reset. (read only)

- **2.5V Regulator Control Disable.** SYSCON Bit 7 (REGD) This bit enables (if =0) or disables (if =1) the on-chip 2.5V Regulator controller when the 2.5V (IVDD) supply is derived from an external regulator (for example, in USB and Mini-PCI applications).

- **Vaux Policy for AC'97 Pad Drivers.** SYSCON Bit 6 (VXPD) This bit selects where power is drawn from to drive the AC'97 and $\overline{PME}$ pads, (if=0) using BVDD when $\overline{RST}$ is deasserted and using EVAUX when $\overline{RST}$ is asserted or (if=1) using EVAUX if possible at all times. (If EVAUX is not powered, drive automatically switches to BVDD.)

- **Vaux Policy for AC'97 Pad Well Bias.** SYSCON Bit 5 (VXPW) This bit selects how the pads' N-wells are biased for the AC'97 and $\overline{PME}$ pads. If Bit 5 = 0 (if=0), the higher voltage of EVAUX is used. If Bit 5 = 1 (if=1), BVDD is used when $\overline{RST}$ is deasserted and EVAUX is used when $\overline{RST}$ is asserted. (Normally, the ADSP-2192 selects the higher voltage of BVDD and EVAUX, but for lowest EVAUX current it may be necessary to always bias the wells from BVDD.)

- **AC'97 External Devices Vaux Powered.** SYSCON Bit 4 (ACVX) This bit selects how the AC'97 interface operates during reset ($\overline{RST}$ asserted, D3cold). If Bit 4 = 0 (if=0), the interface is disabled (drive 0, disable all inputs) during reset. If Bit 4 = 1 (if=1), the interface is enabled during reset. If external AC97 devices are NOT powered during d3cold, the interface disable feature protects the ADSP-2192 from floating inputs and from outputs driving input clamps on an external device.

- **Reset Disable.** SYSCON Bit 2 (RDIS) If Bit 2 = 0 (if=0), a host bus reset of the DSPs, AC'97 codec, and the host interface is enabled. If Bit 2 = 1 (if=1), a host bus reset of the ADSP-2192 is disabled—except for the bus interface itself. (If RDIS is set, the DSP can detect that the bus is in reset by the PCIRST bit in the SYSCON register.)

  Un-masked, a bus reset affects the DSPs, the GPIOs, the AC'97, and the PCI/USB interface. Un-masked, a bus reset does not affect the Mailboxes.

The DSP memory pipeline (last 2 writes per bank) is lost upon reset. If desired, it may be flushed by three writes in a row to the same location.

- **Soft Chip Reset.** SYSCON Bit 0 (RST) When set (=1), this bit performs a soft reset of the ADSP-2192. Clearing RST (=0) has effect—always reads 0.

  Soft reset affects the DSPs and the GPIOs. Soft reset does not affect the PCI, USB, Mailboxes, AC'97, or serial EPROM.

The DSP memory pipeline (last 2 writes per bank) is lost upon reset. If desired, it may be flushed by three writes in a row to the same location.

Power Down Mode Control

The following bits control power down modes for the ADSP-2192. Table B-4 on page B-14 lists all the bits in SYSCON, Table B-5 on page B-18 lists all the bits in PWRCFGx, and Table B-6 on page B-20 lists all the bits in PWRPx.These modes affect operations for both DSP cores:

- **XTAL Force On.** SYSCON Bit 3 (XON) This bit stops (if =0) the crystal oscillator when it is not needed by an on-chip system or runs (if =1) the crystal oscillator always (even when not needed by an on-chip system. (Keeping the oscillator running permits access to the

on-chip control registers when the part is powered down. If the chip and the XTAL oscillator are powered off, attempting to write I/O registers—including this one—results in powering up the XTAL and setting the XON bit. The write succeeds, after a delay for the oscillator to stabilize. Subsequent writes or reads should not be attempted until the oscillator has stabilized, about 8K clocks or 333 µs.)

- **Power Management Event (Status/Clear).** PWRCFGx Bit 15 (PME) This bit indicates that a power management event has not (if = 0) or has (if =1) been detected for this PCI configuration (Config0, 1, or 2). Setting (=1) this bit clears PME. Clearing (=0) this bit has no effect. (This bit is an alias of the PME bit in the Power Management Control/Status Register in PCI Configuration Space for this function.)

- **Power Management Event (Set).** PWRCFGx Bit 14 (SPME) Setting (=1) this bit sets (=1) the PME bit. Clearing this bit (=0) has no effect. Always reads 0.

- **Power Management Event Enable.** PWRCFGx Bit 8 (PME_EN) This bit clears (=0) or sets (=1) the PME_EN bit in the PMCSR register in PCI Configuration space.

- **GPIO Power Management Event Enable.** PWRCFGx Bit 6 (GPME) This bit disables (if =0) or enables (if =1) setting this configuration's PME bit upon a GPIO wake up event.

- **AC'97 Power Management Event Enable.** PWRCFGx Bit 5 (APME) This bit disables (if =0) or enables (if =1) setting this configuration's PME bit upon an AC'97 interrupt/wake event.

- **PCI Function Power State.** PWRCFGx Bits 1–0 (PWRST1-0) Reports this configuration's PCI Power Management state from its PMCSR register in PCI Configuration Space. (read only)

- **DSP Interrupt Pending from GPIO Input.** `PWRPx` Bit 14 (`GINT`) This bit indicates that no interrupt is pending (if =0) or an interrupt is pending (if =1) from GPIO input. Setting this bit (=1) clears this interrupt flag. Clearing the bit (=0) has no effect. Programs should clear the interrupt source first (for example, clearing a GPIO status bit) before clearing this interrupt flag, or the flag may be retriggered. Similarly, this interrupt flag must be cleared before executing an RTI from the DSP interrupt handler routine, or the DSP may immediately take another interrupt.

- **DSP Interrupt Pending from AC'97 Input.** `PWRPx` Bit 13 (`AINT`) This bit indicates that no interrupt is pending (if =0) or an interrupt is pending (if =1) from AC'97 input. Setting this bit (=1) clears this interrupt flag. Clearing the bit (=0) has no effect. Programs should clear the interrupt source first (for example, clearing an AC'97 status bit) before clearing this interrupt flag, or the flag may be retriggered. Similarly, this interrupt flag must be cleared before executing an RTI from the DSP interrupt handler routine, or the DSP may immediately take another interrupt.

- **Power Management Interrupt Pending.** `PWRPx` Bit 12 (`PMINT`) This bit indicates that no interrupt is pending (if =0) or an interrupt is pending (if =1) from the DSP's Power Management State Change event. Setting this bit (=1) clears this interrupt flag. Clearing this bit (=0) has no effect.

- **DSP Interrupt Enable for GPIO Input.** `PWRPx` Bit 10 (`GIEN`) This bit disables (if =0) or enables (if =1) a DSP interrupt on GPIO input. When disabled (=0), GPIO input does not signal an interrupt, and the DSP does not set the corresponding Interrupt Pending bit on GPIO input. When enabled (=1), the DSP responds to an interrupt on GPIO input. (read/write)

- **DSP Interrupt Enable for AC'97 Input.** `PWRPx` Bit 9 (`AIEN`) This bit disables (if =0) or enables (if =1) a DSP interrupt on AC'97 input. When disabled (=0), AC'97 input does not signal an interrupt, and the DSP does not set the corresponding Interrupt Pending bit on AC'97 input. When enabled (=1), the DSP responds to an interrupt on AC'97 input. (read/write)
- **Power Management Interrupt Enable.** `PWRPx` Bit 8 (`PMIEN`) This bit disables (if =0) or enables (if =1) the DSP's interrupt on a Power Management State Change event.
- **DSP Wake up on GPIO Input Enable.** `PWRPx` Bit 6 (`GWE`) This bit disables (if =0) or enables (if =1) the DSP to wake from power down on GPIO input. (read/write)
- **DSP Wake up on AC'97 Input Enable.** `PWRPx` Bit 5 (`AWE`) This bit disables (if =0) or enables (if =1) the DSP to wake from power down on AC'97 input. (read/write)
- **Power Management Wake up Enable.** `PWRPx` Bit 4 (`PMWE`) This bit disables (if =0) or enables (if =1) the DSP to wake up on a Power Management State Change event. (read/write)
- **DSP Interrupt on AC'97 Frame Input Enable.** `PWRPx` Bit 3 (`FIEN`) This bit disables (if =0) or enables (if =1) an AC'97 Frame interrupt. If disabled (=0), no interrupt is signalled (read/write).

  The actual interrupt occurs once per AC'97 Frame, at the second bit of Slot 12.
- **DSP Soft Reset.** `PWRPx` Bit 2 (`RSTD`) This bit causes a soft reset to this DSP core when set (=1). The bit remains set until cleared (=0) with a write. (If the DSP core is powered down, it must be powered up first—`PWRPx` register `PU` bit set—before resetting.

- **DSP Power Up.** `PWRPx` Bit 1 (`PU`) This bit causes the DSP to power up —exit the IDLE within its power down handler—when set (=1). Programs also can use this bit to abort a power down by setting `PU` while the DSP is within its power down handler prior to the IDLE. At this point, setting `PU` causes execution to immediately continue through the IDLE without stopping clocks. On a read, `PU`=1 indicates that this DSP is in the power down interrupt handler, whether or not it has executed the power down IDLE.
- **DSP Power Down.** `PWRPx` Bit 0 (`PD`) This bit causes the DSP to power down—enter its power down handler—when set (=1). Programs also can use this bit to abort a power up by setting `PD` while the DSP is in the power down handler after executing an IDLE. At this point, setting `PD` causes the DSP to immediately re-enter the Power Down interrupt handler after it executes the RTI. On a read, `PD`=1 indicates that this DSP is powered down: either (a) it is in the power down handler and has executed an IDLE instruction), and/or (b) the DSP Clock Generator (PLL) is not running and stable. Whenever both DSPs are powered down, the DSP Clock Generator is powered down and is automatically restarted when either DSP wakes up.

DSP memory cannot be accessed via PCI when the DSP clock generator is powered down. There is a delay after powering up the DSPs with the PU bit. During this delay, memory reads must not be performed because the XTAL or the DSP PLL is not yet running and stable. After powering up by writing a 1 to the PU bit, the PD bit must be polled until it becomes 0. After it becomes 0, you know the clock generator is running and it is safe to access DSP memory again.

Clock Multiplier Mode Control

The PLLCTL register controls clock multiplier modes for the ADSP-2192. "DSP PLL Control (PLLCTL) Register" on page B-23 describes the bits in PLLCTL. These modes affect operations for both DSP cores.

The PLLCTL controls the frequencies of the PLL (Phase Locked Loop) clock generator. It should not be written unless the PLLs are powered down.

- **DSP PLL Divisor Selects.** PLLCTL Bits 7–4, 9–8, and 11–10 (DPLLM, DPLLK, DPLLN) These bits select the output frequency of the ADSP-2192's PLL as shown in Figure 6-1.

$$F_{OUT} = F_{IN} \times \frac{(DPLLM + 1) \times (DPLLK + 1)}{2 \times (DPLLN + 1)}$$

Figure 6-1. PLL F_{OUT} Formula (DSP PLL Divisor)

Where: F_{IN} = 24.576 MHz; reset values for selects are DPLLM=11, DPLLK=0, and DPLLN=0; and $F_{OUT} = 6 \times F_{IN}$ = 147.456 MHz.

To achieve the maximum possible ADSP-2192 clock rate (163.840 MHz), programs should change the DSP PLL divisor selects to: DPLLM=9, DPLLK=3, and DPLLKN=2.

- **DSP PLL Resistor Select.** PLLCTL Bit 3 (DSELR) is reserved—leave at reset value.
- **DSP PLL Capacitor Select.** PLLCTL Bit 2 (DSELC) is reserved—leave at reset value.
- **DSP PLL Boost.** PLLCTL Bit 1 (DBOOST) is reserved—leave at reset value.

- **DSP PLL Adjust.** PLLCTL Bit 0 (DADJ) This bit selects whether the PLLCTL bits for the DSP are not (if =0) or are (if =1) applied from PLLCTL bits 11-1. When DADJ is cleared (=0), the PLL uses reset values for the DSP PLL. These reset values also are returned on reading the register. When DADJ is set (=1), the PLL uses the values in bits 11–1 for the DSP PLL.

The clock speed for the DSP cores may not be changed while the DSP cores are powered. Programs should use the following procedure to change the core clock speed through the host interface:

1. Put the DSP cores into Idle, waiting to go into power management interrupt service routine (power management event interrupt is enabled).
2. Power down the DSP cores using the host interface to set (=1) the PD bit in the PWRP0 and PWRP1 registers.
3. Load the clock divisor selects into the PLLCTL register using the host interface--for the maximum core clock rate of 163.840 MHz (and defaults for the other settings), load 0x0B91 into PLLCTL.
4. Power up the cores by setting the PU bit in the PWRP0 and PWRP1 registers.

After the clocks stabilize, the DSP cores service the power management event interrupt, exiting the Idle state and resuming execution after the RTI.

GPIO and Serial EEPROM Mode Control

The following bits control general-purpose I/O and serial EEPROM modes for the ADSP-2192. See “ADSP-2192 DSP Peripheral Registers” on page B-1 for more information about the bits in these registers. These modes affect operations for both DSP cores:

- **GPIO Configuration.** (GPIOCFG) Bits 7–0 of this register select General-purpose I/O bit direction control for each I/O line as 1=input and 0=output. This bit resets to 0x7F.
- **GPIO Polarity.** (GPIOPOL) Bits 7–0 of this register select General-purpose I/O polarity as: Inputs: 0=active high, Outputs: 0=CMOS, 1=Open Drain. This bit resets to 0xFF.
- **GPIO Sticky.** (GPIOSTKY) Bits 7–0 of this register enable the General-purpose I/O “sticky-bit” feature as 1=sticky, 0=not sticky. This bit resets to 0x00.
- **GPIO Wake up Enable.** (GPIOWAKECTL) Bits 7–0 of this register enable the General-purpose I/O wake up on input feature as 1=wake-up enabled, 0=disabled (this enable requires sticky set).
- **GPIO Status.** (GPIOSTAT) Bits 7–0 of this register indicate the pin status for General-purpose I/O pins as Read=Pin state; Write: 0=clear sticky status, 1=no effect. This bit resets to 0xFF.
- **GPIO Control.** (GPIOCTL) Bits 7–0 of this register control output states for General-purpose I/O pins as Read=(Non-Inverting) Power-on state; Write = 0 or 1 set the state of output pins. This bit resets to 0x7F.
- **GPIO Pull up.** (GPIOPUP) Bits 7–0 of this register enable pull-up resistors on General-purpose I/O pins (if input) as 1=enable, 0=hi-Z. This bit resets to 0xFF.
- **GPIO Pull down.** (GPIOPDN) Bits 7–0 of this register enable pull-down resistors on General-purpose I/O pins (if input) as 1=enable, 0=hi-Z. This bit resets to 0x00.

Serial EPROM I/O Control/Status. (SPROMCTL) This register contains the following bits that configure Serial EPROM access features: SCKI, SENI, SDAI, SCK, SEN, and SDA. Table B-9 on page B-29 lists all the bits in SPROMCTL

Using Dual-DSP Interrupts and Flags

Two features on each DSP core permit DSP-to-DSP communications: the DSP-DSP interrupt and DSP-DSP flags (or semaphores). Each DSP core has its own interrupt controller and interrupt vector table as shown in Table 6-1. Each DSP core has a FLAGS system control register as shown in Table 6-2.

Table 6-1. Interrupt Vectors for an ADSP-2192 DSP Core

IRPTL/IMASK Bit	Interrupt Priority	Interrupt Description	Vector Address Offset[1]
0	1	Reset (Non-maskable)	0x00
1	2	Power down (Non-maskable)	0x04
2	3	Kernel interrupt (single step)	0x08
3	4	Stack Status	0x0C
4	5	Mailbox	0x10
5	6	Timer	0x14
6	7	Reserved	0x18
7	8	PCI Bus Master	0x1C
8	9	DSP-DSP	0x20
9	10	FIFO0 Transmit	0x24
10	11	FIFO0 Receive	0x28
11	12	FIFO1 Transmit	0x2C
12	13	FIFO1 Receive	0x30
13	14	Reserved	0x34

Table 6-1. Interrupt Vectors for an ADSP-2192 DSP Core (Continued)

IRPTL/IMASK Bit	Interrupt Priority	Interrupt Description	Vector Address Offset[1]
14	15	Reserved	0x38
15	16	AC'97 Frame	0x3C

1 The Interrupt Vector Address Values are represented as offsets from Address 0x10000. This Address corresponds to the start of Program RAM in each DSP core.

Table 6-2. ADSP-2192 DSP Core FLAGS Register

Flag Bit	Direction	Name—Description
0	Output	FLAG0_OUT—DSP-to-DSP Flag 0.
1	Output	FLAG1_OUT—DSP-to-DSP Flag 1.
2	Output	DSP_IRQ_OUT—DSP-to-DSP Interrupt.
3	Output	Reserved
4	Output	Reserved
5	Output	Reserved
6	Output	Reserved
7	Output	BUSLK_REQ—I/O register bus lock request.
8	Input	FLAG0_IN—DSP-to-DSP Flag 0.
9	Input	FLAG1_IN—DSP-to-DSP Flag 1.
10	Input	DSP_IRQ_IN—DSP-to-DSP Interrupt.
11	Input	Reserved
12	Input	AC97RG_STAT—AC'97 codec register, I/O register bus access status.

Table 6-2. ADSP-2192 DSP Core FLAGS Register (Continued)

Flag Bit	Direction	Name—Description
13	Input	DSPRG_STAT—DSP I/O register bus status (pending write from DSP).
14	Input	
15	Input	BUSLK_STAT—Register Bus Lock Status.

Figure 6-2 shows how the bits in the FLAGS register on one DSP core affect the state of the FLAGS register on the other DSP core. Because the two cores' FLAGS are interconnected this way, programs can use interrupt driven or polled techniques for DSP-to-DSP communications.

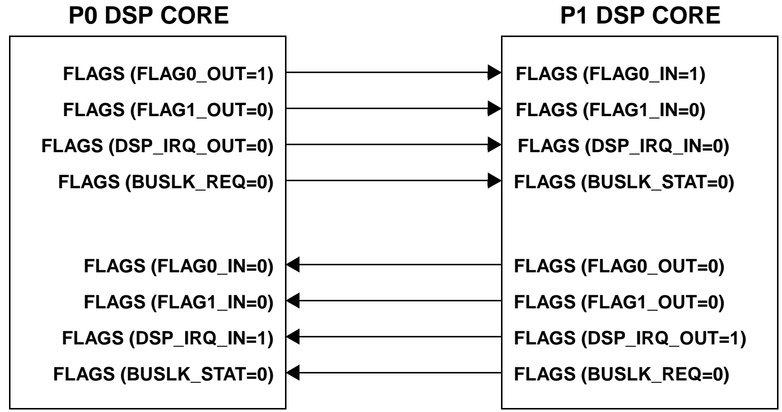


Figure 6-2. DSP-to-DSP Core Flags, Interrupts, and Bus Lock

The following example illustrates a process in which DSP core P0 asserts DSP-DSP Flag 0 (a semaphore) of DSP core P1.

```
/* assert DSP-to-DSP flag 0 */
ax0 = 0x0001;
reg(0x34) = ax0;
```

When a 1 is written to bit 0 of the core FLAGS register of DSP core P0, the related core flag of DSP core P1, bit 8, is set. The DSP-to-DSP Flag1 works the same way.

The DSP-DSP interrupt lets DSP core P0 interrupt DSP core P1 if the DSP-DSP interrupt is unmasked in the IMASK register of DSP core P1. The DSP-DSP interrupts can either be nested with higher priority interrupts taking precedence or they can be processed sequentially.

The following example illustrates how code could initialize the DSP-DSP interrupt.

```
/* Initialize DSP-to-DSP Interrupt */
AY0=IMASK;
AY1=0x0100;
AR = AY0 or AY1;
/* Unmask DSP-DSP Interrupts */
IMASK=AR;
/* Enable global interrupts */
ENA INT;
```

The content of IMASK is OR'd with 0x0100 and written back to IMASK. This unmasks DSP-DSP interrupt for that particular DSP core. The last instruction globally enables interrupt servicing. If the above DSP-DSP interrupt initialization executes on DSP core P1, DSP core P0 could initiate the DSP-DSP interrupt by asserting bit 2 of its FLAGS register.

Controlling I/O Register Bus Accesses

Looking at the ADSP-2192 as a multiprocessing system (dual DSP cores), the ADSP-2192 must provide some means for the individual DSP cores to perform uninterrupted sequences of register bus accesses. Although, this may be an infrequently used feature, it is critical for certain cases. To support uninterrupted accesses for a DSP core, the ADSP-2192 provides a bus lock feature for retaining mastership of the I/O memory-mapped registers bus.

As shown in Figure 6-2, to lock the I/O registers bus, a DSP core sets (=1) the BUSLK_REQ bit in its FLAGS register. Setting this FLAGS output bit generates a continuous request on the I/O registers bus. After mastership of this bus is granted to that DSP core, it remains granted until the bit is cleared. A DSP core can check whether the bus has been granted by examining the BUSLK_STAT it its FLAGS register. When set (=1), the BUSLK_STAT bit indicates the DSP core has been granted the bus.

While the I/O registers bus is locked for a DSP core, that DSP core can perform Read-Modify-Write operations without the danger of the other DSP core or the PCI/USB interface changing the register.

Programs should be careful to avoid locking the I/O register bus for extended periods of time.

The I/O register bus is a shared resource that affects almost all aspects of operation of an ADSP-2192. On this bus, either a DSP core or the external bus interface (whether PCI, USB or sub-ISA) initiates transactions. Transactions may be targeted at I/O registers for PCI, USB, system control, AC'97 codec, GPIO, and serial EEPROM functions. Only one transaction may be in progress at a time, and all other initiators must wait for the current transaction to complete.

While waiting for I/O register bus access, DSP execution halts at the I/O register access instruction, and the DSP is *unable to complete any other task*.

A DSP core in this halt state ignores interrupts, does not process DMA, and does not execute other instruction. For example,

- DSP P1 is in the middle of an I/O register bus transaction, and every initiator wants to launch a I/O register bus transaction.
- DSP P1 can not start a new transaction until the current I/O register access instruction completes.
- DSP P0 will halt, waiting for the bus, until its transaction is granted and completed. Alternately, DSP P0 may attempt to lock the bus and then check for bus lock status. If the bus was not locked by DSP P0, it knows a transaction is in progress and may choose to remove the lock request and try again later.
- The PCI bus cannot initiate a transaction and receives a Retry semantic.
- The USB bus cannot initiate a transaction and receives a Retry semantic.
- The sub-ISA bus cannot initiate a transaction and does not receive an ISA acknowledge until the transaction is granted and completed.

A zero wait state I/O register transaction consumes approximately 12 DSP cycles. Most transactions are zero wait state, but some—such as transactions that must be mapped through external interfaces—may be much longer. For example, an access to AC97 codec registers must go through the AC'97 interface.

Writes to AC'97 codec registers are posted, but only one may complete per AC'97 frame. Up to two writes may be pending at any one time. The first write will complete with zero I/O register bus wait states. A second write launched immediately after the first incurs I/O register bus wait states equivalent to a few AC'97 BITCLKs. A third write in a row blocks for an entire AC'97 frame.

Programs should make use of the Frame interrupt to time AC'97 codec writes out to one per frame, assuring that the writes all complete with zero wait states.

Reads from AC'97 codec registers must always wait for the data to be returned. A read must also wait for any pending AC'97 codec register writes to complete before it can begin. In the best case, a read takes one full AC'97 frame plus another three AC'97 slots (25.39 µs, or approximately 3,744 DSP cycles). This should also be the typical case if the AC'97 Frame Interrupt is used to time the read.

The worst case AC'97 read time is 4 frames plus 3 slots (87.89 µs, or approximately 12,960 DSP cycles). This occurs only if there were already two AC'97 codec register writes pending just after the start of a frame.

Most AC'97 codec registers may be shadowed and actual reads should be rare.

In the worst case example,

- DSP P1 posts two AC'97 codec register writes just after the start of a new Frame.
- DSP P0 immediately follows with a read to an AC'97 codec register. DSP P0 is unable to compute, DMA, or interrupt for 87.89 µs.
- DSP P1 can compute with data in its own memory, but cannot communicate with DSP P0 nor access any I/O register for 87.89 µs.
- The external bus interface can communicate with DSP P1, but cannot communicate with DSP P0 nor access any I/O register for 87.89 µs.

In this state, the entire ADSP-2192 system is highly constrained.

Using DSP and PCI Mailbox Registers

The ADSP-2192 contains mailbox registers for passing data between the host and DSP cores. These registers let the DSP cores and host pass data back and forth with minimal intervention. Optionally, writes to a mailbox can cause an interrupt for the mailbox's owner. For example a DSP write to an outgoing mailbox can cause a PCI interrupt, or a PCI write to an incoming mailbox can cause a DSP interrupt. Because mailboxes are read/writable by both DSP cores and the host, the DSPs also can use the mailboxes to pass data to each other—there is not interrupt support for this technique. The ADSP-2192 contains the following mailbox registers:

- **Mailbox Status.** (MBXSTAT) This register indicates the pending status for mailbox I/O and interrupts.
- **Mailbox Interrupt Control.** (MBXCTL) This register disables or enables interrupts for DSP and PCI mailbox I/O.
- **Incoming Mailboxes (PCI/USB to DSP mailboxes).** (MBX_IN0, MBX_IN1) These registers contain mailbox I/O from the host to the DSP. These registers are read-write for the host and both DSP cores.
- **Outgoing Mailboxes (DSP to PCI/USB mailboxes).** (MBX_OUT0, MBX_OUT1) These registers contain mailbox I/O from the DSPs to the host. These registers are read-write for the DSP cores and host.

Mailbox Status (MBXSTAT) Register

Note: *All bits in this register are reset to zero.*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OUT1 Valid	OUT0 Valid	IN1 Valid	IN0 Valid	DSP2 OUT1 PEND	DSP2 OUT0 PEND	DSP2 IN1 PEND	DSP2 IN0 PEND	DSP1 OUT1 PEND	DSP1 OUT0 PEND	DSP1 IN1 PEND	DSP1 IN0 PEND	PCI OUT1 PEND	PCI OUT0 PEND	PCI IN1 PEND	PCI IN0 PEND

Table 6-3. MBXSTAT Register Bit Descriptions

Bit Position	Bit Name	Description
0	PCI IN0 PEND	`InBox0` PCI Interrupt Pending. This bit is set when the DSP reads valid data from `InBox0`, if enabled by the corresponding Mailbox Control Register bit.
1	PCI IN1 PEND	`InBox1` PCI Interrupt Pending. This bit is set when the DSP reads valid data from `InBox1`, if enabled by the corresponding Mailbox Control Register bit.
2	PCI OUT0 PEND	`OutBox0` PCI Interrupt Pending. This bit is set when the DSP writes valid data to `OutBox0`, if enabled by the corresponding Mailbox Control Register bit.
3	PCI OUT1 PEND	`OutBox1` PCI Interrupt Pending. This bit is set when the DSP writes valid data to `OutBox1`, if enabled by the corresponding Mailbox Control Register bit.
4	DSP1 IN0 PEND	`InBox0` DSP 1 Interrupt Pending. This bit is set when the PCI writes valid data to `InBox0`, if enabled by the corresponding Mailbox Control Register bit.

Table 6-3. MBXSTAT Register Bit Descriptions (Continued)

Bit Position	Bit Name	Description
5	DSP1 IN1 PEND	InBox1 DSP 1 Interrupt Pending. This bit is set when the PCI writes valid data to InBox1, if enabled by the corresponding Mailbox Control Register bit.
6	DSP1 OUT0 PEND	OutBox0 DSP 1 Interrupt Pending. This bit is set when the PCI acknowledges reading data from OutBox0 by writing a 1 to bit 14, if enabled by the corresponding Mailbox Control Register bit.
7	DSP1 OUT1 PEND	OutBox1 DSP 1 Interrupt Pending. This bit is set when the PCI acknowledges reading data from OutBox1 by writing a 1 to bit 15, if enabled by the corresponding Mailbox Control Reg bit.
8	DSP2 IN0 PEND	InBox0 DSP 2 Interrupt Pending. This bit is set when the PCI writes valid data to InBox0, if enabled by the corresponding Mailbox Control Register bit.
9	DSP2 IN1 PEND	InBox1 DSP 2 Interrupt Pending. This bit is set when the PCI writes valid data to InBox1, if enabled by the corresponding Mailbox Control Register bit.
10	DSP2 OUT0 PEND	OutBox0 DSP 2 Interrupt Pending. This bit is set when the PCI acknowledges reading data from OutBox0 by writing a 1 to bit 14, if enabled by the corresponding Mailbox Control Register bit.
11	DSP2 OUT1 PEND	OutBox1 DSP 2 Interrupt Pending. This bit is set when the PCI acknowledges reading data from OutBox1 by writing a 1 to bit 15, if enabled by the corresponding Mailbox Control Register bit.
12	IN0 Valid	InBox0 Data Valid. A one means valid data has been written into the InBox0 register. The bit is cleared when it is written with ones, or when InBox0 is read.

Table 6-3. MBXSTAT Register Bit Descriptions (Continued)

Bit Position	Bit Name	Description
13	IN1 Valid	InBox1 Data Valid. A one means valid data has been written into the InBox1 register. The bit is cleared when it is written with ones, or when InBox1 is read.
14	OUT0 Valid	OutBox0 Data Valid. A one means valid data has been written into the OutBox0 register. The bit is cleared when it is written with ones, or when OutBox0 is read by the DSP. Reads by the PCI have no side effects—the PCI must clear the valid status explicitly by writing this bit with a 1 after reading OutBox0.
15	OUT1 Valid	OutBox1 Data Valid. A one means valid data has been written into the OutBox1 register. The bit is cleared when it is written with ones, or when OutBox1 is read. Reads by the PCI have no side effects: the PCI must clear the valid status explicitly by writing this bit with a 1 after reading OutBox1.

Mailbox Interrupt Control (MBXCTL) Register

Note: *All bits in this register are reset to zero.*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				DSP2 OUT1 ENA	DSP2 OUT0 ENA	DSP2 IN1 ENA	DSP2 IN0 ENA	DSP1 OUT1 ENA	DSP1 OUT0 ENA	DSP1 IN1 ENA	DSP1 IN0 ENA	PCI OUT1 ENA	PCI OUT0 ENA	PCI IN1 ENA	PCI IN0 ENA

Table 6-4. MBXCTL Register Bit Descriptions

Bit Position	Bit Name	Description
0	PCI IN0 ENA	`InBox0` PCI Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
1	PCI IN1 ENA	`InBox1` PCI Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
2	PCI OUT0 ENA	`OutBox0` PCI Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
3	PCI OUT1 ENA	`OutBox1` PCI Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
4	DSP1 IN0 ENA	`InBox0` DSP core P0 Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.

Table 6-4. MBXCTL Register Bit Descriptions (Continued)

Bit Position	Bit Name	Description
5	DSP1 IN1 ENA	`InBox1` DSP core P0 Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
6	DSP1 OUT0 ENA	`OutBox0` DSP core P0 Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
7	DSP1 OUT1 ENA	`OutBox1` DSP core P0 Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
8	DSP2 IN0 ENA	`InBox0` DSP core P1 Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
9	DSP2 IN1 ENA	`InBox1` DSP core P1 Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
10	DSP2 OUT0 ENA	`OutBox0` DSP core P1 Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
11	DSP2 OUT1 ENA	`OutBox1` DSP core P1 Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
15:12	Reserved	

InBox 0 - PCI/USB to DSP Mailbox 0 (MBX_IN0) Register

INBOX0, INBOX1, OUTBOX0, OUTBOX1 do not have individual bits. The whole register is a 16-bit word.

InBox 1 - PCI/USB to DSP Mailbox 1 (MBX_IN1) Register

INBOX0, INBOX1, OUTBOX0, OUTBOX1 do not have individual bits. The entire register is a 16-bit word.

OutBox 0 - DSP to PCI/USB Mailbox 0 (MBX_OUT0) Register

INBOX0, INBOX1, OUTBOX0, OUTBOX1 do not have individual bits. The entire register is a 16-bit word.

Note: *All bits in this register are reset to zero.*

OutBox 1 - DSP to PCI/USB Mailbox 1 (MBX_OUT1) Register

INBOX0, INBOX1, OUTBOX0, OUTBOX1 do not have individual bits. The entire register is a 16-bit word.

7 I/O PROCESSOR

Overview

The DSP's I/O processor manages Direct Memory Accessing (DMA) of DSP memory through the host (PCI) port and AC'97 codec port. Each DMA operation transfers an entire block of data. By managing DMA, the I/O processor lets programs move data as a background task while using the processor core for other DSP operations. The I/O processor's architecture, which appears in Figure 7-1 on page 7-3, supports a number of DMA operations. These operations include the following transfer types:

- Internal memory ↔ host (PCI)
- Internal memory ↔ AC'97 codec port I/O

This chapter describes the I/O processor and how the I/O processor controls host port and AC'97 port operations. For information on connecting external devices to the Host port or AC'97 ports, see "Host (PCI/USB) Port" on page 8-1 or "AC'97 Codec Port" on page 9-1.

DMA transfers between internal memory and a host use the DSP's host port. For these types of transfers, a DSP program sets up the DSP core's DMA controller with the internal memory DMA address, DMA next (process) address, DMA count, and DMA current count. These DMA set up parameters are the Transfer Control Block (TCB) for the DMA transfer.

A host program needs to set up the PCI interfaces' DMA controller with similar parameters for the host system to receive or transmit the DMA. After setup, the DMA transfers begin when the DSP or host program enables the channel and continue until the I/O processor transfers the entire buffer to or from DSP memory.

Similarly, DMA transfers between internal memory and the AC'97 port have DMA parameters (a TCB). When the I/O processor performs DMA between internal memory and one of these ports, the DSP program sets up the parameters and the I/O goes through the port.

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.

To further minimize loading on the processor core, the I/O processor supports chained DMA operations through the DMA next (process) address feature. When using chained DMA, a program can set up a DMA transfer to automatically start the next DMA transfer after the current one completes.

Figure 7-1 on page 7-3 shows the DSP's I/O processor, related ports, and buses. Figure 7-2 on page 7-4 shows more detail on DMA channel data paths.

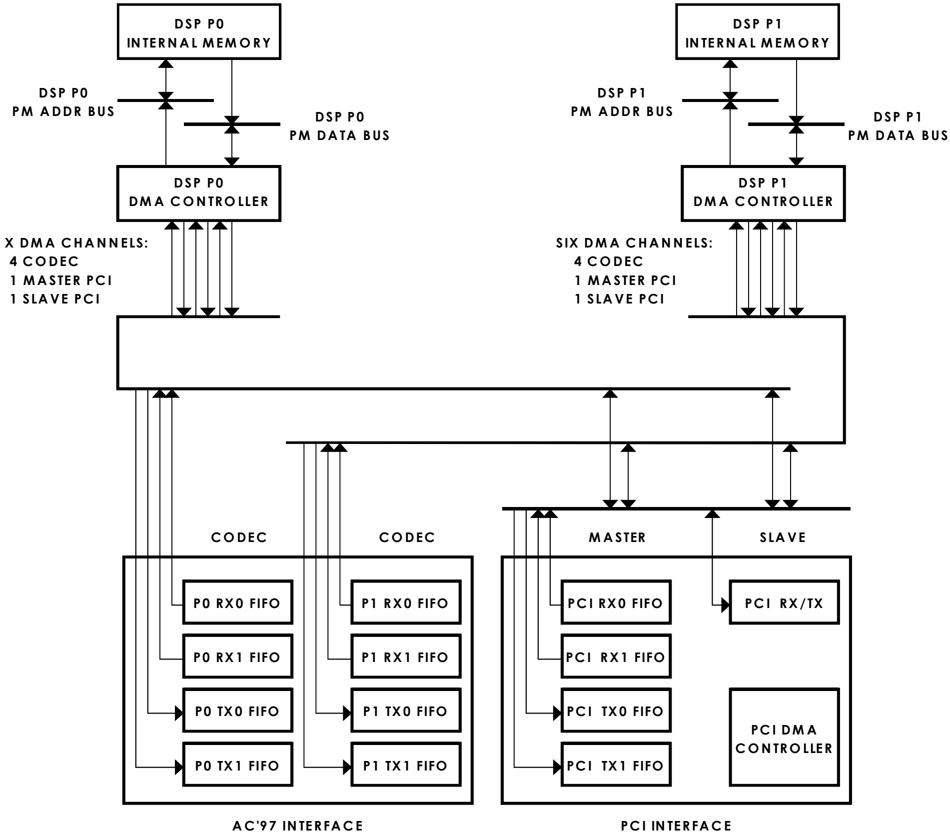


Figure 7-1. ADSP-2192 DMA Channels, Requests and Data Paths

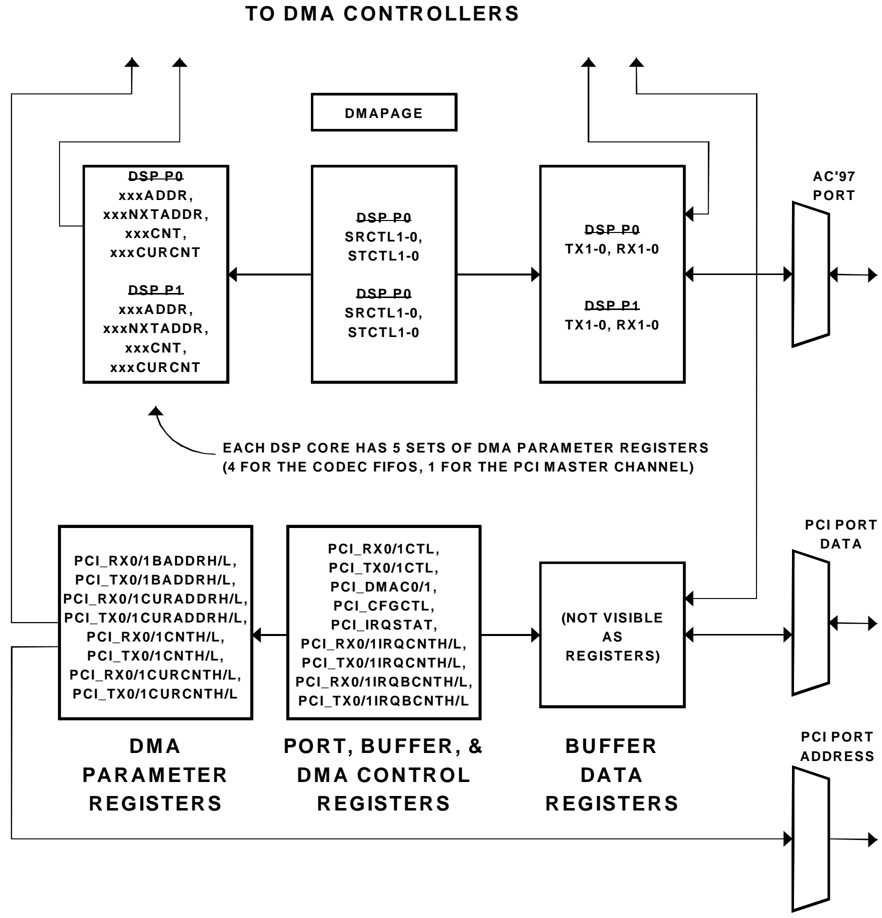


Figure 7-2. ADSP-2192 DMA Control, Status and Buffer Registers

Each DSP core has four Codec DMA FIFOs (RX0/1, TX0/1). These FIFOs can be connected to the AC'97 codec. These FIFOs are eight levels deep and are located in system control register space (in the cores). These FIFOs' control registers are located in system control register space and configure the codec connection and DMA enable (STCTL0/1, SRCTL0/1).

The codec DMA FIFOs' DMA parameter registers are located in system control register space and configure the DMA address (xxxADDR in DSP memory), DMA next address (xxxNXTADDR in DSP memory), DMA count (xxxCNT), and DMA current count (xxxCURCNT). These registers permit circular buffering (through the next address feature).

The PCI interface in the host port has four DMA FIFOs (two receive, two transmit). These FIFOs are connected to the PCI master DMA channel. These FIFO's are four levels deep and are not visible as registers. These FIFO's control registers are located in shared I/O register space and configure the DMA mode as plain or scatter-gather (PCI_Rx0/1CTL, PCI_Tx0/1CTL) and control the PCI FIFOs and enable DMA (PCI_DMAC0/1).

Because the PCI interface has more FIFO features than the core FIFOs, the PCI FIFOs also have registers for control and status of PCI interrupts (PCI_CFGCTL, PCI_IRQSTAT) and have registers for the DMA interrupt count (PCI_Rx0/1IRQCNTH/L, PCI_Tx0/1IRQCNTH/L, PCI_Rx0/1IRQBCNTH/L, PCI_Tx0/1IRQBCNTH/L).

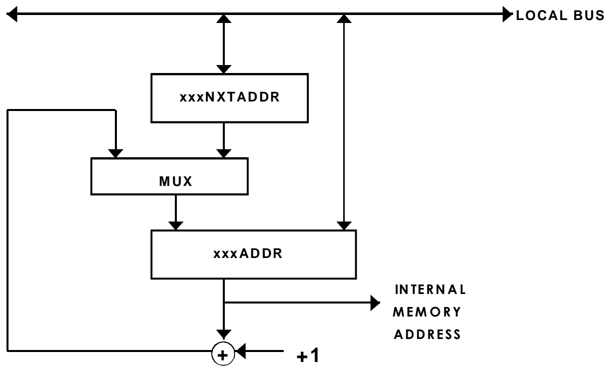
The PCI FIFO's DMA (host-side) parameter registers are located in shared I/O register space and configure the DMA base address (PCI_Rx0/1BADDRH/L, PCI_Tx0/1BADDRH/L), DMA current address (PCI_Rx0/1CURADDRH/L, PCI_Tx0/1CURADDRH/L), DMA count (PCI_Rx0/1CNTH/L, PCI_Tx0/1CNTH/L), and DMA current count (PCI_Rx0/1CURCNTH/L, PCI_Tx0/1CURCNTH/L). The address and count information in the PCI FIFO's DMA parameter registers refers to addresses on the PCI host.

Each DSP core has parameter registers for PCI master channel DMA (DSP-side) located in the core's system control register space. These parameter registers configure the DMA address (MASTADDR in DSP memory), DMA next address (MASTNXTADDR in DSP memory), DMA count (MASTCNT), and DMA current count (MASTCURCNT). These registers permit circular buffering (through the next address feature). The address and count information in the PCI master channel DMA parameter registers refers to addresses in the DSP core's internal memory. These master channel DMAs are controlled by the PCI FIFO's control registers.

Although PCI slave transfers use a DMA channel, there are no DMA parameters associated with these slave transfers.

Figure 7-3 on page 7-7 shows block diagrams of the I/O processor's address generator (DMA controller); Figure 7-4 on page 7-8 shows those block diagrams for Host/PCI. Table 7-1 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 that the I/O processor uses to address a data buffer in internal memory. The xxxADDR 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 on the DSP's DM address bus and applies the address to internal memory during each DMA cycle—a clock cycle in which a DMA transfer is taking place.



DMA WORD COUNTER (INTERNAL ADDRESSES)

LOCAL BUS

xxxCNT

MUX

xxxCURCNT

+

– 1

Figure 7-3. DMA Address Generator (Internal Addresses)

DMA ADDRESS GENERATOR (HOST/PCI ADDRESSES)

LOCAL BUS

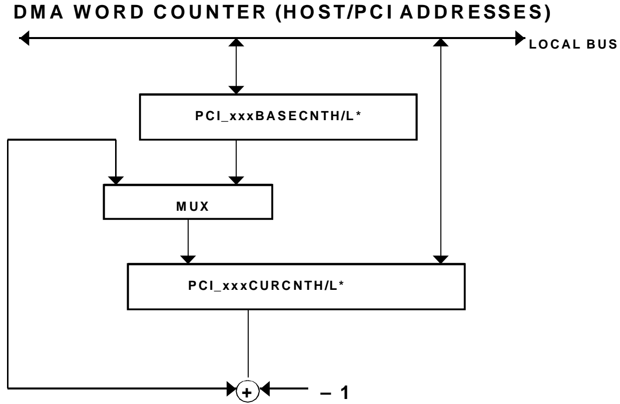
PCI_xxxBASEADDRH/L*

MUX

PCI_xxxCURADDRH/L*

INTERNAL
MEMORY
ADDRESS

+1



* 32-bit register (H/L = high/low) doing a 32-bit increment/decrement

Figure 7-4. DMA Address Generator (PCI)

After transferring each data word to or from internal memory, the I/O processor adds the modify value to the address register to generate the address for the next DMA transfer and writes the modified address value to the address register. The modify value is +1.

If the I/O processor modifies the address register past the maximum value for a memory page, the 16-bit address wraps around to zero, and DMA continues from the start address of that Page in memory.

Each DMA channel has a count register (`xxxCNT`) that programs load with a word count to be transferred. At the start of the DMA, the I/O processor loads the `xxxCURCNT` register from the `xxxCNT` register. The I/O decrements the count register after each DMA transfer on that channel. When the count reaches zero, the I/O processor generates the interrupt for that DMA channel.

If a program loads the count (`xxxCNT`) register with zero, the I/O processor does not disable DMA transfers on that channel. The I/O processor interprets the zero as a request for 2^{16} transfers. This count occurs because the I/O processor starts the first transfer before testing the count value. The only way to disable a DMA channel is to clear its DMA enable bit.

Each DMA channel also has a chain pointer register (`xxxNXTADDR`). Chained DMA sequences are a set of multiple DMA sequences, the next starting when the previous one is complete. For more information, see "Chaining DMA Processes" on page 7-22.

The host port DMA channels each contain additional parameter registers that set up the host side DMA. The I/O processor generates 32-bit PCI host memory addresses during DMA transfers between internal memory and a PCI host.

When a particular I/O port needs to perform transfers to or from internal memory, the channel asserts a request. The I/O processor prioritizes this request with all other valid DMA requests. Table 7-1 lists the DMA channels in priority order. For more information, see "Managing DMA Channel Priority" on page 7-21.

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.

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.

Each DSP core's six DMA channels are numbered as shown in Table 7-1. This table also shows the control, parameter, and data buffer registers that correspond to each channel.

Table 7-1. DMA Channel Registers For Each DSP Core: Controls, Parameters, and Buffers

DMA Chan#	Control Registers	Parameter Registers	Buffer Register	Description
0	SRCTL0	Rx0ADDR, Rx0NXTADDR, Rx0CNT, Rx0CURCNT	RX0	AC'97 Port Receive
1	SRCTL1	Rx1ADDR, Rx1NXTADDR, Rx1CNT, Rx1CURCNT	RX1	AC'97 Port Receive
2	STCTL0	Tx0ADDR, Tx0NXTADDR, Tx0CNT, Tx0CURCNT	TX0	AC'97 Port Transmit
3	STCTL1	Tx1ADDR, Tx1NXTADDR, Tx1CNT, Tx1CURCNT	TX1	AC'97 Port Transmit

Table 7-1. DMA Channel Registers For Each DSP Core: Controls, Parameters, and Buffers (Continued)

DMA Chan#	Control Registers	Parameter Registers	Buffer Register	Description
4	PCI_DMAC0/1	MSTRADDR, MSTRNX-TADDR, MSTRCNT, MSTRCURCNT	not visible	Host Port FIFO Buffer
5	PCI Slave Channel, no parameters or controls			

The codec channel DMA control and parameter registers are system control registers in each DSP core, and the host channel DMA control and parameter registers are I/O memory-mapped registers. For more information on these registers, see "ADSP-2192 DSP Peripheral Registers" on page B-1.

To set up DMA on the "DSP-side", the DSP program must load the control and parameter registers. Because the I/O processor registers are memory-mapped, the DSP and host have access to program the host side of DMA operations. A processor sets up a DMA channel by writing the transfer's parameters to the DMA parameter registers. After these registers are loaded, the DSP (or host) is ready to start the DMA.

The host port and AC'97 port each have a DMA enable bit (DEN or SDEN) 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 (count parameter).

To start a new (non-chained) DMA sequence after the current one is finished, programs must disable the channel (clear its DEN bit); write new parameters to the registers; then enable the channel (set its DEN bit). For looped or chained DMA operations, this disable-enable process is not necessary. For more information, see "Chaining DMA Processes" on page 7-22.

Setting I/O Processor—Host Port Modes

The PCI_Rx0-1CTL, PCI_Tx0-1CTL, and PCI_DMAC0-1 registers control the host port operating mode for the I/O processor. See "ADSP-2192 DSP Peripheral Registers" on page B-1 for more information about the bits in these registers.

The following bits control host port I/O processor modes. Except for the FLSH bit, the control bits in the PCI_DMACx registers have a one cycle effect latency (take effect on the second cycle after change). The FLSH bit has a two cycle effect latency. Programs should not modify an active DMA channel's PCI_Rx0-1CTL, PCI_Tx0-1CTL, or PCI_DMAC0-1 register; other than to disable the channel by clearing the DEN bit.

- **Scatter-gather DMA Enable.** PCI_Rx0-1CTL and PCI_Tx0-1CTL Bit 0 (SGDEN) This bit disables (if =0) or enables (if =1) scatter-gather DMA mode.

- **Loop Enable.** PCI_Rx0-1CTL and PCI_Tx0-1CTL Bit 1 (LPEN) This bit disables (if =0) or enables (if =1) DMA looping mode.

- **Interrupt Mode.** PCI_Rx0-1CTL and PCI_Tx0-1CTL Bit 3–2 (INTMODE) These bits select the DMA interrupt mode as: 00 = interrupt disabled, 01 = interrupt on count, 10 = interrupt on SGD flag, or 11 = interrupt on EOL.

- **Current Scatter-gather DMA Valid.** PCI_Rx0-1CTL and PCI_Tx0-1CTL Bit 5–4 (SGVL) These bits indicate the state of the current scatter-gather DMA as: 00 = full SGD descriptor needed (software must initialize this value), 01 = partial SGD descriptor fetched, 10 = SGD valid, or 11 = reserved (invalid status).
- **Flag Bit Set in Current Scatter-gather DMA.** PCI_Rx0-1CTL and PCI_Tx0-1CTL Bit 6 (FLG) This bit indicates a flag is not (if =0) or is (if =1) set in the current scatter-gather DMA.
- **EOL Bit Set in Current Scatter-gather DMA.** PCI_Rx0-1CTL and PCI_Tx0-1CTL Bit 7 (EOL) This bit indicates an EOL is not (if =0) or is (if =1) set in the current scatter-gather DMA.
- **DMA Enable.** PCI_DMAC0-1 Bit 0 (DEN) This bit enables (if set, =1) or disables (if cleared, =0) DMA for the corresponding host port FIFO buffer. The I/O processor will automatically clear this bit when the DMA transfer is complete on the host interface.
- **DMA Direction.** PCI_DMAC0-1 Bit 1 (TRAN) This bit selects the transfer direction (transmit if set, =1) (receive if cleared, =0) for the host port DMA.
- **Flush FIFO.** PCI_DMAC0-1 Bit 2 (FLSH) This bit flushes the corresponding FIFO when set (=1).
- **DSP P0/P1 Select.** PCI_DMAC0-1 Bit 3 (DSP) This bit selects the DSP core for the DMA process as: 0 = P0 or 1 = P1.
- **DMA Packing Disable (Double Word Mode).** PCI_DMAC0-1 Bit 4 (DPD) This bit enables (if =0) or disables (if =1) PCI interface word packing.
- **Configuration Select 2, 1, or 0.** PCI_DMAC0-1 Bits 7, 6, 5 (CFGx) These bits select a PCI device configuration for the DMA; only one of the three configurations may be active (=1) at a time.

- **DMA FIFO Empty Status.** PCI_DMAC0-1 Bit 8 (EMPTY) This bit indicates the FIFO status as: 0 = not empty or 1 = empty. A one written to this bit clears it. This bit is also cleared by writing the DEN bit to initiate a DMA transaction.
- **DMA Channel Halt Status.** PCI_DMAC0-1 Bit 9 (HALT) This bit is set to one when the master DMA channel is disabled by the PCI address generation logic. This occurs if the host interface receives an error signal for an attempted DMA transfer. A one written to this bit clears it. This bit is also cleared by writing the DEN bit to initiate a DMA transaction.
- **DMA Channel Loop Status.** PCI_DMAC0-1 Bit 10 (LOOP) This bit indicates DMA loop status as: 0 = no looping occurred or 1 = looping occurred. A one written to this bit clears it. This bit is also cleared by writing the DEN bit to initiate a DMA transaction.

Host Port Buffer Modes

The DPD bit in the PCI_DMAC0-1 registers select a buffer's packing mode. Packing is enabled when this bit is cleared, and each 32-bit transfer on the PCI bus will contain two 16-bit words from DSP memory; this is the normal mode to use when transferring 16-bit data samples to and from DSP memory while efficiently using host memory. The DPD bit should be set for transferring 24-bit instructions into DSP memory; in this mode the 32-bit transfer on the PCI bus contains a single 24-bit word with the upper 8 bits of PCI unused.

Figure 7-5 on page 7-15 illustrates the DMA bus mastering formats for packed and unpacked data.

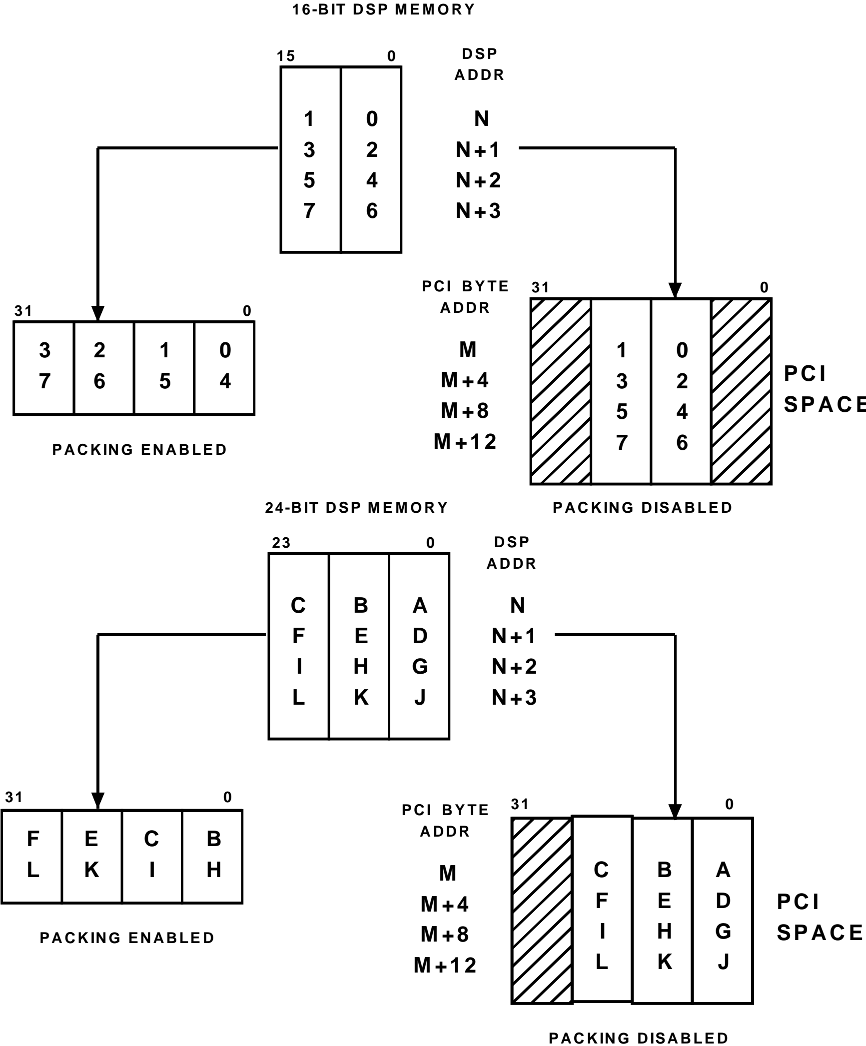


Figure 7-5. DMA Bus Mastering Formats (Packed and Unpacked)

Host Port Scatter-Gather DMA Mode

The SGDEN bit in the PCI_Rx0-1CTL and PCI_Tx0-1CTL registers enables scatter-gather DMA mode.

Each bus master DMA channel includes 4 registers to specify a standard circular buffer in system memory. The Base Address points to the start of the circular buffer. The Current Address is a pointer to the current position within that buffer. The Base Count specifies the size of the buffer in bytes, while the Current Count keeps track of how many bytes need to be transferred before the end of the buffer is reached. When the end of the buffer is reached, the channel can be programmed to loop back to the beginning and continue the transfers. When this looping occurs, a Status bit is set in the DMA Control Register.

When transferring samples to and from DSP memory, the PCI DMA controller can be programmed to perform scatter-gather DMA. This mode allows the data to be split up in memory, and yet able to be transferred to and from the ADSP-2192 without processor intervention. In scatter-gather mode, the DMA controller can read the memory address and word count from an array of buffer descriptors called the Scatter-Gather Descriptor (SGD) table. This allows the DMA engine to sustain DMA transfers until all buffers in the SGD table are transferred.

To initiate a scatter-gather transfer between memory and the ADSP-2192, the following steps are involved:

1. Software driver prepares a SGD table in system memory.

   Each descriptor is eight bytes long and consists of an address pointer to the starting address and the transfer count of the memory buffer to be transferred. In any given SGD table, two consecutive SGDs are offset by eight bytes and are aligned on a 4-byte boundary. Each SGD contains:

   - Memory Address (Buffer Start) – 4 bytes
   - Byte Count (Buffer Size) – 3 bytes

- End of Linked List (EOL) – 1 bit (MSB)
- Flag – 1 bit (MSB – 1)

2. Initialize DMA control registers with transfer specific information such as number of total bytes to transfer, direction of transfer, etc.
3. Software driver initializes the hardware pointer to the SGD table.
4. Engage scatter-gather DMA by writing the start value to the PCI channel Control/Status register.
5. The ADSP-2192 pulls in samples as pointed to by the descriptors as needed by the DMA engine.
6. When the EOL is reached, a status bit is set, and the DMA ends if the data buffer is not to be looped. If looping is to occur, DMA transfers continue from the beginning of the table until the channel is turned off.

   Bits in the PCI Control/Status register control whether or not an interrupt occurs when the EOL is reached or when the FLAG bit is set.

Scatter-gather DMA uses the same four registers as Normal Circular Buffer mode but maps the function of each. In scatter-gather mode the registers are mapped as shown in Table 7-2.

Table 7-2. Normal DMA Mode Versus Scatter-gather DMA Mode

Normal Circular Buffer Mode	Scatter-Gather Mode Function
Base Address	SGD Table Pointer
Current Address	SGD Current Pointer Address
Base Count	SGD Pointer
Current Count	Current SGD Count

In either mode of operation, interrupts can be generated based upon the total number of bytes transferred. Each channel has two 24-bit registers to count the bytes transferred and generate interrupts as appropriate. The Interrupt Base Count register specifies the number of bytes to transfer prior to generating an interrupt. The Interrupt Count register specifies the current number left prior to generating the interrupt. When the Interrupt Count register reaches zero, a PCI interrupt can be generated. Additionally, the Interrupt Count register will be reloaded from the Interrupt Base Count and continue counting down for the next interrupt.

Setting I/O Processor—AC'97 Port Modes

The SRCTLx and STCTLx registers in each DSP core's system control registers control the AC'97 port operating mode for the I/O processor. See "ADSP-2192 DSP Peripheral Registers" on page B-1 for more information about the bits in these registers.

The following bits control AC'97 port I/O processor modes. The control bits in the SRCTLx and STCTLx registers have a one cycle effect latency (take effect on the second cycle after change). Programs should not modify an active DMA channel's bits in the SRCTLx or STCTLx registers; other than to disable the channel by clearing the SDEN bit. To change an inactive AC'97 port's operating mode, programs should clear a AC'97 port's control register before writing new settings to the control register.

- **AC'97 FIFO Connection Enable.** SRCTLx and STCTLx Bit 0 (SPEN) These bits enable or disable the corresponding AC'97 port connection as follows: 00 = disabled, 01 = reserved (disabled), 10 = connect to AC'97, or 11 = reserved (disabled).
- **AC'97 Slot Select.** SRCTLx and STCTLx Bits 7-4 (SSEL) These bits select the AC'97 slot as: 0000–0010=Reserved, 0011=Slot 3, 0100=Slot 4, 0101=Slot 5, 0110=Slot 6, 0111= Slot 7, 1000=Slot 8, 1001=Slot 9, 1010=Slot 10, 1011=Slot 11, 1100=Slot 12, or 1101–1111=Reserved.

- **AC'97 FIFO Interrupt Position.** SRCTLx and STCTLx Bits 10–8 (FIP) These bits set the FIFO level for triggering an interrupt or DMA request as: (#data in FIFO) <= FIP.
- **AC'97 Port DMA Enable.** SRCTLx and STCTLx Bit 18 (SDEN) This bit enables (if set, =1) or disables (if cleared, =0) the AC'97 port's receive DMA.

Host Port DMA Status

The I/O processor monitors the status of data transfers on the host port.

When performing PCI Bus Master DMA transactions, the PCI transfer count register is typically loaded with the same value that is loaded into the MSTRCNT register. In this way, a DSP interrupt can be generated by the MSTRCURCNT register counting to 0 when the entire block of data has been transferred.

When the data is being transferred from the host into DSP memory, this interrupt signals the end of the DMA block transfer. When the data is being transferred from DSP memory into the host, this interrupt only signals that all DMA data has been read from DSP memory; the last few words may still be in the PCI FIFO waiting to be transferred to the host.

The DSP can check that the DMA is completing by checking the DEN bit of the PCI_DMACx register. This bit is cleared by the I/O processor when all words have been transferred to the host.

DMA Controller Operation

DMA sequences start in different ways depending on whether DMA chaining is enabled. When chaining is not enabled, only the DMA enable bit (DEN) allows DMA transfers to occur. A DMA sequence starts when one of the following occurs:

- Chaining is disabled and the DMA enable bit (DEN) transitions from low to high.
- Chaining is enabled, DMA is enabled (DEN=1), and the xxxNXTADDR register address field is written with a non-zero value. In this case, TCB chain loading of the channel parameter registers occurs first.
- Chaining is enabled, the xxxNXTADDR register address field is non-zero, 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.
- Chaining is disabled and the channel's DEN bit transitions from high to low. If the DEN bit goes low (=0) and chaining is enabled, the channel enters chain insertion mode and the DMA sequence continues.

When a program sets the DEN bit (=1) after a single DMA finishes, the DMA sequence continues from where it left off (for non-chained operations only). To start a new DMA sequence after the current one is finished, a program must first clear the DEN enable bit, write new parameters to the registers, then set the DEN bit to re-enable DMA. For chained DMA operations, these steps are not necessary. For more information, see "Chaining DMA Processes" on page 7-22.

If a DMA operation completes and the count register is rewritten before the DMA enable bit is cleared, the DMA transfer will restart at the new count.

Once a program starts a DMA process, the process is influenced by two external controls: DMA channel priority and DMA chaining.

Managing DMA Channel Priority

The DMA channels for each of the DSP's I/O ports negotiate channel priority with the I/O processor using an internal DMA request/grant handshake. Each I/O port (AC'97 port and host port) has one or more DMA channels, with each channel having 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 7-1 on page 7-3 shows the paths for internal DMA requests within the I/O processor.

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

Because more than one DMA channel can make a DMA request in a particular cycle, the I/O processor prioritizes DMA channel service. DMA channel prioritization determines which channel can use the data bus to access memory. DMA channel priority is fixed by DMA channel type (AC'97 port, host port, DSP P0, DSP P1).

Chaining DMA Processes

DMA chaining lets the I/O processor automatically start the next DMA when the current DMA finishes. This feature permits unlimited multiple DMA transfers without processor core intervention. Using chaining, programs can set up multiple DMA operations, and each operation can have different attributes.

To chain together multiple DMA operations, the I/O processor must load the start address of the next DMA into the xxxNXTADDR register and the count for the next DMA into the xxxCNT register before the current DMA completes.

Host Port DMA

The DSP support a number of DMA modes for host port DMA.

The method for setting up and starting a host port DMA sequence varies slightly with the selection of transfer and DMA handshake for the channel. For more detailed information on host port DMA features, see "Setting I/O Processor—Host Port Modes" on page 7-12.

In general, the following sequence describes a typical host to internal DMA operation where a host transfers a block of data into the DSP's internal memory:

1. The DSP writes the DMA channel's (DSP-side) parameter registers (MSTRADDR, MSTRCNT, and optionally MSTRNXTADDR).
2. The host (or DSP) writes the DMA channel's (host-side) parameter registers (PCI_xxxADDR and PCI_xxxCNT) and control registers (PCI_DMACx and PCI_XXXCTL), initializing the channel for receive (TRAN=0).
3. The host (or DSP) sets (=1) the channel's DEN bit enabling the DMA process.

4. The host begins writing data to the PCI bus, which is buffered through the host port.

5. The host port PCI buffer detects data is present and asserts an internal DMA request to the I/O processor.

6. The I/O processor grants the request and performs the internal DMA transfer, emptying the host port PCI buffer FIFO.

In general, the following sequence describes a typical internal to external DMA operation where an external device transfers a block of data from the DSP's internal memory:

1. The DSP writes the DMA channel's (DSP-side) parameter registers (MSTRADDR, MSTRCNT, and optionally MSTRNXTADDR).

2. The host (or DSP) writes the DMA channel's (host-side) parameter registers (PCI_xxxADDR and PCI_xxxCNT) and control registers (PCI_DMACx and PCI_XXXCTL), initializing the channel for transmit (TRAN=1).

3. The host (or DSP) sets (=1) the channel's DEN bit enabling the DMA process.

   Because this is a transmit, setting DEN automatically asserts an internal DMA request to the I/O processor.

4. The I/O processor grants the request and performs the internal DMA transfer, filling the host port PCI buffer's FIFO.

5. The host begins reading data from the PCI bus, which is buffered through the host port.

6. The host port PCI buffer detects that there is room in the buffer (it is now "partially empty") and asserts another internal DMA request to the I/O processor, continuing the process.

AC'97 Port DMA

The DSP support a number of DMA modes for AC'97 port DMA. The method for setting up and starting an AC'97 port DMA sequence varies slightly with the transfer mode for the channel. For more detailed information on AC'97 port DMA features, see "Setting I/O Processor—AC'97 Port Modes" on page 7-18.

In general, the following sequence describes a typical AC'97 codec to internal DMA operation where a codec transfers a block of data into the DSP's internal memory using a AC'97 port:

1. The DSP enables the DMA channel's AC'97 port, setting the port's SPEN bits in the port's SRCTLx register.
2. The DSP writes the DMA channel's parameter registers (RxxADDR, RxxCNT, and optionally RxxNXTADDR) and SRCTLx control register, initializing the channel for receive.
3. The DSP sets (=1) the channel's SDEN bit enabling the DMA process.
4. The codec begins writing data to the Rxx buffer through the AC'97 port.
5. The Rxx buffer detects data is present and asserts an internal DMA request to the I/O processor.
6. The I/O processor grants the request and performs the internal DMA transfer, emptying the Rxx buffer.

In general, the following sequence describes a typical internal to AC'97 DMA operation where a codec transfers a block of data from the DSP's internal memory using a AC'97 port:

1. The DSP enables the DMA channel's AC'97 port, setting the port's SPEN bits in the port's STCTLx register.

2. The DSP writes the DMA channel's parameter registers (TxxADDR, TxxCNT, and optionally TxxNXTADDR) and STCTLx control register, initializing the channel for transmit.

3. The DSP sets (=1) the channel's SDEN bit enabling the DMA process.

   Because this is a transmit, setting SDEN automatically asserts an internal DMA request to the I/O processor.

4. The I/O processor grants the request and performs the internal DMA transfer, filling the Txx buffer.

5. The external device begins reading data from the Txx buffer (through the AC'97 port).

6. The Txx buffer detects that there is room in the buffer (it is now "partially empty") and asserts another internal DMA request to the I/O processor, continuing the process.

8 HOST (PCI/USB) PORT

Overview

The ADSP-2192 can interface with a host computer through its USB port or through its PCI port. Both ports provide access to the host computer via the Peripheral Device Control (PDC) bus, which is connected directly to the IDMA ports on each DSP. The USB port connects through the internal USB interface to the PDC bus, and the PCI port connects through the internal PCI interface to the PDC bus. This chapter describes the PCI parallel interface, and then describes the USB serial interface. The type of bus connection is determined by the BUSMODE[1:0] pins.

Host Port Selection

Four options, selected by the BUSMODE[1:0] pins, specify bus connection to the host. The DSP can detect the BUSMODE configuration and hence the system type by reading bits in the SCFG register (IO space address 0x000).

Table 8-1. BUSMODE Configuration

Bus Type	BUS MODE1	Bus MODE0	SCFG:BUS(1:0) Register field (bits 11:10)
PCI or Mini-PCI	GND	GND	00
CardBus PC-Card	GND	Open	01
Sub-ISA	Open	GND	10
USB Serial Bus	Open	Open	11

Mode Strap Pin Connections

The BUSMODE[1:0] pin status is sampled at Power-On-Reset. These pins should either be left open or tied directly to GND; no external resistors are necessary. The BUSMODE input pins have a weak internal pull-up resistor (to 2.5V Internal VDD), which is activated only during power-on. After power-on, their state is latched, and then the input receiver and its pull-up resistor are disabled. No DC current flows. Even if the pin voltage floats to a mid-state level, no current is dissipated in the input receiver.

PCI Parallel Interface

The ADSP-2192 includes a 33-MHz, 32-bit PCI interface to provide control and data paths between the device and a host CPU. The PCI interface complies with the PCI Local Bus Specification, Revision 2.2. The interface supports bus mastering as well as bus target interfaces. PCI Bus Power Management Interface Specification, Revision 1.1 is supported, and additional features needed by mini-PCI designs are included.

Configuration Spaces

The ADSP-2192 has three separate configuration spaces that can be defined to support user functions by writing to the class code register for that function during bootup. Additionally, during boot time, the DSP can disable one or more of the functions. If only two functions are enabled, they will be functions zero and one. If only one function is enabled, it will be function 0.

Each function contains a complete set of registers in the predefined header region, as defined in PCI Local Bus Specification, Revision 2.2. In addition, each function contains optional registers to support PCI Bus Power Management. Registers that are unimplemented or read-only in one function are similarly defined in the other functions. Table 8-2 describes a typical configuration space.

Table 8-2. PCI Configuration Space

Address	Name	Reset	Comments
0x01-0x00	Vendor ID	0x11D4	Writable from the DSP during initialization
0x03-0x02	Device ID	0x2192	Writable from the DSP during initialization
0x05-0x04	Command Register	0x0	Bus Master, Memory Space Capable, I/O Space Capable
0x07-0x06	Status Register	0x0	Bits enabled: Capabilities List, Fast B2B, Medium Decode
0x08	Revision ID	0x0	Writable from the DSP during initialization
0x0B-0x09	Class Code	0x078000	Writable from the DSP during initialization
0x0C	Cache Line Size	0x0	Read-only
0x0D	Latency Timer	0x0	
0x0E	Header Type	0x80	Multifunction bit set
0x0F	BIST	0x0	Unimplemented
0x13-0x10	Base Address 1	0x08	Register Access for all ADSP-2192 Registers, Prefetchable Memory

Table 8-2. PCI Configuration Space (Continued)

Address	Name	Reset	Comments
0x17-0x14	Base Address 2	0x08	24-bit DSP Memory Access
0x1B-0x18	Base Address 3	0x08	16-bit DSP Memory Access
0x1F-0x1C	Base Address 4	0x01	I/O access for control registers and DSP memory
0x23-0x20	Base Address 5	0x0	Unimplemented
0x27-0x24	Base Address 6	0x0	Unimplemented
0x2B-0x28	Cardbus CIS Pointer	0x1FF03	CIS RAM Pointer - Function 0 (Read Only).
0x2D-0x2C	Subsystem Vendor ID	0x11D4	Writable from the DSP during initialization
0x2F-0x2E	Subsystem Device ID	0x2192	Writable from the DSP during initialization
0x33-0x30	Expansion ROM Base Address	0x0	Unimplemented
0x34	Capabilities Pointer	0x40	Read-only
0x3C	Interrupt Line	0x0	
0x3D	Interrupt Pin	0x1	Uses INTA# Pin
0x3E	Min_Gnt	0x1	Read-only
0x3F	Max_Lat	0x4	Read-only
0x40	Capability ID	0x1	Power Management Capability Identifier
0x41	Next_Cap_Ptr	0x0	Read-only

Table 8-2. PCI Configuration Space (Continued)

Address	Name	Reset	Comments
0x43-0x42	Power Management Capabilities	0x6C22	Writable from the DSP during initialization
0x45-0x44	Power Management Control/ Status	0x0	Bits 15 and 8 initialized only on Power-up
0x46	Power Management Bridge	0x0	Unimplemented
0x47	Power Management Data	0x0	Unimplemented

Interactions Between Functions

Because all functions access and control a single set of resources, potential conflicts may occur in the control specified by the configuration. Table 8-3 on page 8-5 and Table 8-4 on page 8-10 identify the interactions and suggest conflict resolutions. Table 8-3 on page 8-5 identifies the registers in the predefined header space, and Table 8-4 on page 8-10 identifies the interactions in the Power Management registers.

Table 8-3. Configuration Space—Function Interactions

Address	Name	Comments
Vendor ID		Separate registers, no interaction
Device ID		Separate registers, no interaction
Command Register Bit 0	I/O Space Enable	Enables are separate in each function, go along with the function's base addresses
Command Register Bit 1	Memory Space Enable	Enables are separate in each function, go along with the function's base addresses

Table 8-3. Configuration Space—Function Interactions (Continued)

Address	Name	Comments
Command Register Bit 2	Bus Master Enable	Enables are separate in each function, go along with the function's base addresses
Command Register Bit 3	Special Cycles	None of the functions support special cycles, read-only
Command Register Bit 4	Memory Write and Invalidate	No function generates Memory Write and Invalidate commands, read-only
Command Register Bit 5	VGA Palette Snoop	Not applicable, read-only
Command Register Bit 6	Parity Error Response	If any function has the bit set, PERR# may be asserted
Command Register Bit 7	Stepping Control	No address stepping is done, read-only
Command Register Bit 8	SERR# Enable	If any function enables SERR# driver, then SERR# may be asserted
Command Register Bit 0	Fast Back-to-back Enable	No function generates fast back-to-back transactions
Status Register Bit 4	Capabilities List	Read-only.
Status Register Bit 5	66 Mhz Capable	Read-only.
Status Register Bit 6	Reserved	Read-only.
Status Register Bit 7	Fast Back-to-back Capable	Read-only.
Status Register Bit 8	Master Data Parity Error	Separate for each function, no interaction

Table 8-3. Configuration Space—Function Interactions (Continued)

Address	Name	Comments
Status Register Bit 10:9	DEVSEL Timing	Read-only.
Status Register Bit 11	Signaled Target Abort	Separate for each function, no interaction
Status Register Bit 12	Received Target Abort	Separate for each function, no interaction
Status Register Bit 13	Received Master Abort	Separate for each function, no interaction
Status Register Bit 14	Signaled System Error	Separate for each function, set if SERR# enabled and SERR# asserted
Status Register Bit 15	Detected Parity Error	Separate for each function, but set in all functions simultaneously
Revision ID		Read-only.
Class Code		Separate registers, no interaction
Cache Line Size		Read-only.
Latency Timer		Separate for each function, no interaction
Header Type		Read-only.
Base Address 1		In range signal ORed between functions, any function can access memory
Base Address 2		In range signal ORed between functions, any function can access memory
Base Address 3		In range signal ORed between functions, any function can access memory

Table 8-3. Configuration Space—Function Interactions (Continued)

Address	Name	Comments
Base Address 4		In range signal ORed between functions, any function can access memory
Subsystem Vendor ID		Separate registers, no interaction
Subsystem Device ID		Separate registers, no interaction
Capabilities Pointer		Read-only.
Interrupt Line		Separate registers, no interaction
Interrupt Pin		Read-only.
Min_Gnt		Read-only.
Max_Lat		Read-only.

Base Address Registers

Each function contains four base address registers used to access ADSP-2192 control registers and DSP memory. Base Address Register 1 (BAR1) points to the control registers; the address specified for each of the functions is an offset from BAR1. PCI memory-type accesses read and write the registers. Byte-wide accesses to the control registers are supported only for those registers within the PCI interface itself.

DSP memory accesses use BAR2 or BAR3 of each function. BAR2 is used to access 24-bit DSP memory, and BAR3 accesses 16-bit DSP memory. The lower half of the allocated space pointed to by each DSP memory BAR is the DSP memory for DSP #1. The upper half is the memory space associated with DSP #2. PCI transactions to and from DSP memory use the DMA function within the DSP core. Each word transferred to or from PCI space uses a single DSP clock cycle to perform internal DSP data transfer. Byte-wide accesses to DSP memory are not supported.

I/O type accesses are supported via BAR4. Both the control registers accessible via BAR1 and the DSP memory accessible via BAR2 and BAR3 can be accessed with I/O accesses. Indirect access is used to read and write the control registers and the DSP memory. For control register accesses, an address register points to the word to be accessed and a separate register is used to transfer the data. Read/write control is part of the address register. Only 16-bit accesses are possible via the I/O space. A separate set of registers performs the same function for DSP memory access. Control for these accesses includes a 24-bit/16-bit select as well as direction control. The data register for DSP memory access is 24-bits wide. 16-bit accesses are loaded into the lower 16 bits of the register.

Peripheral Device Control Registers

The Peripheral Device Control Register space is distributed throughout the ADSP-2192 and connected through the Peripheral Device Control Bus. The PCI bus can access the Peripheral Device Control Registers directly. PCI Base Address Register 1 (BAR1) points to the Control Registers (including the Peripheral Device Control Registers). PCI register accesses are byte wide. PCI register addresses are 24 bits long. Registers can be accessed only in PCI Bus Target/Slave mode.

Power Management Interactions

Conflicts can occur with three functions. Table 8-4 on page 8-10 identifies these potential conflicts and provides suggested resolutions.

Target accesses to registers and DSP memory can go through any function. If the Memory Space access enable bit is set in that function, PCI memory accesses (whose address matches the locations programmed into functions BAR[3:1]) can read or write any visible register or memory location within the ADSP-2192. Similarly, if I/O Space access enable is set, PCI I/O accesses can be performed via BAR4.

There are interactions within the Power Management section of the configuration blocks. The device stays in the highest power state of the three functions. When one of the functions is in a low power state, it can respond only to configuration accesses, regardless of the power state of the other functions. Similarly, when a function transitions from power management state D3 to D0 (see Chapter 11 “System Design”), that function’s configuration space is reinitialized. Each function has a separate PME enable bit and PME status bit. When no determination is possible, both PME status bits are set.

Table 8-4. Power Management—Function Interactions

Name	Register Bits	Comments
Capability ID		Read-only.
Next_Cap_Ptr		Read-only.
Version	Power Management Capability Bits 2:0	Read-only.
PME Clock	Power Management Capability Bit 3	Read-only.
Reserved	Power Management Capability Bit 4	Read-only.
Device Specific Initialization	Power Management Capability Bit 5	Read-only.
Aux Current	Power Management Capability Bit 6	Read-only by PCI, writable by DSP.
D1 Support	Power Management Capability Bit 9	Read-only.
D2 Support	Power Management Capability Bit 10	Read-only.

Table 8-4. Power Management—Function Interactions (Continued)

Name	Register Bits	Comments
PME Support	Power Management Capability Bits 15:11	Read-only by PCI, writable by DSP.
Power State	Power Management Control/Status Bit 1:0	Part will be in highest power state of the three functions
Reserved	Power Management Control/Status Bit 7:2	Read-only, no interaction
PME Enable	Power Management Control/Status Bit 8	Separate for each function, no interaction
Data Select	Power Management Control/Status Bit 12:9	Read-only, no interaction
Data Scale	Power Management Control/Status Bit 14:13	Read-only, no interaction
PME Status	Power Management Control/Status Bit 15	Separate for each function, may be set in all functions by a wakeup

PCI Clock Domain

Figure 8-1 on page 8-12 shows the relationship of the PCI clock to the ADSP-2192 internal clocks.

This domain is driven from the PCI CLK input pin, with a nominal frequency of 33 MHz. This frequency may vary from system to system. There are no controls inside the ADSP-2192 to control this clock since it is entirely under control of the Host operating system and BIOS. The PCI Clock, under control of the CLKRUN signal, may stop when the PC bus segment is powered down. This clock domain is active in PCI and CardBus modes. In Sub-ISA mode, the CLK input is not active (tied to GND). In USB mode, the PCI clock domain is inactive.

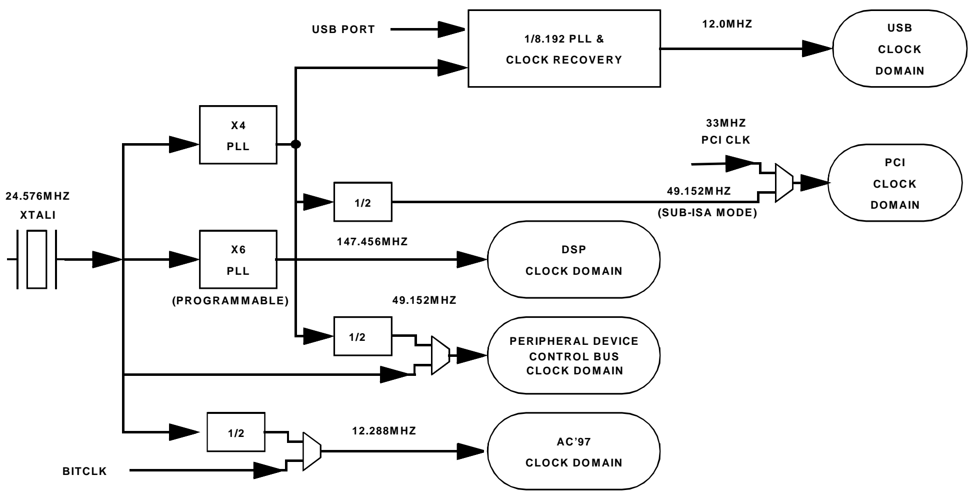


Figure 8-1. Clock Domains

Peripheral Device Control Register Access

Because each DSP may try to access the I/O registers simultaneously, hardware has been added to control access to the register bus and eliminate conflicts. The PCI interface and the USB interface access the same set of registers and use the same internal register access bus. (The USB interface cannot access registers in the PCI clock domain, and the PCI interface cannot access registers within the USB clock domain.)

Prioritization between the different possible masters is fixed. The priorities from highest to lowest are: DSP #1, DSP #2, Host (PCI/USB) interface.

When employed in a multiprocessing system, the ADSP-2192 must perform uninterrupted sequences of register bus accesses. Although these accesses may be required infrequently, they are essential in certain cases. To this end, a Lock function is provided.

To Lock the Bus, the DSP writes a "1" to Bit7 of the FLAGS register. This Flag output bit is assigned to the Bus Lock Request functionality and generates a continuous request on the Peripheral Device Control Bus. Once the Bus is granted to the DSP, it remains granted until the bit is cleared. The DSP can check to see if the bus has been granted by examining Bit15 of the FLAGS register. A "1" in this bit indicates that the bus has been granted to the DSP.

Once the Bus is Locked, the DSP can perform Read-Modify-Write operations without the danger of the other DSP or the PCI/USB interface changing the register. Avoid locking the Bus for extended periods of time.

The PDC bus is a shared resource that affects almost all aspects of operation of an ADSP-2192 system. Transactions may be initiated by either DSP or by the external bus interface (whether PCI, USB or sub-ISA). Transactions may be targeted at registers for PCI, USB, system control, AC'97, and GPIO functions. Only one transaction may be in progress at a time; all other initiators must wait for the current transaction to complete.

Each DSP IO space transaction is translated into a PDC bus transaction in hardware. The initiating DSP waits for its IO Acknowledge (IOACK) signal until the transaction is granted and completed. While waiting, DSP execution is halted at the IO space access instruction, and the DSP cannot complete any other task. In this state, interrupts are ignored, DMA cannot take place, and no other instruction may execute.

Example

DSP#2 is in the middle of a PDC bus transaction, and every initiator wants to launch a PDC bus transaction. DSP #2 cannot start a new transaction until the current IO space access instruction completes. DSP #1 halts, waiting for IOACK, until its transaction is granted and completed. DSP #1 may attempt to lock the bus and check for bus lock status. If the bus was not locked by DSP #1, it assumes that a transaction is in progress and may remove the lock request and try again later.

The PCI bus cannot initiate a transaction and is issued a Retry semantic. The USB bus cannot initiate a transaction and is issued a Retry semantic. The sub-ISA bus cannot initiate a transaction and does not receive an ISA acknowledge until the transaction is granted and completed.

A zero wait state PDC transaction consumes approximately 12 DSP cycles. Most transactions have zero wait states, but some transactions (those that must be mapped through external interfaces, for example) may be much longer.

Resets

In addition to power-on and system resets described in other chapters, the ADSP-2192 can be reset by the PCI or USB bus. The Reset Handler that gets executed is dictated by the CRST[1:0] bits in the Chip Mode/Status Register (CMSR). For PCI or USB Reset, set the CRST bits to [1:0].

Interrupts

Table 8-5 on page 8-15 shows a variety of potential sources of interrupts to the PCI host. The PCI Interrupt Register consolidates all of the possible interrupt sources and the bits of this register to a single interrupt pin, INTA#, used to signal the interrupts back to the host. The register bits are set by the various sources and can be cleared by writing a 1 to the bits to be cleared.

Interrupts may be sensitive either to edges or levels, as indicated in Table 8-5 on page 8-15. The PCI GPIO interrupt is level sensitive, and is asserted when any of the GPIO's individual sticky status bits is true. If an interrupt service routine is in the process of acknowledging one GPIO interrupt (by clearing its sticky status and then writing a 1 to PCIINT:GPIO) while an event occurs on another GPIO, it is possible for the ISR to miss the second event, should it occur between the time the ISR reads the GPIO's status and when the ISR clears the PCIINT:GPIO bit.

The GPIO interrupt is level sensitive to accommodate this case; the PCI-INT:GPIO interrupt bit and the INTA# pin immediately reassert after clearing. The ISR may be written in two ways to detect this case: it may exit and be immediately retriggered, or it may read back the PCIINT register after the clear to see if any bit has been set again, which indicates the occurrence of some new interrupt.

Table 8-5. PCI Interrupt Register

Bit	Name	Comments[1]	Sensitivity
15	Reserved		
14	PCI Target Abort Interrupt	PCI Interface Target Abort Detected	Edge
13	PCI Master Abort Interrupt	PCI Interface Master Abort Detected	Edge
12	AC'97 Wakeup Edge	AC'97 Interface Initiated	Edge
11	GPIO Wakeup	I/O Pin Initiated Level	Level
10	Reserved		
9	Reserved		
8	Outgoing Mailbox 1 PCI Interrupt	DSP to PCI Mailbox 1 Transfer	Edge
7	Outgoing Mailbox 0 PCI Interrupt	DSP to PCI Mailbox 0 Transfer	Edge
6	Incoming Mailbox 1 PCI Interrupt	PCI to DSP Mailbox 1 Transfer	Edge
5	Incoming Mailbox 0 PCI Interrupt	PCI to DSP Mailbox 0 Transfer	Edge

Table 8-5. PCI Interrupt Register (Continued)

Bit	Name	Comments[1]	Sensitivity
4	Tx1 DMA Channel Interrupt	Transmit Channel 1 Bus Master Transactions	Edge
3	Tx0 DMA Channel Interrupt	Transmit Channel 0 Bus Master Transactions	Edge
2	Rx1 DMA Channel Interrupt	Receive Channel 1 Bus Master Transactions	Edge
1	Rx0 DMA Channel Interrupt	Receive Channel 0 Bus Master Transactions	Edge
0	Reserved		

[1] The Interrupt Status is Latched even when the Interrupt Source is not enabled. Therefore, the Interrupt should be cleared before being enabled unless previous Interrupt history is considered important.

PCI Control Register

The PCI Control Register must be initialized by the DSP ROM code prior to PCI enumeration. (It has no effect in ISA or USB mode.) Once the Configuration Ready bit is set to 1, Bits[2:0] of the PCI Control Register become read-only, and further write access by the DSP to configuration space is disallowed.

Table 8-6. PCI Control Register

Bit	Name	Comments
15	Reserved	
14	TAbort IEN PCI Target Abort Interrupt Enable.	PCI Interface Target Abort Detect Int. Enabled.

Table 8-6. PCI Control Register (Continued)

Bit	Name	Comments
13	MAbort IEN PCI Master Abort Interrupt Enable.	PCI Interface Master Abort Detect Int. Enabled.
12	AC’97 IEN AC’97 Interrupt Enable.	AC’97 Interface Initiated Interrupt Enabled.
11	GPIO IEN GPIO Interrupt Enable.	I/O Pin Initiated Interrupt Enabled.
10	Reserved	
9	Reserved	
8	D2PM1 IEN Mailbox 1 PCI Interrupt Enable.	DSP to PCI Mailbox 1 Transfer Interrupt Enabled.
7	D2PM0 IEN Mailbox 0 PCI Interrupt Enable.	DSP to PCI Mailbox 0 Transfer Interrupt Enabled.
6	P2DM1 IEN Mailbox 1 PCI Interrupt Enable.	PCI to DSP Mailbox 1 Transfer Interrupt Enabled.
5	P2DM0 IEN Mailbox 0 PCI Interrupt Enable.	PCI to DSP Mailbox 0 Transfer Interrupt Enabled.
4	Reserved	
3	Reserved	
2	Conf Rdy Configuration Ready	When 0, disables PCI accesses to the ADSP-2192 (terminated with Retry). Must be set to 1 by DSP ROM code after initializing configuration space. Once 1, cannot be written to 0.
1-0	PCIF[1:0] Number of PCI Functions Configured	00 = one PCI Function enabled, 01= two functions, 10= three functions

PCI Port Priority on the PDC Bus

The PCI port shares use of the Peripheral Device Control (PDC) bus with the two DSPs and with other I/O ports. Transactions may be initiated by either DSP or by the external bus interface (whether PCI, USB, or Sub-ISA). Transactions may be targeted at registers for PCI, USB, system control, or serial port functions. Only one transaction may be in progress at a time; all other initiators must wait for the current transaction to complete.

The prioritization between the different possible masters is fixed. The priorities from highest to lowest are: DSP #1, DSP #2, Host (PCI/USB) interface.

The syntax for access to the PDC registers is described in "ADSP-2192 DSP Peripheral Registers" on page B-1.

DSP Mailbox Registers

The DSP Mailbox registers allow you to construct an efficient communications protocol between the PCI device driver and the DSP code. The mailbox functions consist of InBox0, InBox1, OutBox0, OutBox1, a status register, and a control register.

InBoxes

The incoming mailboxes (InBox0 and InBox1) are 16 bits wide. They may be read or written by the PCI device or the DSP core. PCI writes to the InBoxes may generate DSP interrupts. DSP reads of InBoxes may generate PCI interrupts.

OutBoxes

The outgoing mailboxes (OutBox0 and OutBox1) are 16 bits wide. They may be read or written by the PCI device or the DSP core. DSP writes to the OutBoxes may generate PCI interrupts.

PCI reads of OutBoxes may generate DSP interrupts with special handling. The PC host must perform the following sequence when reading an outbox:

1. Read OutBox
2. Write a 1 to the OutBox Valid bit to clear it

PCI reads of OutBoxes cannot generate interrupts directly, as they would be "read side-effects" which are prohibited in the PCI Specification.

Status

This register consists of read/write-one-clear status bits (denoted `R/WC`). A read/write-one-clear bit is cleared when a one is written to it. Writing a zero has no effect. See "ADSP-2192 DSP Peripheral Registers" on page B-1 for the bit names of the `MBXSTAT` register.

Table 8-7. Mailbox Status Register

Bit	Type	Description
0	R/WC	InBox0 PCI Interrupt Pending. This bit is set when the DSP reads valid data from InBox0, if enabled by the corresponding Mailbox Control Register bit.
1	R/WC	InBox1 PCI Interrupt Pending. This bit is set when the DSP reads valid data from InBox1, if enabled by the corresponding Mailbox Control Register bit.

Table 8-7. Mailbox Status Register (Continued)

Bit	Type	Description
2	R/WC	OutBox0 PCI Interrupt Pending. This bit is set when the DSP writes valid data to OutBox0, if enabled by the corresponding Mailbox Control Register bit.
3	R/WC	OutBox1 PCI Interrupt Pending. This bit is set when the DSP writes valid data to OutBox1, if enabled by the corresponding Mailbox Control Register bit.
4	R/WC	InBox0 DSP 1 Interrupt Pending. This bit is set when the PCI writes valid data to InBox0, if enabled by the corresponding Mailbox Control Register bit.
5	R/WC	InBox1 DSP 1 Interrupt Pending. This bit is set when the PCI writes valid data to InBox1, if enabled by the corresponding Mailbox Control Register bit.
6	R/WC	OutBox0 DSP 1 Interrupt Pending. This bit is set when the PCI acknowledges reading data from OutBox0 by writing a 1 to bit 14, if enabled by the corresponding Mailbox Control Reg bit.
7	R/WC	OutBox1 DSP 1 Interrupt Pending. This bit is set when the PCI acknowledges reading data from OutBox1 by writing a 1 to bit 15, if enabled by the corresponding Mailbox Control Reg bit.
8	R/WC	InBox0 DSP 2 Interrupt Pending. This bit is set when the PCI writes valid data to InBox0, if enabled by the corresponding Mailbox Control Register bit.
9	R/WC	InBox1 DSP 2 Interrupt Pending. This bit is set when the PCI writes valid data to InBox1, if enabled by the corresponding Mailbox Control Register bit.

Table 8-7. Mailbox Status Register (Continued)

Bit	Type	Description
10	R/WC	OutBox0 DSP 2 Interrupt Pending. This bit is set when the PCI acknowledges reading data from OutBox0 by writing a 1 to bit 14, if enabled by the corresponding Mailbox Control Reg bit.
11	R/WC	OutBox1 DSP 2 Interrupt Pending. This bit is set when the PCI acknowledges reading data from OutBox1 by writing a 1 to bit 15, if enabled by the corresponding Mailbox Control Reg bit.

Control

This register consists of read/write interrupt enable control bits (denoted R/W). See "ADSP-2192 DSP Peripheral Registers" on page B-1 for the bit names of the MBXCTL register.

Table 8-8. Mailbox Control Register

Bit	Type	Description
0	R/W	InBox0 PCI Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
1	R/W	InBox1 PCI Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
2	R/W	OutBox0 PCI Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
3	R/W	OutBox1 PCI Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
4	R/W	InBox0 DSP #1 Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.

Table 8-8. Mailbox Control Register (Continued)

Bit	Type	Description
5	R/W	InBox1 DSP #1 Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
6	R/W	OutBox0 DSP #1 Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
7	R/W	OutBox1 DSP #1 Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
8	R/W	InBox0 DSP #2 Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
9	R/W	InBox1 DSP #2 Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
10	R/W	OutBox0 DSP #2 Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
11	R/W	OutBox1 DSP #2 Interrupt Enable. When asserted allows the corresponding Interrupt Pending bit to be set.
15:12	RO	Reserved

Indirect Access to I/O Space

PCI I/O access to the ADSP-2192 registers is supported via BAR4. The registers listed in Table 8-9 are directly accessible from BAR4.

Table 8-9. I/O Space Indirect Access Registers

Offset	Name	Reset	Comments
0x03-0x00	IOREGA Control Register Address	0x0000	Address and direction control for registers access
0x07-0x04	IOREGD Control Register Data	0x0000	Data for register access
0x0B-0x08	IOMEMA DSP Memory Addresses	0x00000000	Address and direction control for DSP memory access
0x0F-0x0C	IOMEMD DSP Memory Data	0x000000	Data for DSP memory access

To access the PCI I/O space registers:

1. Write `IOREGA` with the register address, setting bit 14 for direction control. If the access is a read, the register is prefetched into the `IOREGD` register.
2. Read the `IOREGD` register for the appropriate data. The Ready status is not necessary in PCI mode, since the `IOREGD` access is retried until the data is ready.
3. For writes, the register transaction is initiated when the `IOREGD` register is written.

To access PCI Memory:

1. Write `IOMEMA` with the DSP Memory address, setting the read/write control and the 16/24-bit control appropriately.
2. The data is prefetched for reads. The address automatically increments for memory accesses, allowing subsequent `IOMEMD` reads from the subsequent locations.
3. Consecutive writes to `IOMEMD` are written to consecutive memory locations.

Table 8-10. BIT Organization of PCI I/O Space Registers

Register	Bits	Function
IOREGA	15	Ready Status
	14	Write/Read
	13:0	PDC Address
IOREGD	15:0	IO Data
IOMEMA	23	Write/Read
	22	16bit/24bit
	17:0	DSP Memory Address
IOMEMD	23:0	Memory Data

USB Interface

Overview

The USB port on the ADSP-2192 complies with the Universal Bus Specification, Version 1.1 and allows you to interface with a compatible host. An 8051 compatible MCU is supported on board, which allows you to soft download different configurations and support any number of class specific commands.

In addition to the 8051 core, the interface includes USB accessible registers, an interrupt subsystem, configuration and clock control, and a data path that allows USB Endpoint data transfer directly between the DSP internal memory and a USB-host. The module interfaces with an on-chip USB transceiver on the USB side and the DMA and PDC bus on the ADSP-2192 system side.

USB Requirements

This section describes some features of the protocol upon which this USB implementation has been based. A separate reference section lists the resources that provide the detailed description of the USB.

USB is a master-slave bus, in which a single master generates data transfer requests to the attached slaves and allocates bandwidth on the serial cable according to a specific algorithm. The bus master is referred to as the USB host, and the bus slaves are referred to as USB devices. Each USB device implements one or more USB Endpoints which are akin to virtual data channels. Each Endpoint on a USB device operates independently of all others.

Data flows between the USB Host and attached devices in packets that are 8, 16, 32, or 64 bytes. The packets are grouped into larger units called transfers.

The USB Host implements a traffic scheduling algorithm to allocate the serial bus bandwidth fairly across all of the attached USB devices and Endpoints. From the point of view of a USB device, this algorithm is not deterministic. While the specific scheduling algorithm is standardized, the bus dynamically reallocates bandwidth based on criteria such as packet error conditions and flow control. As far as the device is concerned, it can transfer requests for any Endpoint at any time. Depending upon the bus loading at any given time, the USB Host may request packets back-to-back, or it may request packet transfers to each Endpoint in a round-robin fashion. The USB protocol also allows for detection and retransmission of packets in cases of bit errors and flow-control.

Any USB device implementation must maintain state information for each of its Endpoints which allow large data transfers to occur one packet at a time and each packet to be retransmitted, if necessary.

Implementation

The USB module in ADSP-2192 has the following features:

- Control Endpoint for all USB control transactions including downloading application-specific MCU firmware
- 3 Endpoints dedicated to downloading DSP code
- 8 User-Programmable Endpoints for DSP data
- 4 registers per Data Endpoint to define a DSP memory buffer associated with each Endpoint
- Support for all four USB transaction types (Bulk, Control, Interrupt, and Isochronous)

- 8051 compatible microcontroller (having 2K bytes of Program Memory ROM, 4K bytes of Program Memory RAM, and 256 bytes of Data Memory RAM) to support flexible descriptor definitions and USB device requests. Application-specific MCU microcode needs to be downloaded through the USB interface.
- A dedicated hardware block to stream the data into and out of the data Endpoints from the host. This hardware block manages the USB transactions to each data Endpoint and serves as a conduit for the data as it moves from the memory buffers (FIFOs) in DSP memory space. No MCU involvement is required to manage these data pipes.
- Interaction with the DSP using the DMA and PDC buses.
- 12MB/s (run at full speed)
- Support for OHCI and UHCI Hosts

Block Diagram of USB Module

Figure 8-2 on page 8-28 shows a block diagram of the USB module in the ADSP-2192. Each element is described in this section.

USB-SIE

This block interfaces to the outside interface. All the USB data traffic goes through it. Its two paths, I/P and O/P, support both control and data traffic flow. On the Receive side, the clock employs recovery, error checking, and bit stuffing as it decodes the NRZ data. On the transmit side, it does the opposite: NRZ encodes it, appends the CRC, bit-stuffs it if necessary, and transmits the data. It also sends out the sync header and end-of-packet fields.

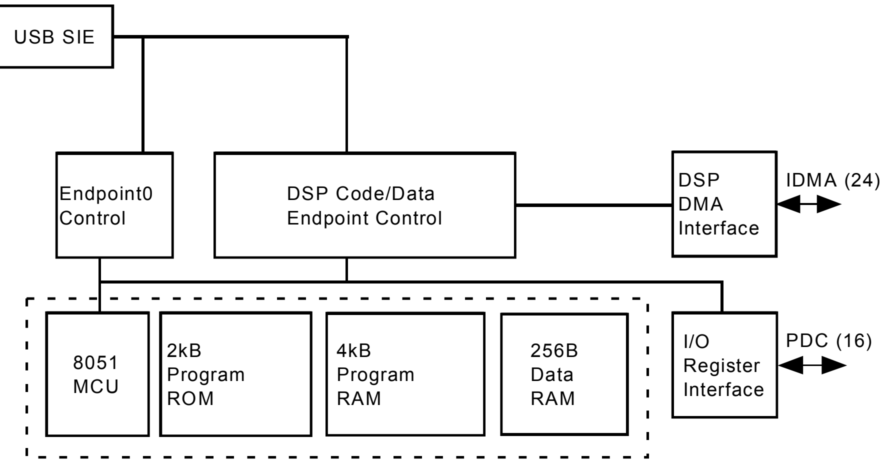


Figure 8-2. ADSP-2192 USB Block Diagram

Endpoint 0 Control

This block manages all traffic duties for Endpoint 0, serving as the communication link between the USB host and the MCU. For host-to-device transfers, it notifies the MCU when valid SETUP commands or OUT packet data arrives for processing by the MCU. In responding to device-to-host transfers, it transmits the proper data packet or handshake packet under MCU guidance.

MCU

The MCU is an 8-bit 8051 compatible MCU within the USB. It acts as the controller for the USB block. It handles all the command and data processing for the Endpoint 0. It also parameterizes the DSP code and data Endpoints. There is no involvement in the actual data transfer for the DSP code and data endpoints.

I/O REG Interface

This is the 16-bit interface to the internal PDC (Peripheral Device Control) bus. All the writing and reading to and from the I/O space is performed through this interface. PDC is a multiplexed address and Data bus.

DSP DMA Interface

This interface is the link between the USB and the internal IDMA bus, which goes into the DSP memory. The serial data collected by the SIE interface is converted into byte-wide packets and sent to DSP DMA interface. Based upon the packet type, program, or Data, the DSP DMA interface converts it into a 3-byte packet (24 bits) or a 2-byte packet (upper 16-bits of the IDMA bus) and then dispatches it to the DSP memory. It also does the retrieval from the Internal Memory 16-bit-wide data, splits it into byte-wide format, and sends it to the SIE. SIE then splits it in serial format and sends it to the host over the serial bus.

DSP Code/Data Endpoint Control

This block manages all traffic duties for both DSP code (via Endpoints 1:3) and DSP data (via Endpoints 4:11). All USB data through this block streams directly to and from DSP memory with no involvement from the MCU. The only requirement from the MCU is to program the personalities (transfer type, maximum packet size, direction, etc.) of all the enumerated Endpoints prior to any USB transactions.

Features and Modes

The different functions supported by the USB module are as follows:

Endpoint Types

The USB supports four different transfer types: Bulk, Control, Interrupt, and Isochronous. The Endpoint 0 (EP0) is configured for Control Type transaction with a fixed packet size of 8. The other user specific Endpoints (EP4-11) can be defined as Bulk, Interrupt, and Isochronous. Endpoints 1, 2, and 3 are reserved for downloading DSP code and are hard-wired to be Bulk Out pipes with a maximum packet size of 64. They cannot be used for data transfers for user-specific purpose.

Data Transfers

The USB supports four different data transfer types: Bulk, Control, Isochronous, and Interrupt. These types are described in the subsections that follow.

Bulk

Bulk transactions guarantee error-free delivery of data between the host and a function Endpoint. Bulk transactions start with the host issuing an `IN` or an `OUT` token. For example, if the host is writing data, it issues an `OUT` transaction followed by a `DATA0` (PID) packet. Upon receiving the packet, the function can respond by:

- Issuing an `ACK` to signify the proper receipt of data. The host can then send the next data packet (if it has any) in the data sequence.
- Not acknowledging the transaction to force a timeout if there is a `CRC` error or bit stuffing errors in the data packet. In this situation, the host retransmits the data packet using the same PID as the failed packet.

Packet sizes for the Bulk data are limited to 8, 16, 32, and 64 bytes. Each USB Endpoint is assigned one of these sizes and may not switch between them. Large data transfers are broken down into units of these basic packet sizes. For example, a 1044 byte file transfer across USB to a 64 byte bulk Endpoint consists of 17 packet transfers as follows: 16 packets of 64 bytes each and 1 packet of 20 bytes. The USB Host assumes that the short packet represents an end of transfer indicator.

Bulk Transactions do not have a guaranteed bandwidth associated with them. They use whatever bandwidth that has been left over after all other types of transactions have been serviced. They are the preferred transaction type if guaranteed error free delivery of data is required.

Isochronous

The isochronous transaction is used when there is a requirement for guaranteed constant bandwidth for the data transfer. These transfers occur in every frame and guarantee one packet transfer per USB frame. The maximum packet size (up to 1023 bytes) is specified in the Endpoint descriptor table. These transfers are not acknowledged. The data in the Isochronous transfers must be error tolerant. In the presence of CRC, an error may be detected, but the data cannot be retransmitted.

Control

Control Transfers are used for short command/control messages between a host and a function. Control transactions start with a setup stage. The format and contents of the packet are defined in the USB specification. If a data stage follows, the transfer is essentially similar to the bulk transfer. The status stage completes the handshaking sequence.

During the setup stage, the USB Host sends an 8-byte setup packet to the USB Device. The packet specifies information such as the types of command (standard-request, vendor-specific, device-class specific), direction and size of the data phase (if any), and the specific command code.

After receiving the command, the device (function) can either process the command and proceed through the data and status phase or ignore the command. Functions cannot issue a NAK or STALL to ignore setup tokens. If the setup token packet is corrupt, it is ignored and a timeout occurs.

Interrupt

In the USB system, only a host can initiate a USB transaction. The function can have the host make regularly scheduled polls of itself. The frequency of the polling is specified by the function during the bus enumeration process. This process consists of a host sending out an IN token packet to the function and the function responding with either a data packet (if available) or a NAK or STALL (if a data packet is not available). This is an Interrupt transaction. The maximum allowable data payload for an interrupt transaction is 64 bytes.

References

The following are references that you might want to use:

- Universal Serial Bus Specification, Revision 1.1, USB Implementers Forum, www.usb.org
- OHCI Specification, Revision 1.0 USB Implementers Forum, www.usb.org
- Tools – Keil Software Developer's Kit P.No. DK51
  Keil Software Inc., 16990 Dallas Parkway, Suite 120, Dallas, TX
- USB Device Class Specifications, www.usb.org.Device Developers can use this to make use of standardized device drivers on the USB host
- USB System Architecture, Don Anderson, Mindshare Inc.

MCU Register Definitions

MCU registers are defined in four memory spaces grouped by the following address ranges:

0x0XXX This address range defines general-purpose USB status and control registers

0x1XXX This address range defines registers that are specific to Endpoint setup and control

0x2XXX This address range defines the registers used for REGIO accesses to the DSP register space

0x3XXX This address range defines the MCU program memory write address space

Table 8-11. USB MCU Register Definitions

Address	Name	Comment
0x0000-0x0007	USB SETUP Token Cmd	8 bytes total
0x0008-0x000F	USB SETUP Token Data	8 bytes total
0x0010-0x0011	USB SETUP Counter	16 bit counter
0x0012-0x0013	USB Control	Misc control including re-attach
0x0014-0x0015	USB Address/Endpoint	Address of device/active Endpoint
0x0016-0x0017	USB Frame Number	Current frame number
0x1000-0x1001	USB EP4 Description	Configures Endpoint
0x1002-0x1003	USB EP4 NAK Counter	
0x1004-0x1005	USB EP5 Description	Configures Endpoint

Table 8-11. USB MCU Register Definitions (Continued)

Address	Name	Comment
0x1006-0x1007	USB EP5 NAK Counter	
0x1008-0x1009	USB EP6 Description	Configures Endpoint
0x100A-0x100B	USB EP6 NAK Counter	
0x100C-0x100D	USB EP7 Description	Configures Endpoint
0x100E-0x100F	USB EP7 NAK Counter	
0x1010-0x1011	USB EP8 Description	Configures Endpoint
0x1012-0x1013	USB EP8 NAK Counter	
0x1014-0x1015	USB EP8 Description	Configures Endpoint
0x1016-0x1017	USB EP9 NAK Counter	
0x1018-0x1019	USB EP10 Description	Configures Endpoint
0x101A-0x101B	USB EP10 NAK Counter	
0x101C-0x101D	USB EP11 Description	Configures Endpoint
0x101E-0x101F	USB EP11 NAK Counter	
0x1020-0x1021	USB EP STALL Policy	
0x1040-0x1043	USB EP1 Code Download Base Address	Starting address for code download on Endpoint 1
0x1044-0x1047	USB EP2 Code Download Base Address	Starting address for code download on Endpoint 2
0x1048-0x104B	USB EP3 Code Download Base Address	Starting address for code download on Endpoint 3

Table 8-11. USB MCU Register Definitions (Continued)

Address	Name	Comment
0x1060-0x1063	USB EP1 Code Current Write Pointer Offset	Current write pointer offset for code download on Endpoint 1
0x1064-0x1067	USB EP2 Code Current Write Pointer Offset	Current write pointer offset for code download on Endpoint 2
0x1068-0x106B	USB EP3 Code Current Write Pointer Offset	Current write pointer offset for code download on Endpoint 3
0x2000-0x2001	USB Register I/O Address	
0x2002-0x2003	USB Register I/O Data	
0x3000-0x3FFF	USB MCU Program Mem	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TB	LT	LT	TY	TY	DR	PS	PS	PS	PS	PS	PS	PS	PS	PS	PS

Figure 8-3. USB Endpoint Description Register

The USB Endpoint Description Register provides the USB core with information about the Endpoint type, direction, and maximum packet size. This register is read/write by the MCU only. This register is defined for Endpoints[4:11].

Table 8-12. USB Endpoint Description Register

PS[9:0]	Maximum packet size for Endpoint
LT[1:0]	Last transaction handshake indicator bits sent by the ADSP-2192: 00 = Clear 01 = ACK 10 = NAK 11 = ERR
TY[1:0]	Endpoint type bits: 00 = DISABLED 01 = ISO 10 = Bulk 11 = Interrupt
DR	Endpoint direction bit: 1 = IN 0 = OUT
TB	Toggle bit for Endpoint. Reflects the current state of the DATA toggle bit.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	NE	ST	NC	NC	NC	NC

Figure 8-4. USB Endpoint NAK Counter Register

The USB Endpoint NAK Counter Register contains the individual NAK count, stall control, and NAK counter enable bits for Endpoints 4-11. This register is read/write by the MCU only.

Table 8-13. USB Endpoint NAK Counter Register

NC[3:0]	NAK counter. Number of sequential NAKs that have occurred on a given Endpoint. When N[3:0] is equal to the base NAK counter NK[3:0] value in the Endpoint Stall Policy register, a zero-length packet or packet less than maxpacketsize will be issued.
ST	A value of 1 means: Endpoint is stalled
NE	1 = Enable NAK counter 0 = Disable NAK counter

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NK	NK	NK	NK	X	X	X	X	X	TB3	TB2	TB1	ST3	ST2	ST1	FE

Figure 8-5. USB Endpoint Stall Policy Register

The USB Endpoint Stall Policy Register contains the base `NAK` count and FIFO error policy bits for Endpoints 4-11. The `STALL` status and Data toggle bits for Endpoints 1-3 are included as well. This register is read/write by the MCU only.

Table 8-14. USB Endpoint Stall Policy Register

ST[3:1]	A value of 1 means the Endpoint is stalled. ST[1] maps to Endpoint 1, ST[2] maps to Endpoint 2, etc.
TB[3:1]	Toggle bit for Endpoint. Reflects the current state of the DATA toggle bit. ST[1] maps to Endpoint 1, ST[2] maps to Endpoint 2, etc.
NK[3:0]	Base NAK counter. Determines how many sequential NAKs are issued before sending zero length packet, or a packet less than the maximum packet size, on any given Endpoint.
FE	FIFO error policy. A value of 1 means: Endpoint FIFO is overrun/underrun, STALL Endpoint

LSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD

MSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	DS	AD

Figure 8-6. USB Endpoint 1 Code Download Base Address Register

The USB Endpoint 1 Code Download Base Address Register contains an 18 bit address which corresponds to the starting location for DSP code download on Endpoint 1. This register is read/write by the MCU only. The most significant bit (`DS` bit) selects either DSP1 PM address space (`DS=0`) or DSP2 PM address space (`DS=1`).

LSW															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD
MSW															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	DS	AD

Figure 8-7. USB Endpoint 2 Code Download Base Address Register

The USB Endpoint 2 Code Download Base Address Register contains an 18-bit address that corresponds to the starting location for DSP code download on Endpoint 2. This register is read/write by the MCU only. The most significant bit (`DS` bit) selects either DSP1 PM address space (`DS=0`) or DSP2 PM address space (`DS=1`).

LSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD

MSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	DS	AD

Figure 8-8. USB Endpoint 3 Code Download Base Address Register

The USB Endpoint 3 Code Download Base Address Register contains an 18-bit address that corresponds to the starting location for DSP code download on Endpoint 3. This register is read/write by the MCU only. The most significant bit (`DS` bit) selects either DSP1 PM address space (`DS=0`) or DSP2 PM address space (`DS=1`).

LSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD

MSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	AD	AD

Figure 8-9. USB Endpoint 1 Code Download Current Write Pointer Offset Register

The USB Endpoint 1 Code Download Current Write Pointer Offset Register contains an 18-bit address that corresponds to the current write pointer offset from the base address register for DSP code download on Endpoint 1. The sum of this register and the EP1 code download base address register represents the last DSP PM location written.

This register is read by the MCU only and is cleared to 3FFFF (-1) when the Endpoint 1 Code Download Base Address Register is updated.

LSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD

MSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	AD	AD

Figure 8-10. USB Endpoint 2 Code Download Current Write Pointer Offset Register

The USB Endpoint 2 Code Download Current Write Pointer Offset Register contains an 18-bit address that corresponds to the current write pointer offset from the base address register for DSP code download on Endpoint 2. The sum of this register and the EP2 code download base address register represents the last DSP PM location written.

This register is read by the MCU only and is cleared to 3FFFF (-1) when the Endpoint 2 Code Download Base Address Register is updated.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	AD	AD

Figure 8-11. USB Endpoint 3 Code Download Current Write Pointer Offset Register

The USB Endpoint 3 Code Download Current Write Pointer Offset Register contains an 18-bit address that corresponds to the current write pointer offset from the base address register for DSP code download on Endpoint 3. The sum of this register and the EP3 code download base address register represents the last DSP PM location written.

This register is read by the MCU only and is cleared to 3FFFF (-1) when the Endpoint 3 Code Download Base Address Register is updated.

The USB SETUP Token Command Register is defined as 8 bytes long and contains the data sent on the USB from the most recent SETUP transaction. This register is read by the MCU only.

Table 8-15. USB SETUP Token Command Register

Byte	7 0
0	bmRequest
1	b Request
2	w Value (L)
3	w Value (H)
4	w Index (L)
5	w Index (H)
6	w Length (L)
7	w Length(H)

If the most recent SETUP transaction involves a data OUT stage, the USB SETUP Token Data Register is defined as 8 bytes long and contains the data sent on the USB during the data stage. This is also where the MCU writes data to be sent in response to a SETUP transaction involving a data IN stage. This register is read/write by the MCU only.

Table 8-16. USB SETUP Token Data Register

Byte	7 0
0	Data 0
1	Data 1
2	Data 2
3	Data 3
4	Data 4
5	Data 5
6	Data 6
7	Data 7

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	C3	C2	C1	C0

Figure 8-12. USB SETUP Counter Register

The USB SETUP Counter Register provides information about the total size of the SETUP transaction data stage. This register is read/write by the MCU only.

C[3:0]	Counter bits.

The counter hardware is a modulo 4-bit down counter used for tallying data bytes in both the IN and OUT data stages of SETUP transactions. As such, the count value stored has different meanings.

IN Transfers:	The MCU loads the counter with the number of bytes to transfer (must be 8 or less since the USB Setup Token Data Register file is 8 bytes maximum). The USB interface then decrements the count value after each byte is transferred to the host.
OUT Transfers:	Starting from a cleared value of 0, the counter is decremented with each byte received from the host, including the two CRC bytes. For example, if 8 bytes are received, the count value progresses from 15, 14, 13, etc. to a value of 6 (inclusive is the 2 CRC bytes). The MCU reads the value and subtracts it from 14 to determine the actual number of data bytes in the USB Setup Token Register file (14 - 6 = 8 bytes).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0

Figure 8-13. USB Register I/O Address Register

The USB Register I/O Address Register contains the address of the ADSP-2192 register to be read/written. This register is read/write by the MCU only.

Table 8-17. USB Register I/O Address Register

A[15]	MCU sets to 1 to notify the PDC Register Interface block to start ADSP-2192 read/write cycle. PDC Register Interface block clears to 0 to notify MCU the read/write cycle has completed.
A[14]	1 = WRITE, 0 = READ
A[13:0]	ADSP-2192 address to read/write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Figure 8-14. USB Register I/O Data Register

The USB Register I/O Address Register contains the data of the ADSP-2192 register that has been read or is to be written. This register is read/write by the MCU only.

Table 8-18. USB Register I/O Data Register

D[15:0]	During READ this register contains the data read from the ADSP-2192, during WRITE this register is the data to be written to the ADSP-2192

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INT	ISE	IIN	IOU	BY	X	X	ER	X	X	X	X	RW	MO	BB	DI

Figure 8-15. USB Control Register

The USB Control Register controls various USB functions. This register is read/write by the MCU only.

Table 8-19. USB Control Register

MO	A value of 1 means: MCU has completed boot sequence and is ready to respond to USB commands
DI	A value of 1 means: Disconnect CONFIG device and enumerate again using the downloaded MCU configuration
BB	A value of 1 means: After reset boot from MCU RAM, 0 = after reset boot from MCU ROM
RW	A value of 1 means: Enables remote wake-up capability, 0 = disables remote wake-up capability
INT	Active interrupt for the 8051 MCU
ISE	Current interrupt is for a SETUP token
IIN	Current interrupt is for an IN token sent with a non zero length data stage
IOU	Current interrupt is for an OUT token received with a non zero length data stage
BY	Busy bit. A value of 1 means: MCU is busy processing a command. USB interface responds with NAK to further IN/OUT requests from the host until MCU clears this bit.
ER	Error in the current SETUP transaction. Generate STALL condition on EP0.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	EP3	EP2	EP1	EP0	A6	A5	A4	A3	A2	A1	A0

Figure 8-16. USB Address/Endpoint Register

The USB Address/Endpoint Register contains the USB address and active Endpoint. This register is read/write by the MCU only.

Table 8-20. USB Address/Endpoint Register

A[6:0]	USB address assigned to device
EP[3:0]	USB last active Endpoint

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	FN1	FN9	FN8	FN7	FN6	FN5	FN4	FN3	FN2	FN1	FN1	FN0

Figure 8-17. USB Frame Number Register

The USB Frame Number Register contains the last USB frame number. This register is read by the MCU only.

Table 8-21. USB Frame Number Register

FN[10:0]	USB frame number

Config USB Device Definitions and Descriptor Tables

CONFIG DEVICE is the default USB device obtained when this function is initially plugged into the HOST. It is a single configuration containing only Endpoint 0. It allows the ADSP-2192 to be enumerated by the HOST. Following are the tables that go with the device. These tables reside in the MCU ROM memory space and are coded in the firmware, which resides on the ROM. Please refer to the USB specification for a description of these tables.

Table 8-22. CONFIG DEVICE Device Descriptor

Offset	Field	Description	Value
0	bLength	Length = 18 bytes	12H
1	bDescriptorType	Type = DEVICE	01H
2 - 3	bcdUSB	USB Specification 1.1	0110H
4	bDeviceClass	Device class vendor specific	FFH
5	bDeviceSubClass	Device sub-class vendor specific	FFH
6	bDeviceProtocol	Device protocol vendor specific	FFH
7	bMaxPacketSize	Maximum packet size for EP0 = 8 bytes	08H
8 - 9	**idVendor (L)**	**Vendor ID (L) = 0456 ADI**	**0456H**
10 - 11	**idProduct (L)**	**Product ID (L) = ADSP-2192**	**2192H**
12 - 13	**bcdDevice (L)**	**Device release number = 1.00**	**0100H**
14	iManufacturer	Manufacturer index string	01H
15	iProduct	Product index string	02H

Table 8-22. CONFIG DEVICE Device Descriptor (Continued)

Offset	Field	Description	Value
16	iSerialNumber	Serial number index string	00H
17	bNumConfigurations	Number of configurations = 1	01H

Offset fields in bold can be overwritten by the Serial EEPROM

Table 8-23. CONFIG DEVICE Interface Descriptor

Offset	Field	Description	Value
0	bLength	Descriptor Length = 9 bytes	09H
1	bDescriptorType	Descriptor Type = Interface	04H
2	bInterfaceNumber	Number of Interface = 0	00H
3	bAlternateSetting	Alternate Setting = 0	00H
4	bNumEndpoints	Number of Endpoints = 0	00H
5	bInterfaceClass	Interface class vendor specific	FFH
6	bInterfaceSubClass	Interface sub-class vendor specific	FFH
7	bInterfaceProtocol	Interface protocol vendor specific	FFH
8	iInterface	Interface Index String	00H

Table 8-24. CONFIG DEVICE String Descriptor Index 0

Offset	Field	Description	Value
0	bLength	Descriptor Length = 4 bytes	04H
1	bDescriptorType	Descriptor Type = String	03H
2	wLANGID[0]	LangID = 0409 (US English)	0409H

Table 8-25. CONFIG DEVICE String Descriptor Index 1 (Manufacturer)

Offset	Field	Description	Value
0	bLength	Descriptor Length = 10 bytes	0AH
1	bDescriptorType	Descriptor Type = String	03H
2-9	bString	ADI	

Table 8-26. CONFIG DEVICE String Descriptor Index 2 (Product)

Offset	Field	Description	Value
0	bLength	Descriptor Length = 22 bytes	16H
1	bDescriptorType	Descriptor Type = String	03H
2-21	bString	USB DEVICE	

Vendor-Specific Commands

In addition to the normally defined USB standard device requests, the following vendor-specific device requests are supported with the use of EP0. These requests are issued from the host driver via normal SETUP transactions on the USB.

Table 8-27. USB MCUCODE (Code Download)

Offset	Field	Size	Value	Description
0	bmRequest	1	0x40	Vendor Request, OUT
1	bRequest	1	0xA1	USB MCUCODE
2	wValue (L)	1	XXX	Address <0:7>
3	wValue (H)	1	XXX	Address <8:15>
4	wIndex (L)	1	0x00	
5	wIndex (H)	1	0x00	
6	wLength (L)	1	0xXX	Length = xx bytes
7	wLength (H)	1	0xXX	

Address <15:0> is the first address where code download begins; the address is incremented automatically after each byte is written. USB MCUCODE is a three-stage control transfer with an OUT data stage. Stage 1 is the SETUP stage, stage 2 is the data stage involving the OUT packet, and stage 3 is the status stage. The length of the data stage is determined by the driver and is specified by the total length of the MCU code to be downloaded.

Table 8-28. USB REGIO (Register Write)

Offset	Field	Size	Value	Description
0	bmRequest	1	0x40	Vendor Request, OUT
1	bRequest	1	0xA0	USB REGIO
2	wValue (L)	1	XXX	Address <0:7>
3	wValue (H)	1	XXX	Address <8:15>
4	wIndex (L)	1	0x00	
5	wIndex (H)	1	0x00	
6	wLength (L)	1	0x02	Length = 02 bytes
7	wLength (H)	1	0x00	

Address <15:15> = 1 indicates a write to the MCU register space, and Address <15:15> = 0 indicates a write to the DSP register space. When accessing DSP register space, the MCU must write the data to be written into the USB Register I/O Data register and write the address to be written to the USB Register I/O Address register. Bit 15 of the USB Register I/O Address register starts the transaction, and bit 14 is set to one to indicate a WRITE. The MCU then polls Bit 15 of the USB Register I/O Address Register looking for a value of 0, which indicates that the write cycle has completed.

USB REGIO (register write) is a three stage control transfer with an OUT data stage. Stage 1 is the SETUP stage, stage 2 is the data stage involving the OUT packet, and stage 3 is the status stage.

Table 8-29. USB REGIO (Register Read)

Offset	Field	Size	Value	Description
0	bmRequest	1	0xC0	Vendor Request, IN
1	bRequest	1	0xA0	USB REGIO
2	wValue (L)	1	XXX	Address <0:7>
3	wValue (H)	1	XXX	Address <8:15>
4	wIndex (L)	1	0x00	
5	wIndex (H)	1	0x00	
6	wLength (L)	1	0x02	Length = 02 bytes
7	wLength (H)	1	0x00	

Address <15:15> = 1 indicates a read from the MCU register space, and Address <15:15> = 0 indicates a read from the DSP register space. When accessing DSP register space, the MCU must write the address to be read to the USB Register I/O Address register. Bit 15 of the USB Register I/O Address register starts the transaction and bit 14 is set to zero to indicate a READ. The data read is placed into the USB Register I/O Data register. The MCU polls Bit 15 of the USB Register I/O Address Register, looking for a value of 0, which indicates that the read cycle has completed.

USB REGIO (register read) is a three-stage control transfer with an IN data stage. Stage 1 is the SETUP stage, stage 2 is the data stage involving the IN packet, and stage 3 is the status stage.

DSP Register Definitions

For each Data Endpoint, four registers provide a memory buffer in the DSP DM (Data Memory) space. These registers are defined for each Endpoint shared by all interfaces for a total of 4x8 = 32 registers. These registers are read/write by the DSP. The USB Data Pipe hardware block also has access to them as part of its buffer management duties.

USB DSP Register Definitions

Table 8-30. USB DSP Register Definitions

Page	Address	Name
0x0C	0x00-0x03	DSP Memory Buffer Base Addr EP4
0x0C	0x04-0x05	DSP Memory Buffer Size EP4
0x0C	0x06-0x07	DSP Memory Buffer RD Offset EP4
0x0C	0x08-0x09	DSP Memory Buffer WR Offset EP4
0x0C	0x10-0x13	DSP Memory Buffer Base Addr EP5
0x0C	0x14-0x15	DSP Memory Buffer Size EP5
0x0C	0x16-0x17	DSP Memory Buffer RD Offset EP5
0x0C	0x18-0x19	DSP Memory Buffer WR Offset EP5
0x0C	0x20-0x23	DSP Memory Buffer Base Addr EP6
0x0C	0x24-0x25	DSP Memory Buffer Size EP6
0x0C	0x26-0x27	DSP Memory Buffer RD Offset EP6
0x0C	0x28-0x29	DSP Memory Buffer WR Offset EP6

Table 8-30. USB DSP Register Definitions (Continued)

Page	Address	Name
0x0C	0x30-0x33	DSP Memory Buffer Base Addr EP7
0x0C	0x34-0x35	DSP Memory Buffer Size EP7
0x0C	0x36-0x37	DSP Memory Buffer RD Offset EP7
0x0C	0x38-0x39	DSP Memory Buffer WR Offset EP7
0x0C	0x40-0x43	DSP Memory Buffer Base Addr EP8
0x0C	0x44-0x45	DSP Memory Buffer Size EP8
0x0C	0x46-0x47	DSP Memory Buffer RD Offset EP8
0x0C	0x48-0x49	DSP Memory Buffer WR Offset EP8
0x0C	0x50-0x53	DSP Memory Buffer Base Addr EP9
0x0C	0x54-0x55	DSP Memory Buffer Size EP9
0x0C	0x56-0x57	DSP Memory Buffer RD Offset EP9
0x0C	0x58-0x59	DSP Memory Buffer WR Offset EP9
0x0C	0x60-0x63	DSP Memory Buffer Base Addr EP10
0x0C	0x64-0x65	DSP Memory Buffer Size EP10
0x0C	0x66-0x67	DSP Memory Buffer RD Offset EP10
0x0C	0x68-0x69	DSP Memory Buffer WR Offset EP10
0x0C	0x70-0x73	DSP Memory Buffer Base Addr EP11
0x0C	0x74-0x75	DSP Memory Buffer Size EP11
0x0C	0x76-0x77	DSP Memory Buffer RD Offset EP11

Table 8-30. USB DSP Register Definitions (Continued)

Page	Address	Name
0x0C	0x78-0x79	DSP Memory Buffer WR Offset EP11
0x0C	0x80-0x81	USB Descriptor Vendor ID
0x0C	0x84-0x85	USB Descriptor Product ID
0x0C	0x86-0x87	USB Descriptor Release Number
0x0C	0x88-0x89	USB Descriptor Device Attributes

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	DS	BA

most significant word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BA	BA	BA	BA	BA	BA	BA	BA	BA	BA	BA	BA	BA	BA	BA	BA

least significant word

Figure 8-18. DSP Memory Buffer Base Addr Register

The DSP Memory Buffer Base Addr Register points to the base address for the DSP memory buffer assigned to this Endpoint.

Table 8-31. DSP Memory Buffer Base Addr Register

[DS, BA16:0]	Memory Buffer Base Address
DS	DSP Memory select bit. 0 = DSP1 memory space, 1 = DSP2 memory space
BA[16:0]	Lower 17 address bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ

Figure 8-19. DSP Memory Buffer Size Register

The DSP Memory Buffer Size Register indicates the size of the DSP memory buffer assigned to this Endpoint.

Table 8-32. DSP Memory Buffer Size Register

SZ[15:0]	Memory Buffer Size

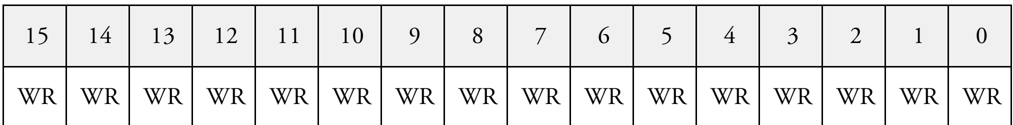
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RD	RD	RD	RD	RD	RD	RD	RD	RD	RD	RD	RD	RD	RD	RD	RD

Figure 8-20. DSP Memory Buffer RD Pointer Offset Register

The DSP Memory Buffer RD Pointer Offset Register provides the offset from the base address for the read pointer of the memory buffer assigned to this Endpoint.

Table 8-33. DSP Memory Buffer RD Pointer Offset Register

RD[15:0]	Memory Buffer RD Offset



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WR	WR	WR	WR	WR	WR	WR	WR	WR	WR	WR	WR	WR	WR	WR	WR

Figure 8-21. DSP Memory Buffer WR Pointer Offset Register

The DSP Memory Buffer WR Pointer Offset Register provides the offset from the base address for the write pointer of the memory buffer assigned to this Endpoint.

Table 8-34. DSP Memory Buffer WR Pointer Offset Register

WR[15:0]	Memory Buffer WR Offset

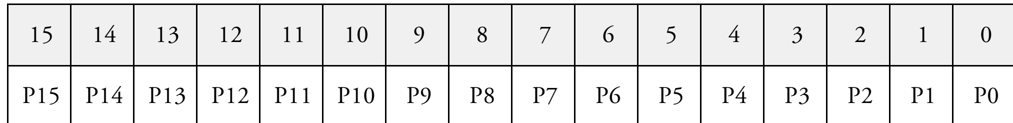
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
V15	V14	V13	V12	V11	V10	V9	V8	V7	V6	V5	V4	V3	V2	V1	V0

Figure 8-22. USB Descriptor Vendor ID

The Vendor ID returned by the GET DEVICE DESCRIPTOR command is contained in the USB Descriptor Vendor ID Register. The DSP can change the Vendor ID by writing to this register during the Serial EEPROM initialization. The default Vendor ID, 0x0456, corresponds to Analog Devices.

Table 8-35. USB Descriptor Vendor ID

V[15:0]	Vendor ID (default = 0x0456)



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

Figure 8-23. USB Descriptor Product ID

The Product ID returned by the GET DEVICE DESCRIPTOR command is contained in the USB Descriptor Product ID Register. The DSP can change the Product ID by writing to this register during the Serial EEPROM initialization. The default Product ID is 0x2192.

Table 8-36. USB Descriptor Product ID

P[15:0]	Product ID (default = 0x2192)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0

Figure 8-24. USB Descriptor Release Number

The Release Number returned by the GET DEVICE DESCRIPTOR command is contained in USB Descriptor Release Number Register. The DSP can change the Release Number by writing to this register during the Serial EEPROM initialization. The default Release Number is 0x0100 which corresponds to version 01.00

Table 8-37. USB Descriptor Release Number

R[15:0]	Release Number (default = 0x0100)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C7	C6	C5	C4	C3	C2	C1	C0	1	SP	RW	0	0	0	0	0

Figure 8-25. USB Descriptor Device Attributes

The device-specific attributes returned by the GET DEVICE DESCRIPTOR command are contained in this register. The DSP can change the attributes by writing to this register during the Serial EEPROM initialization. The default attributes are 0xFA80, which corresponds to bus-powered, no remote wake-up, and maximum power = 500mA.

Table 8-38. USB Descriptor Device Attributes

SP	1 = self-powered, 0 = bus-powered (default = 0)
RW	1 = have remote wake-up capability, 0 = no remote wake-up capability (default = 0)
C[7:0]	Power consumption from bus expressed in 2mA units (default = 0xFA 500mA)

DSP Code Download

Since EP0 has a maximum packet size of 8, downloading DSP code on EP0 can be inefficient when operating on a UHCI controller, which permits only a fixed number of control transactions per frame. Therefore to gain better throughput for code download, downloading of DSP code involves synchronizing a control SETUP command on EP0 with BULK OUT commands on Endpoints 1, 2, or 3. Each Endpoint has an associated DSP download address that is set by using the following SETUP command.

Since three possible interfaces are supported, each interface has its own DSP download address and uses its own BULK pipe to download code. The driver for each interface must set the download address before using the BULK pipe to download DSP code. The download address increments as each byte of data is sent on the BULK pipe to the DSP.

Since DSP instructions are 3 bytes long and USB BULK pipes have even number packet sizes, the instructions to be downloaded must be formatted into 4-byte groups with the least significant byte always zero. The USB interface strips off the least significant bits and formats the DSP instruction before writing it into the program memory. For example, to write the 3-byte opcode, 0x400000, to DSP program memory, the driver sends 0x40000000 down the BULK pipe.

The following example illustrates the proper order of commands and synchronizing that the driver must follow:

1. Device enumerates with two interfaces. Each interface has the capability to download DSP code and can initiate at any time.

2. The driver for Interface 1 begins code download by sending the USB REGIO (write) command with the starting download address. The driver must wait for this command to finish before starting code download.

3. The driver for Interface 2 begins code download by sending the USB REGIO (write) command with the starting download address. The driver must wait for this command to finish before starting code download.

4. Each driver now streams the code to be downloaded to the DSP: Driver 1 onto BULK EP1 for Interface 1, and Driver 2 onto BULK EP2 for Interface 2. The code is written to the DSP in three-byte instructions starting at the location specified by the USB REGIO (Write) command. The driver waits for each command to finish before sending a new code download address.

5. If there is more code to be downloaded at a different starting address, the driver begins the entire sequence, again using steps 1-4.

General Comments

DSP code download is available only after the ADSP-2192 has re-enumerated using application-specific, MCU RAM-based firmware. The DSP code download functionality is not available in the MCU boot ROM for the default CONFIG device.

After setting the download addresses using the USB REGIO (write) command, code download can be initiated for any length using normal BULK traffic.

Starting DSP Code Execution

During the code download phase, both DSPs execute a command polling loop out of DSP ROM. This command polling loop monitors 2 locations of DSP DM, waiting for a Jump instruction to be inserted, which would vector off the respective DSP to the user-application code. In order to write to these locations in DSP DM, two of the 8 USB Data Endpoints would have to be programmed as OUT pipes for downloading the jump code patches; one endpoint is targeted for DSP 1, and the other endpoint is targeted for DSP 2.

Table 8-39. DSP DM Memory Content of the Polled Locations

DSP 1 Address	DSP 2 Address	Contents
0x00000	0x20000	Write JUMP Opcode here
0x00001	0x20001	Write NOP Opcode here (a don't care)
0x00002	0x20002	Write JUMP Address here

Some facts to keep in mind on the behavior of the USB Data Endpoints:

- DSP Data Endpoints follow a pre-increment addressing scheme. The first address written is: Write Pointer + 1 + Base Address
- The Read and Write pointers control data movement in and out of the DSP Memory Buffers and are not allowed to move past one another. Therefore, to download instructions to DM, the Read Pointer must be positioned so that it does not interfere with the auto-incrementing/auto-rolling nature of the Write Pointer. Since this example calls for 3 writes to DM, the Read Pointer must be positioned at least 4 locations greater than the starting Write Pointer value.
- The Size value indicates the length of the memory buffer and is used as a trigger mechanism for both the Write and Read pointers to automatically roll back to the top of the buffer as indicated by the Base Address value.

The following is an outline of the steps involved using EndPoints 4 and 5 to download these Jump patches:

1. Program EndPoints 4 and 5 Data Memory Buffer Registers as follows:

   Base Address = 0x00000 for DSP 1, 0x20000 for DSP 2

   Size Value = 0x0003 (Causes Write Pointer to roll back to top: location 0x00000 for DSP 1, 0x20000 for DSP 2)

   Read Pointer = 0x0010 (A non-essential value that allows the Write Pointer to increment and roll properly.)

   Write Pointer = 0x0000

2. Program EndPoints 4 and 5 Pipe Traits:

Type:	BULK
Direction:	OUT
Maxpacketsize:	64 (Could also be 8,16, or 32)

Steps 1 and 2 can be performed only after MCU RAM-based firmware has been downloaded and the device has been enumerated for a second time. These steps could be programming steps executed as part of MCU startup code. See the section on device initialization process for more details on USB interface configuration.

3. Start USB OUT transaction of a 6-byte packet that contains the NOP Opcode, the Jump Address and the Jump Opcode.
   a. Data packet bytes 1 and 2 are the NOP Opcode and are written to DM locations 0x00001 for DSP 1 or 0x20001 for DSP 2.
   b. Data packet bytes 3 and 4 are the Jump Address and are written to DM locations 0x00002 for DSP 1 or 0x20002 for DSP 2.
   c. Data packet bytes 5 and 6 are the Jump Opcode and are written to DM locations 0x00000 for DSP 1 or 0x20000 for DSP 2.

Once the Jump Opcode is loaded, the DSPs vector to the desired address and begin executing user code. This code can re-program the DSP Memory Buffer Register values of Endpoints 4 and 5 for normal buffer use.

Be sure to load the Jump Address location before loading the Jump Opcode to insure the DSP has a valid address for vectoring. The above programming sequence of the endpoint memory buffer registers provides for this behavior.

MCU ROM Firmware Structure

The MCU ROM firmware is structured into the following three main sections:

- Device, Configuration, Interface, and String Descriptor tables for the default USB config device that is enumerated during initial attachment to the USB bus.
- Code to support all the standard USB commands except:
    - Set Descriptor
    - Sync Frame
    - Get Descriptor support for Interface Descriptor
- Code to support the following vendor specific commands:
    - USB REGIO – Write and Read of chip registers (both MCU space and DSP space)
    - USB MCUCODE – Download of MCU firmware into PM RAM.

Upon initial startup from PM ROM, the MCU performs the following functions prior to responding to any USB control traffic from the host:

1. Clears all 256 bytes of DM
2. Sets the USB interface in the device Default State. (See the USB specification Chapter 9 for details on device states.)
3. Enables its external interrupt 0 which is used to handshake between the USB Endpoint 0 hardware and the MCU.
4. Reads the Serial EEPROM registers and overwrites the changeable fields in the USB descriptor tables for the default config device.

5. Sets the least significant bit of DSP Mailbox register 0x24 as a signal to the DSP that the MCU has completed reading the Serial EEPROM registers and updating the USB descriptor tables.
6. Writes the MCU OK bit (bit 2 of USB Control Register) to signal the USB Endpoint 0 hardware that the MCU has completed initialization and is ready to respond to USB commands.

Once it has finished the above sequence, the MCU enters an idle loop, waiting from an interrupt from the USB Endpoint 0 hardware. When the MCU gets an interrupt, it jumps to its interrupt service routine in which it decodes the USB command and calls the appropriate command service routine.

As stated earlier, the MCU initially sets the USB interface into the device Default State. As the enumeration process with the USB host takes place, the MCU controls the movement of the USB interface from the Default State to the Addressed State, and finally to the Configuration State.

Application-specific MCU RAM-based firmware should be based on the structures used in the MCU ROM firmware for supporting the standard USB commands and the two vendor-specific commands. Typical RAM-based code should be designed as follows:

1. Additional Interface, Endpoint, and/or String Descriptor tables which detail the characteristics of the DSP Code download and Data Endpoints to be enumerated.
2. A copy of the ROM code which supports the standard USB commands and the two vendor-specific commands.

3. Any USB class-specific commands that maybe necessary for this application. These should follow the same programming models used in the ROM for either the standard commands or the vendor-specific commands.

4. Section of code that programs the personalities of all enumerated DSP code and data Endpoints via the Endpoint Description, NAK Counter, and Endpoint Stall Policy registers.

MCU Firmware Programmers Model (Endpoint 0)

The MCU and Endpoint 0 hardware block communicate with each other by a series of signaling bits. For any command that meets the proper USB protocol and contains the correct address, the Endpoint 0 hardware block and MCU begin the following exchange:

1. Endpoint 0 hardware performs unconditional and conditional setting of the following interrupt bits located in the high byte of the USB Control Register:

   a. Always sets `INT` and `BY` bits.

   b. Sets `ISE` if Setup Token arrived indicating new USB command.

   c. Sets `IIN` if Setup Token arrived or `ACK` received from a previous device-to-host transfer of a non-zero length data packet in response to an IN token from the host.

   d. Sets `IOU` if Setup Token arrived or OUT with a non-zero length Data Stage received from the host.

2. MCU jumps to its interrupt service routine and performs the following:
   a. Clears the INT bit of USB Control Register.
   b. Determines if the interrupt is for a new command or a command in progress. If new command, clears its "command_complete" status.
   c. Reads the USB SETUP Token Command Register to determine the command request and performs general error checking of the command fields.
3. MCU vectors off to the appropriate command service routine. The action taken depends on the command being serviced.

The firmware performs error checking on the command fields of the USB SETUP Token Command Register. If it encounters an incorrect field, the MCU instructs the Endpoint 0 hardware block to send the STALL command on the USB bus:

1. Clear USB SETUP Counter Register
2. Set ER bit in the USB Control Register

The structure of the MCU firmware for handling either the standard USB commands (refer to USB Specification, Revision 1.1, Chapter 9) or the two vendor-specific commands maps to one of the following categories:

- OUT type commands with no data stage (USB Specification, Revision 1.1, Chapter 9)
- IN type commands with a single data stage (USB Specification, Revision 1.1, Chapter 9)
- IN type commands with variable-length data stages (USB Specification, Revision 1.1, Chapter 9)

- IN or OUT type commands with a single data stage that are non-Chapter 9 specific.
- MCU Firmware Download to RAM

As noted, the first three types are specific to the standard USB requests outlined in Chapter 9 of the USB specification. Type 4 is for any vendor-specific command or USB class-specific request and is used to access device hardware either local to the USB block (MCU register space) or chip-wide (DSP register space via the PDC bus). Type 5 is specific to downloading MCU firmware to PM RAM.

Type 1: OUTS with No Data Stages (Chapter 9 Specific)

Commands of this type usually involve writes to certain registers or changes to hardware states without the need of additional data stages in the transaction. Once the appropriate task is completed, the MCU performs the following housekeeping steps:

1. Clears the USB Setup Counter Register
2. Clears its internal 'command_complete' status variable
3. Clears the high byte of USB Control Register

MCU returns to its idle loop to await the next interrupt from the Endpoint 0 hardware block. Examples in the MCU firmware that support USB commands of this type:

Set Feature, Clear Feature, Set Address

Type 2: INS with Single Data Stage (Chapter 9 Specific)

1. Read the USB Setup Token Command Register to check for command byte integrity. If any errors, instruct the Endpoint 0 hardware block to issue STALL command:
   a. Clear the USB SETUP Counter Register
   b. Set the `ER` bit in the USB Control Register
   c. Return to the MCU idle loop
2. Load the appropriate data to be transmitted into the USB Setup Token Data Register.
3. Load the USB Setup Counter Register with the number of bytes to be transmitted.
4. Set the internal 'command_complete' status variable.
5. Clear the high byte of the USB Control Register

The MCU returns to its idle loop to await the next interrupt from the Endpoint 0 hardware block.

Examples in the MCU firmware that support USB commands of this type:

Get Status, Get Configuration

Type 3: INS with Variable Length Data Stages (Chapter 9 Specific)

1. Read the upper byte of the USB Control Register to determine if this is the first time entering this code segment. Initial entry is indicated by bits `ISE`, `IIN`, `IOU`, and `BY` set to a 1.

2. If initial entry, read the USB Setup Token Command Register to check for command byte integrity. If any errors, instruct the Endpoint 0 hardware block to issue STALL command:
    a. Clear the USB SETUP Counter Register
    b. Set the ER bit in the USB Control Register
    c. Return to the MCU idle loop
3. If no errors, read USB Setup Token Command Register bytes 6 and 7 to determine the number of bytes to transfer.
4. Begin a data transfer code segment. Re-entry into this segment repeats until all the data is sent. Proper code structure is:
    a. Load the appropriate data to be transmitted into the USB Setup Token Data Register
    b. Load the USB Setup Counter Register with the number of bytes to be transmitted
    c. Set the appropriate state of the internal 'command_complete' status variable. (If last transfer, 'command_complete = 1. If more data to transfer, 'command_complete' = 0).
5. Clear the high byte of the USB Control Register.

MCU returns to its idle loop to await the next interrupt from the Endpoint 0 hardware block. Examples in the MCU firmware that support USB commands of this type:

Get Descriptor

Type 4: INS/OUTS with Single Data Stage for Non-Chapter 9 Specific Commands

1. Read the upper byte of the USB Control Register to determine if this is the first time entering this code segment. Initial entry is indicated by bits `ISE`, `IIN`, `IOU`, and `BY` set to a 1.

2. Determine if the access is to MCU space versus DSP space by interrogating the MSB.

3. USB Setup Token Command Register Byte 3 (1 = MCU, 0 = DSP).

4. If MCU space, convert bytes 2 and 3 of the USB Setup Token Command Register into the proper MCU address. If DSP space, load bytes 2 and 3 directly into the USB Register I/O Address Register.

5. Determine if the access is an OUT or IN by interrogating byte 0 of the USB Setup Token Command Register.

6. If Read from MCU space (an IN transaction):

   a. Load the appropriate data to be transmitted into the USB Setup Token Data Register.

   b. Load the USB Setup Counter Register with the number of bytes to be transmitted.

   c. Set ‘command_complete’ status variable = 1.

   d. Clear the high byte of the USB Control Register.

   e. Return to the MCU idle loop

7. If Read from DSP space (an IN transaction):
   a. Set bit 15 = 1 and bit 14 = 0 of USB Register I/O Address Register to start the read cycle.
   b. Poll bit 15 of the USB Register I/O Address Register until a value of 0 returns, indicating that the read cycle is complete.
   c. Load the USB Setup Token Data Register with the data returned in the USB Register I/O Data Register.
   d. Load USB Setup Counter Register with the number of bytes to be transmitted.
   e. Set 'command_complete' status variable = 1.
   f. Clear the high byte of the USB Control Register.
   g. Return to the MCU idle loop
8. If Write to MCU space (an OUT transaction):
   a. Load the appropriate register with the data from the USB Setup Token Data Register.
   b. Clear the USB Setup Counter Register
   c. Clear its internal 'command_complete' status variable.
   d. Clear the high byte of the USB Control Register
   e. Return to the MCU idle loop.
9. If Write to DSP space (an OUT transaction):
   a. Load the USB Register I/O Data Register with the data from the USB Setup Token Data Register.
   b. Set bit 15 = 1 and bit 14 = 1 of USB Register I/O Address Register to start the write cycle.

c. Poll bit 15 of the USB Register I/O Address Register until a value of 0 returns indicating that the write cycle is complete.

d. Clear the USB Setup Counter Register

e. Clear its internal 'command_complete' status variable.

f. Clear the high byte of the USB Control Register

g. Return to the MCU idle loop.

The vendor specific command USB REGIO is the MCU firmware command that uses the above model. Use it for all non-Chapter 9 specific requests that involve hardware accesses to either MCU space or DSP space in which there is a single data stage.

Type 5: MCU Firmware Download (Variable Length OUT)

1. Read the upper byte of the USB Control Register to determine if this is the first time entering this code segment. Initial entry is indicated by bits `ISE`, `IIN`, `IOU`, and `BY` set to a 1.

2. If initial entry (processing the SETUP Token):

   a. Read the USB Setup Token Command Register bytes 6 and 7 to determine the total length of the code that is to be downloaded.

   b. Read the USB Setup Token Command Register bytes 2 and 3 to determine the starting address in PM memory space to place the code.

   c. Clear the USB Setup Counter Register.

   d. Clear the high byte of the USB Control Register

   e. Return to the MCU idle loop to await the next interrupt from the Endpoint 0 hardware block, which indicates the data has arrived from the host.

3. If not initial entry (processing the data from the OUT transaction):

   a. Read the USB Setup Counter Register to determine how many bytes arrived. (# of bytes transferred = 14 – count value. See USB Setup Counter Register description).

   b. Read the proper number of bytes from the USB Setup Token Data Register and write them to MCU PM RAM.

   c. Determine if the host is trying to send more bytes in the Data Stage than what was told the MCU during the SETUP stage (Length bytes 6 and 7 of the USB Setup Token Command Register).

   d. If true, instruct the Endpoint 0 block to send STALL condition:

      (1.) Clear the USB SETUP Counter Register

      (2.) Set ER bit in the USB Control Register

      (3.) Return to the MCU idle loop

   e. If not true, determine if additional data stages are needed.

      If more data stages are expected:

      (1.) Clear USB Setup Counter Register.

      (2.) Clear the high byte of USB Control Register

      (3.) Return to the MCU idle loop to await the next interrupt from the Endpoint 0 hardware block, which indicates more data has arrived.

If this is the final data stage:

(1.) Clear the USB Setup Counter Register

(2.) Clear its internal ‘command_complete’ status variable.

(3.) Clear the high byte of the USB Control Register

(4.) Return to the MCU idle loop.

This is the structure found in the function called ‘usb_mcucode’ in the MCU ROM firmware.

Example Initialization Process

After attachment to the USB bus, the ADSP-2192 identifies itself as a CONFIG device with 1 Endpoint: one control EP0. This causes a generic user CONFIG driver to load.

The CONFIG driver downloads appropriate MCU code to set up the MCU. This code includes the specific device descriptors, interfaces, and Endpoints.

The external Serial EEPROM is read by the DSP, and the changeable USB descriptive fields are transferred to the MCU. The CONFIG driver, through the control EP0 pipe, generates a register read to determine the configuration value. Based on this configuration code, the host downloads the proper USB configurations to the MCU.

The driver writes the USB Control Register, which causes the device to disconnect and then reconnect so that the new downloaded configuration is enumerated by the system. Each interface, upon enumeration, loads the appropriate user device driver.

An example of this procedure is configuring the ADSP-2192 to be a modem.

1. The ADSP-2192 is attached to USB bus. System enumerates the CONFIG device in the ADSP-2192 first. A user-specific driver is loaded.
2. The user driver reads the device descriptor, which identifies the card as a user-specific device, such as a modem.
3. The user driver downloads USB configuration and MCU code to the MCU for Interface 1, which is the modem. Configuration specifies which Endpoints are used and their definitions. A typical configuration for a modem would be:

Table 8-40. Typical Configuration (Modem)

Endpoint	Type	Maximum Packet Size	Comment
1	BULK OUT	64	DSP CODE
4	BULK IN	64	MODEM RCV
5	BULK OUT	64	MODEM XMT
6	INT IN	16	STATUS

4. The user driver downloads USB configuration for Interface 2, which is the FAX modem. Configuration specifies which Endpoints are used and their definitions. A typical configuration for FAX would be:

Table 8-41. Typical Configuration (FAX)

Endpoint	Type	Maximum Packet Size	Comment
2	BULK OUT	64	DSP CODE
7	BULK IN	64	FAX RCV
8	BULK OUT	64	FAX XMT
9	INT IN	16	STATUS

5. The user driver now writes the USB Config Register, causing the device to disconnect and reconnect. The system enumerates all interfaces and loads the appropriate drivers.
6. The modem driver downloads code to DSP for service. The DSP also initializes the DSP Memory Buffer Base Addr Register, and DSP Memory Buffer Size Register, DSP Memory Buffer RD Pointer Offset, DSP Memory Buffer WR Pointer Offset registers for each Endpoint. Endpoints can be used only when the above registers have been written. Modem service is now available.
7. The FAX driver downloads code to DSP for FAX service. The DSP also initializes the DSP Memory Buffer Base Addr Register, DSP Memory Buffer Size Register, DSP Memory Buffer RD Pointer Offset, DSP Memory Buffer WR Pointer Offset registers for each Endpoint. Endpoints can be used only when the above registers have been written. FAX service is now available.

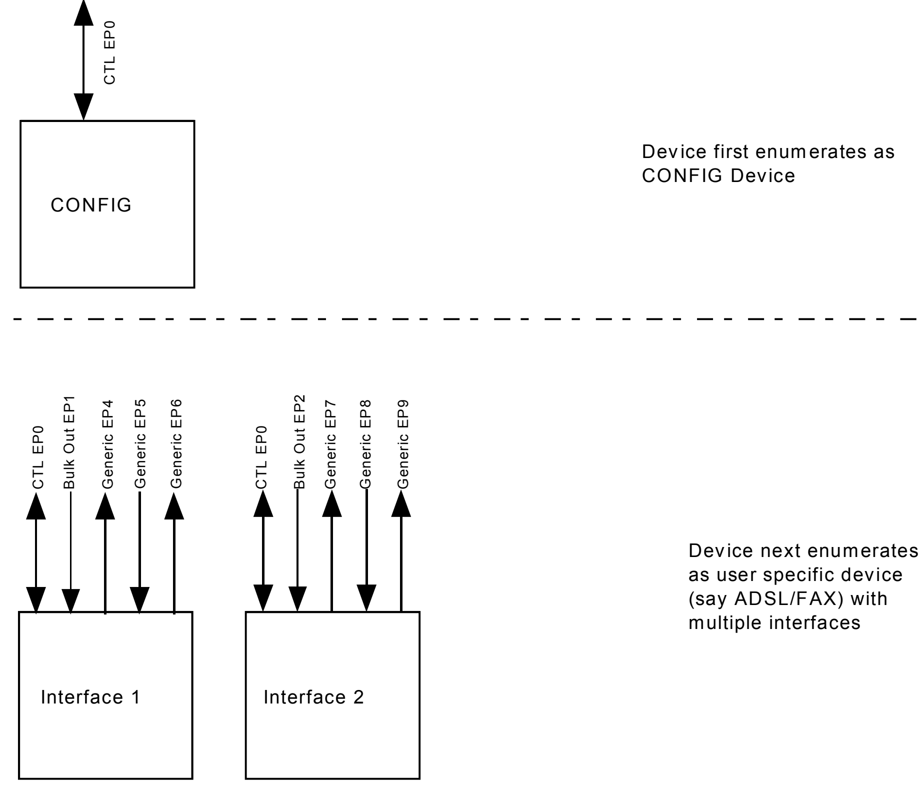


Figure 8-26. ADSP-2192 USB Enumeration

Config Device Definition

Fixed Endpoints

CONTROL Endpoint 0

Type:	Control
Dir:	Bidirectional
Maxpacketsize:	8

Modem Device Definition

Fixed Endpoints

CONTROL Endpoint 0

Type:	Control
Dir:	Bidirectional
Maxpacketsize:	8

BULK OUT Endpoint 1, 2, 3 = Used for code download to DSP

Type:	Bulk
Dir:	OUT
Maxpacketsize:	64

Programmable Endpoints

Generic Endpoints 4, 5, 6, 7, 8, 9, 10, 11

Programmable by:	
Type:	via USB Endpoint Description Register
Direction:	via USB Endpoint Description Register
Maxpacket size:	via USB Endpoint Description Register
Memory Allocation:	via DSP Memory Buffer Base Addr, DSP Memory Buffer Size, DSP Memory Buffer RD Pointer Offset, DSP Memory Buffer Write Pointer Offset Registers

The generic Endpoints are shared between all interfaces.

Serial EEPROM Interface

The Serial EEPROM for the ADSP-2192 can overwrite the information listed below, which is returned during the USB GET DEVICE DESCRIPTOR command. The DSP is responsible during the Serial EEPROM initialization procedure for writing the USB Descriptor Vendor ID, USB Descriptor Product ID, USB Descriptor Release Number, and USB Descriptor Device Attributes registers to change the default settings.

Serial EEPROM Changeable Fields for USB Descriptors

Vendor ID (0x0456 ADI)

Product ID (0x2192)

Device Release Number (0x0100)

Device Attributes (0xFA80)

SP	1 = self-powered, 0 = bus-powered (default = 0)
RW	1 = have remote wake-up capability, 0 = no remote wake-up capability (default = 0)
C[7:0]	Power consumption from bus expressed in 2mA units (default = 0xFA 500mA)

The string descriptors supported in the CONFIG DEVICE are the following and cannot be overwritten by the Serial EEPROM.

Manufacturer	ADI
Product	USB DEVICE

All descriptors can be changed when downloading the RAM based MCU re-numeration code; however the above mentioned restrictions hold for the CONFIG DEVICE.

ADSP-2192 USB Data Pipe Operations

All data transactions involving the generic Endpoints[4:11] stream data into and out of the DSP memory via a dedicated USB hardware block. This hardware block manages all the USB DMA transactions with the DSP memory FIFOs for these Endpoints. While there is no MCU involvement in the management of the data flow through these data pipes, it is the job of the MCU firmware to program the characteristics of these Endpoints via the Endpoint Description, NAK Counter, and Stall Policy registers (MCU Addresses 0x1000-0x1021). Figure 8-27 on page 8-88 is a diagram showing the overall architecture.

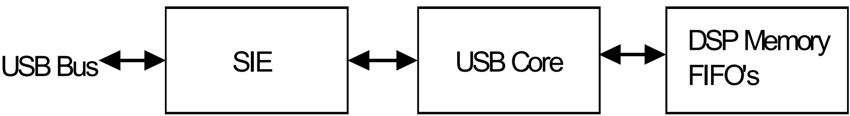


Figure 8-27. USB Data Pipe Architecture

The USB data FIFOs for these generic Endpoints exist in DSP memory space. For each Endpoint, there exists the following memory buffer registers (IO Page 0x0C):

Base Address	**(18 bits)**	
Size	(16 bits)	Offset from the Base Address
Read Offset	(16 bits)	Offset from the Base Address
Write Offset	(16 bits)	Offset from the Base Address

As part of initialization, the DSP code is responsible for setting up these FIFOs before USB data transactions for these Endpoints can begin. As noted in the ADSP-2192 product definition, DSP memory addresses cannot exceed 18 bits (0x000000 - 0x03FFFF). When setting up these USB FIFOs, Base + Size/Read Offset/Write Offset cannot be greater than 18 bits.

The DSP memory interface on the ADSP-2192 only allows reads/writes of 16 bit words. It cannot handle byte transactions. Therefore, a 64 byte maxpacketsize means 32 DSP words. A single byte cannot be transferred to/from the DSP. Endpoint 0 does not have this limitation.

Since these FIFOs exist in DSP memory, the DSP is responsible for sharing some of the pointer management duties with the USB core. For OUT transactions, the write pointer is controlled by the USB core and the read pointer is governed by the DSP. The opposite is true for IN transactions. Both the write and read pointers for each memory buffer would start off as 0.

All USB buffers operate in a circular fashion. Once a pointer reaches the end of the buffer, it needs to be set back to zero. The USB core handles this automatically. You can use the DSP DAGS to control auto incrementing and wrapping of the pointers or you can write code to manipulate them manually.

Below is a listing of Read/Write Pointer characteristics:

1. Both pointers reflect the value of the last memory location on which action took place. The write offset pointer contains the value of the last location written while the read offset pointer contains the value of the last location read.

2. The USB core pre-increments the pointers before using their value. The DSP code that governs pointer control should follow the same model.

3. The USB core recognizes when a memory buffer FIFO is empty when the Read pointer = Write pointer. This has certain ramifications, depending upon whether the USB core is handling an OUT transaction or an IN transaction. This is explained in more detail in the following sections concerning USB traffic direction. The USB core recognizes when a memory buffer FIFO is full when the Write pointer is 1 location behind the Read pointer. The DSP code that governs pointer control needs to mimic this behavior.

4. IO accesses to registers cause DMA transfers to stall for the duration of the IO access. To minimize USB data traffic disturbances, avoid DSP code that programs back-to-back IO accesses. The DSP code that manages the read and write pointer updates must do so via IO access instructions. When writing this code, simply avoid back-to-back IO instructions.

5. Since the memory FIFOs are circular in behavior, the DSP code that calculates either the amount of free space or the amount of available data may need to take into account the Memory Buffer Size value.

   Example

   DSP coding steps required to determine the amount of available data in a memory buffer that has been set up for USB OUT transactions (DSP governing the read pointer).

   Result = Write pointer – Read pointer

   a. If positive, the result directly indicates the amount of available data to process.

   b. If negative, add the buffer size to the result to determine the amount of available data to process. The negative result indicates that the write pointer must have looped back to the top of the FIFO.

In summary, before the USB host can send traffic to the data Endpoints, they need to be programmed in 2 ways:

- MCU: Programs the traits of the Endpoints such as type, direction, maxpacketsize, etc. This is accomplished via writes to the Endpoint Description, NAK Counter, and Stall Policy registers. A convenient method of doing this is to have the MCU firmware program these registers prior to responding to USB control traffic for enumeration on the USB bus.

- DSP: Programs the traits of the DSP DM memory buffers. This is accomplished via IO writes to the Base/Size/Read Offset/Write Offset registers as part of some DSP initialization code.

OUT Transactions (Host to Device)

For OUT transactions, the write pointer is controlled by the USB core, and the read pointer is governed by the DSP. The read and write pointers both start with a value of zero.

When an OUT transaction arrives for a particular Endpoint, the USB core calculates the difference between the write and read pointers to determine how much room there is in the FIFO. If all the OUT data arrives and the write pointer never catches up to the read pointer, that data will be acknowledged (ACK) and the USB core will update the Memory Buffer Write Offset register. If at any time during the transaction the two pointers collide, the USB block will respond with a NAK indicating that the host must resend the same data packet. The write pointer will remain unchanged.

If for some reason the host sends more data than the maxpacketsize, the USB core will accept it as long as there is sufficient room in the FIFO. The USB core will write data to the FIFO to all free locations until the pointers collide. For example, if the FIFO has 32 free bytes and 64 bytes are sent from the host, the USB core will write the first 32 bytes to the FIFO and block the second 32 bytes. During the handshake, the USB core will send a NAK handshake and rewind the write pointer back to its setting prior to the start of the transaction. The USB core never allows the write pointer to be equal to the read pointer as this indicates FIFO empty. A full FIFO is indicated by the write pointer being one location behind the read pointer.

Since the DSP is governing the read pointer, it must perform a similar calculation to determine if there is sufficient data in the FIFO to begin processing. Once it has consumed some amount of data, the DSP will need to update the Memory Buffer Read Offset register. The DSP code cannot program the read pointer to move beyond the write pointer. The DSP can drain all the data from the FIFO, in which case it can program the read pointer to equal the write pointer. Such a condition indicates to the USB core that this FIFO is empty.

IN Transactions (Device to Host)

For IN transactions, the write pointer is controlled by the DSP, and the read pointer is governed by the USB core. The read and write pointers both start with a value of zero.

When an IN transaction arrives for a particular Endpoint, the USB core will again compute how much read data there is available in the FIFO. It will also determine if the amount of read data is greater than or equal to the maxpacketsize. If both conditions are met, the USB core will transfer the data. Upon receiving ACK from the host, the USB core will update the Memory Buffer Read Offset register. The USB core can drain all the data from the FIFO, leaving it empty. It indicates this by setting the read pointer equal to the write pointer.

If the amount of read data is less than the maxpacketsize (a short packet), the USB core will determine whether to send the data based upon a NAK count limit. This is 4-bit field in the Endpoint Stall Policy register that the user can program with a value indicating how many sequential NAKs should be sent prior to transmitting a short packet. The individual endpoint NAK count (`NC` bits of the Endpoint NAK Counter Register) is incremented each time a sequential NAK is sent on that particular endpoint. Once this value exceeds the base NAK count value, a short packet is transmitted. The endpoint-specific `NC` bits are cleared to zero each time a data stage is successfully transmitted for the particular endpoint. This NAK counter system will allow flexibility in how IRPs get retired via short packets.

Along with programming the NK field of the Endpoint Stall Policy Register, the user must program the `NE` (NAK counter enable) bit to enable this counter function. If this bit is set to zero, the USB core will continuously respond with a NAK handshake to the IN token until the number of bytes in the FIFO is greater than or equal to the maxpacketsize.

Since the DSP is governing the write pointer, it must determine if there is sufficient room in the FIFO for placing new data. Once it has completed writes to the FIFO, it needs to update the Memory Buffer Write Offset register. The DSP can fill the FIFO up to the point where the write pointer is one location behind the read pointer. This will be interpreted as a FIFO full condition by the USB core.

Register and Bit #Defines File

The following example definitions file is for the ADSP-2192 DSP PCI. For the most current definitions file, programs should use the version of this file that comes with the software development tools. The version of the file that appears here is included as a guide only.

```
/* ---------------------------------------------------------------------------
def2192_PCI.h - SYSTEM & IOP REGISTER BIT & ADDRESS DEFINITIONS FOR ADSP-2192

Created November 21, 2000. Copyright Analog Devices, Inc.


Note: This file is based on preliminary technical data and is subject to change.
    Updates will be posted on the Analog Devices FTP site:

ftp.analog.com

--------------------------------------------------------------------------------*/
#ifndef __DEF2192_PCI_H_
#define __DEF2192_PCI_H_



/*Chip Control Registers (DSP IOPAGE=0x00)*/

#define   SYSCON    0x00   /*  Chip Mode/Status Register   */
#define   PWRCFG0   0x02   /*  Function 0 Power Management */
#define   PWRCFG1   0x04   /*  Function 1Power Management  */
#define   PWRCFG2   0x06   /*  Function 2 Power Management */
#define   PWRP0     0x08   /*  DSP 0 Interrupt/Power down  */
#define   PWRP1     0x0A   /*  DSP 1 Interrupt/Power down  */
#define   PLLCTL    0x0C   /*  DSP PLL Control    */
#define   REVID     0x0E   /*  AD‘2 Revision ID (read only)  */

/*GPIO Control Registers (DSP IOPAGE=0x00)*/

#define   GPIOCFG      0x10                     /* GPIO Config Direction Control */
                                                             /* 1 = in, 0 = out  */
#define   GPIOPOL      0x12             /* GPIO Polarity (Inputs: 0 = active hi, */
                             /* 1 = active lo; Outputs: 0 = CMOS, 1 = Open Drain) */
#define   GPIOSTKY     0x14           /* GPIO Sticky: 1 = sticky, 0 = not sticky */
#define   GPIOWAKECTL  0x16            /* GPIO Wake Control: 1 = wake-up enabled */
                                                          /* requires sticky set */
#define   GPIOSTAT     0x18                    /*  GPIO Status (Read = Pin state; */
                               /*  Write: 0 = clear sticky status, 1 = no effect) */
#define   GPIOCTL      0x1A  /* GPIO Control(w), Init(r) (Read = Power-on state; */
                                          /* Write : Set state of output pins) */
#define   GPIOPUP      0x1C          /*  GPIO Pull-up Pull-up enable (if input): */
                                                        /* 1 = enable, 0 = hi-Z  */
#define   GPIOPDN      0x1E      /*  GPIO Pull-down Pull-down enable (if input): */
                                                        /* 1 = enable, 0 = hiZ   */

/*PCI/USB Mailbox Registers (DSP IOPAGE=0x00)*/
```

```
#define    MBXSTAT       0x20     /*  Mailbox Status Mailbox Status  */
#define    MBXCTL        0x22     /*  Mailbox Control Mailbox Interrupt Control  */
#define    MBX_IN0       0x24     /*  Incoming Mailbox 0 PCI/USB to DSP mailbox  */
#define    MBX_IN1       0x26     /*  Incoming Mailbox 1 PCI/USB to DSP mailbox  */
#define    MBX_OUT0      0x28     /*  Outgoing Mailbox 0 DSP to PCI/USB mailbox  */
#define    MBX_OUT1      0x2A     /*  Outgoing Mailbox 0 DSP to PCI/USB mailbox  */

/*SERIAL EEPROM Control Register (DSP IOPAGE=0x00)*/

#define    SPROMCTL     0x30     /*  Serial EEPROM I/O Control/Status Direction  */
                             /*  and status  for SEN, SCK, SDA pins. */


/* AC'97 Control Registers (DSP IOPAGE=0x00)*/

#define    AC97LCTL      0xC0     /*  AC'97 Link Control */
#define    AC97LSTAT     0xC2     /*  AC'97 Link Status  */
#define    AC97SEN       0xC4     /*  AC'97 Slot Enable   */
#define    AC97SVAL      0xC6     /*  AC'97 Input Slot Valid */
#define    AC97SREQ      0xC8     /*  AC'97 Slot Request */
#define    AC97GPIO      0xCA     /*  AC'97 External GPIO Register */

/* AC'97 External Codec IO Register Spaces */

#define    AC97CODEC0    0x400       /* External Primary Codec 0 IO page space */
                                                /* registers (0x00 - 0x7F) */
#define    AC97CODEC1    0x500     /* External Secondary Codec 1 IO page space */
                                                /* registers (0x00 - 0x7F) */
#define    AC97CODEC2    0x600     /* External Secondary Codec 2 IO page space */
                                                /* registers (0x00 - 0x7F) */


/* CardBus Function Event Registers(DSP IOPAGE=0x01)*/

#define    CB_EVENT0          0x100     /*  Function 0 Event           */
#define    CB_EVENTMASK0      0x104     /*  Function 0 Event Mask      */
#define    CB_PSTATE0         0x108     /*  Function 0 Present State   */
#define    CB_EVENTFORCE0     0x10C     /*  Function 0 Event Force     */

#define    CB_EVENT1          0x110     /*  Function 1 Event          */
#define    CB_EVENTMASK1      0x114     /*  Function 1 Event Mask     */
#define    CB_PSTATE1         0x118     /*  Function 1 Present State  */
#define    CB_EVENTFORCE1     0x11C     /*  Function 1 Event Force    */

#define    CB_EVENT2          0x120     /*  Function 2 Event          */
#define    CB_EVENTMASK2      0x124     /*  Function 2 Event Mask     */
#define    CB_PSTATE2         0x128     /*  Function 2 Present State  */
#define    CB_EVENTFORCE2     0x12C     /*  Function 2 Event Force    */

/* PCI DMA Address/Count Registers (DSP IOPAGE=0x08)*/

#define   PCI_Rx0BADDRL      0x800    /*  Rx0 DMA Base Address        Bits 15:0   */
#define   PCI_Rx0BADDRH      0x802    /*  Rx0 DMA Base Address        Bits 31:16  */
#define   PCI_Rx0CURADDRL    0x804    /*  Rx0 DMA Current Address     Bits 15:0   */
#define   PCI_Rx0CURADDRH    0x806    /*  Rx0 DMA Current Address     Bits 31:16  */
#define   PCI_Rx0BCNTL       0x808    /*  Rx0 DMA Base Count          Bits 15:0   */
#define   PCI_Rx0BCNTH       0x80A    /*  Rx0 DMA Base Count          Bits 31:16  */
#define   PCI_Rx0CURCNTL     0x80C    /*  Rx0 DMA Current Count       Bits 15:0   */
#define   PCI_Rx0CURCNTH     0x80E    /*  Rx0 DMA Current Count       Bits 31:16  */
```

```
#define   PCI_Tx0BADDRL     0x810    /*  Tx0 DMA Base Address       Bits 15:0   */
#define   PCI_Tx0BADDRH     0x812    /*  Tx0 DMA Base Address       Bits 31:16  */
#define   PCI_Tx0CURADDRL   0x814    /*  Tx0 DMA Current Address    Bits 15:0   */
#define   PCI_Tx0CURADDRH   0x816    /*  Tx0 DMA Current Address    Bits 31:16  */
#define   PCI_Tx0BCNTL      0x818    /*  Tx0 DMA Base Count         Bits 15:0   */
#define   PCI_Tx0BCNTH      0x81A    /*  Tx0 DMA Base Count         Bits 31:16  */
#define   PCI_Tx0CURCNTL    0x81C    /*  Tx0 DMA Current Count      Bits 15:0   */
#define   PCI_Tx0CURCNTH    0x81E    /*  Tx0 DMA Current Count      Bits 31:16  */

#define   PCI_Rx1BADDRL     0x820   /*  Rx1 DMA Base Address     Bits 15:0      */
#define   PCI_Rx1BADDRH     0x822   /*  Rx1 DMA Base Address     Bits 31:16     */
#define   PCI_Rx1CURADDRL   0x824   /*  Rx1 DMA Current Address  Bits 15:0      */
#define   PCI_Rx1CURADDRH   0x826   /*  Rx1 DMA Current Address  Bits 31:16     */
#define   PCI_Rx1BCNTL      0x828   /*  Rx1 DMA Base Count       Bits 15:0      */
#define   PCI_Rx1BCNTH      0x82A   /*  Rx1 DMA Base Count       Bits 31:16     */
#define   PCI_Rx1CURCNTL    0x82C   /*  Rx1 DMA Current Count    Bits 15:0      */
#define   PCI_Rx1CURCNTH    0x82E   /*  Rx1 DMA Current Count    Bits 31:16     */

#define   PCI_Tx1BADDRL     0x830    /*  Tx1 DMA Base Address       Bits 15:0    */
#define   PCI_Tx1BADDRH     0x832    /*  Tx1 DMA Base Address       Bits 31:16   */
#define   PCI_Tx1CURADDRL   0x834    /*  Tx1 DMA Current Address    Bits 15:0    */
#define   PCI_Tx1CURADDRh   0x836   /*  Tx1 DMA Current Address    Bits 31:16
    */
#define   PCI_Tx1BCNTL      0x838    /*  Tx1 DMA Base Count         Bits 15:0    */
#define   PCI_Tx1BCNTH      0x83A    /*  Tx1 DMA Base Count         Bits 31:16   */
#define   PCI_Tx1CURCNTL    0x83C    /*  Tx1 DMA Current Count      Bits 15:0    */
#define   PCI_Tx1CURCNTH    0x83E    /*  Tx1 DMA Current Count      Bits 31:16   */

#define   PCI_Rx0IRQCNTL    0x840    /*  Rx0 DMA Interrupt Count     Bits 15:0    */
#define   PCI_Rx0IRQCNTH    0x842    /*  Rx0 DMA Interrupt Count     Bits 23:16   */
#define   PCI_Rx0IRQBCNTL   0x844   /*  Rx0 DMA Interrupt Base Count Bits 15:0    */
#define   PCI_Rx0IRQBCNTH   0x846   /*  Rx0 DMA Interrupt Base Count Bits 23:16   */

#define   PCI_Tx0IRQCNTL    0x848    /*  Tx0 DMA Interrupt Count     Bits 15:0    */
#define   PCI_Tx0IRQCNTH    0x84A    /*  Tx0 DMA Interrupt Count     Bits 23:16   */
#define   PCI_Tx0IRQBCNTL   0x84C    /*  Tx0 DMA Interrupt Base Count Bits 15:0    */
#define   PCI_Tx0IRQBCNTH   0x84E    /*  Tx0 DMA Interrupt Base Count Bits 23:16   */

#define   PCI_Rx1IRQCNTL    0x850    /*  Rx1 DMA Interrupt Count     Bits 15:0    */
#define   PCI_Rx1IRQCNTH    0x852    /*  Rx1 DMA Interrupt Count     Bits 23:16   */
#define   PCI_Rx1IRQBCNTL   0x854    /*  Rx1 DMA Interrupt Base Count Bits 15:0    */
#define   PCI_Rx1IRQBCNTH   0x856    /*  Rx1 DMA Interrupt Base Count Bits 23:16   */

#define   PCI_Tx1IRQCNTL    0x858    /*  Tx1 DMA Interrupt Count     Bits 15:0    */
#define   PCI_Tx1IRQCNTH    0x85A    /*  Tx1 DMA Interrupt Count     Bits 23:16   */
#define   PCI_Tx1IRQBCNTL   0x85C    /*  Tx1 DMA Interrupt Base Count Bits 15:0    */
#define   PCI_Tx1IRQBCNTH   0x85E    /*  Tx1 DMA Interrupt Base Count Bits 23:16   */

#define   PCI_Rx0CTL         0x860     /*  Rx0 DMA PCI Control/Status   */
#define   PCI_Tx0CTL         0x868     /*  Tx0 DMA PCI Control/Status   */
#define   PCI_Rx1CTL         0x870     /*  Rx1 DMA PCI Control/Status   */
#define   PCI_Tx1CTL         0x878     /*  Tx1 DMA PCI Control/Status   */

/******* PCI_Rx0-1CTL and PCI_Tx0-1CTL Bit definitions ******/

#define SGDEN    0   /* Scatter-gather DMA Enable */
#define LPEN     1   /* Loop Enable */
#define INTMODE1 3   /* Interrupt Mode1 */
#define INTMODE0 2   /* Interrupt Mode0 */
```

```
#define SGVL1    5   /* Current Scatter-gather DMA Valid 1 */
#define SGVL0    4   /* Current Scatter-gather DMA Valid 0 */
#define FLG      6   /* Flag Bit Set in Current Scatter-gather DMA */
#define EOL      7   /* EOL Bit Set in Current Scatter-gather DMA */
/************************************************************/


#define   PCI_MSTRCNT0       0x880     /*  DMA Transfer Count0 Bus master sample */
                                                            /*  transfer count 0 */
#define   PCI_MSTRCNT1       0x882     /*  DMA Transfer Count1 Bus master sample */
                                                            /*  transfer count 1 */
#define   PCI_DMAC0          0x884/*  DMA Control0 Bus master control and status 0 */
#define   PCI_DMAC1          0x886/*  DMA Control1 Bus master control and status 1 */
#define   PCI_IRQSTAT        0x888 /*  PCI Interrupt Reg Status bits for all PCI */
                                                            /* interrupt sources */
#define   PCI_CFGCTL         0x88A      /*  PCI Control Includes config register */
                                                           /* read/write control */


/**************** PCI_DMAC0-1 Bit definitions ************/

#define DEN    0   /* DMA Enable */
#define TRAN   1   /* DMA Direction */
#define FLSH   2   /* Flush FIFO */
#define DSP    3   /* DSP P0/P1 Select */
#define DPD    4   /* DMA Packing Disable, Double Word Mode */
#define CFG2   7   /* Configuration Select 2, 1, or 0 */
#define CFG1   6   /* Configuration Select 2, 1, or 0 */
#define CFG0   5   /* Configuration Select 2, 1, or 0 */
#define EMPTY  8   /* DMA FIFO Empty Status */
#define HALT   9   /* DMA Channel Halt Status */
#define LOOP  10   /* DMA Channel Loop Status */
/************************************************************/

#endif
```

The following example definitions file is for the ADSP-2192 DSP USB. For the most current definitions file, programs should use the version of this file that comes with the software development tools. The version of the file that appears here is included as a guide only.

```
/* --------------------------------------------------------------------------
def2192_USB.h - SYSTEM & IOP REGISTER BIT & ADDRESS DEFINITIONS FOR ADSP-2192

Created November 21, 2000. Copyright Analog Devices, Inc.


Note:   This file is based on preliminary technical data and is subject to change.
    Updates will be posted on the Analog Devices FTP site:

         ftp.analog.com

--------------------------------------------------------------------------*/
```

```
#ifndef __DEF2192_USB_H_
#define __DEF2192_USB_H_

/*Chip Control Registers (DSP IOPAGE=0x00)*/

#define   SYSCON    0x00   /*  Chip Mode/Status Register       */
#define   PWRCFG0   0x02   /*  Function 0 Power Management     */
#define   PWRCFG1   0x04   /*  Function 1Power Management      */
#define   PWRCFG2   0x06   /*  Function 2 Power Management     */
#define   PWRP0     0x08   /*  DSP 0 Interrupt/Power down      */
#define   PWRP1     0x0A   /*  DSP 1 Interrupt/Power down      */
#define   PLLCTL    0x0C   /*  DSP PLL Control                 */
#define   REVID     0x0E   /*  ADSP-2192 Revision ID (read only)  */

/*GPIO Control Registers (DSP IOPAGE=0x00)*/

#define   GPIOCFG      0x10  /*  GPIO Config Direction Control (1 = in, 0 = out) */
#define   GPIOPOL      0x12  /*  GPIO Polarity Inputs: 0 = active hi,          */
                             /*  1 = active lo; Outputs: 0 = CMOS, 1 = Open Drain*/
#define   GPIOSTKY     0x14  /*  GPIO Sticky: 1 = sticky, 0 = not sticky        */
#define   GPIOWAKECTL  0x16  /*  GPIO Wake Control: 1 = wake-up enabled         */
                             /*  (requires sticky set)                          */
#define   GPIOSTAT     0x18  /*  GPIO Status (Read = Pin state;                 */
                             /*  Write: 0 = clear sticky status, 1 = no effect)  */
#define   GPIOCTL      0x1A /*  GPIO Control(w), Init(r) (Read = Power-on state; */
                             /*  Write : Set state of output pins)              */
#define   GPIOPUP      0x1C  /*  GPIO Pull-up Pull-up enable                    */
                             /*  (if input): 1 = enable, 0 = hi-Z               */
#define   GPIOPDN      0x1E  /*  GPIO Pull-down Pull-down enable                */
                             /*  (if input): 1 = enable, 0 = hiZ                */

/*PCI/USB Mailbox Registers (DSP IOPAGE=0x00)*/

#define    MBXSTAT       0x20     /*  Mailbox Status Mailbox Status              */
#define    MBXCTL        0x22     /*  Mailbox Control Mailbox Interrupt Control  */
#define    MBX_IN0       0x24     /*  Incoming Mailbox 0 PCI/USB to DSP mailbox  */
#define    MBX_IN1       0x26     /*  Incoming Mailbox 1 PCI/USB to DSP mailbox  */
#define    MBX_OUT0      0x28     /*  Outgoing Mailbox 0 DSP to PCI/USB mailbox  */
#define    MBX_OUT1      0x2A     /*  Outgoing Mailbox 0 DSP to PCI/USB mailbox  */

/*SERIAL EEPROM Control Register (DSP IOPAGE=0x00)*/

#define    SPROMCTL     0x30  /*  Serial EEPROM I/O Control/Status Direction and */
                                            /*  status  for SEN, SCK, SDA pins */


/* AC'97 Control Registers (DSP IOPAGE=0x00)*/

#define    AC97LCTL      0xC0     /*  AC'97 Link Control  */
#define    AC97LSTAT     0xC2     /*  AC'97 Link Status   */
#define    AC97SEN       0xC4     /*  AC'97 Slot Enable  */
#define    AC97SVAL      0xC6     /*  AC'97 Input Slot Valid  */
#define    AC97SREQ      0xC8     /*  AC'97 Slot Request  */
#define    AC97GPIO      0xCA     /*  AC'97 External GPIO Register  */

/* AC'97 External Codec IO Register Spaces */

#define    AC97CODEC0    0x400           /*  External Primary Codec 0 IO page    */
                                         /*  space registers (0x00 - 0x7F)       */
```

```
#define    AC97CODEC1    0x500          /*  External Secondary Codec 1 IO page  */
                                        /*  space registers (0x00 - 0x7F)       */
#define    AC97CODEC2    0x600          /*  External Secondary Codec 2 IO page  */
                                        /*  space registers (0x00 - 0x7F)       */

/* USB Endpoint DMA Control Registers (DSP IOPAGE=0x0C) */

#define   USB_EP4_ADDR     0x0C00    /*     Memory Buffer Base Addr. EP4  */
#define   USB_EP4_SIZE     0x0C04    /*     Memory Buffer Size EP4        */
#define   USB_EP4_RD       0x0C06    /*     Memory Buffer RD Offset EP4   */
#define   USB_EP4_WR       0x0C08    /*     Memory Buffer WR Offset EP4   */

#define   USB_EP5_ADDR      0x0C10    /*     Memory Buffer Base Addr. EP5   */
#define   USB_EP5_SIZE      0x0C14    /*     Memory Buffer Size EP5         */
#define   USB_EP5_RD        0x0C16    /*     Memory Buffer RD Offset EP5    */
#define   USB_EP5_WR        0x0C18    /*     Memory Buffer WR Offset EP5    */

#define   USB_EP6_ADDR      0x0C20    /*     Memory Buffer Base Addr. EP6   */
#define   USB_EP6_SIZE      0x0C24    /*     Memory Buffer Size EP6         */
#define   USB_EP6_RD        0x0C26    /*     Memory Buffer RD Offset EP6    */
#define   USB_EP6_WR        0x0C28    /*     Memory Buffer WR Offset EP6    */

#define   USB_EP7_ADDR      0x0C30    /*     Memory Buffer Base Addr. EP7    */
#define   USB_EP7_SIZE      0x0C34    /*     Memory Buffer Size EP7          */
#define   USB_EP7_RD        0x0C36    /*     Memory Buffer RD Offset EP7     */
#define   USB_EP7_WR        0x0C38    /*     Memory Buffer WR Offset EP7     */

#define   USB_EP8_ADDR      0x0C40    /*     Memory Buffer Base Addr. EP8    */
#define   USB_EP8_SIZE      0x0C44    /*     Memory Buffer Size EP8          */
#define   USB_EP8_RD        0x0C46    /*     Memory Buffer RD Offset EP8     */
#define   USB_EP8_WR        0x0C48    /*     Memory Buffer WR Offset EP8     */

#define   USB_EP9_ADDR      0x0C50    /*     Memory Buffer Base Addr. EP9    */
#define   USB_EP9_SIZE      0x0C54    /*     Memory Buffer Size EP9          */
#define   USB_EP9_RD        0x0C56    /*     Memory Buffer RD Offset EP9     */
#define   USB_EP9_WR        0x0C58    /*     Memory Buffer WR Offset EP9     */

#define   USB_EP10_ADDR      0x0C60    /*     Memory Buffer Base Addr. EP10   */
#define   USB_EP10_SIZE      0x0C64    /*     Memory Buffer Size EP10         */
#define   USB_EP10_RD        0x0C66    /*     Memory Buffer RD Offset EP10    */
#define   USB_EP10_WR        0x0C68    /*     Memory Buffer WR Offset EP10    */

#define   USB_EP11_ADDR     0x0C70     /*     Memory Buffer Base Addr. EP11   */
#define   USB_EP11_SIZE     0x0C74     /*     Memory Buffer Size EP11         */
#define   USB_EP11_RD       0x0C76     /*     Memory Buffer RD Offset EP11    */
#define   USB_EP11_WR       0x0C78     /*     Memory Buffer WR Offset EP11    */


#define   USB_DescriptVendorID           0x0C80  /* USB Descriptor Vendor ID       */
#define   USB_DescriptProductID          0x0C84  /* USB Descriptor Product ID      */
#define   USB_DescriptReleaseNumber      0x0C86  /* USB Descriptor Release Number   */
#define   USB_DescriptDeviceAttributes    0x0C88  /* USB Descriptor Device Attrib    */

/*  USB MCU Register Definitions */

#define   USB_MCU_SetupTokenCmd   0x0000     /*  USB SETUP Token Cmd 8 bytestotal */
#define   USB_MCU_SetupTokenData  0x0008   /*  USB SETUP Token Data 8 bytes total */
#define   USB_MCU_SetupCounter    0x0010     /*  USB SETUP Counter 16 bit counter */
```

```
#define   USB_MCU_CtlMiscCtl      0x0012  /* USB Control Misc including re-attach */
#define   USB_MCU_EndPointAddress 0x0014        /*  USB Address/Endpt Address of  */
                                                            /* device/active endpt */
#define   USB_MCU_FrameNumber       0x0016   /* USB Frame Num. Current frame num. */
#define   USB_MCU_EP4_DescriptConfig0x1000  /* USB EP4 Description Configs endpt */
#define   USB_MCU_EP4_NAK_Counter     0x1002              /*  USB EP4 NAK Counter */
#define   USB_MCU_EP5_DescriptConfig  0x1004 /* USB EP5 Description Configs endpt */
#define   USB_MCU_EP5_NAK_Counter     0x1006              /*  USB EP5 NAK Counter */
#define   USB_MCU_EP6_DescriptConfig  0x1008  * USB EP6 Description Configs endpt */
#define   USB_MCU_EP6_NAK_Counter     0x100A              /*  USB EP6 NAK Counter */
#define   USB_MCU_EP7_DescriptConfig0x100C  /* USB EP7 Description Configs endpt */
#define   USB_MCU_EP7_NAK_Counter     0x100E              /*  USB EP7 NAK Counter */
#define   USB_MCU_EP8_DescriptConfig0x1010/* USB EP8 Description Configs endpt */
#define   USB_MCU_EP8_NAK_Counter     0x1012              /*  USB EP8 NAK Counter */
#define   USB_MCU_EP9_DescriptConfig0x1014/* USB EP8 Description Configs endpt */
#define   USB_MCU_EP9_NAK_Counter     0x1016              /*  USB EP9 NAK Counter */
#define   USB_MCU_EP10_DescriptConfig 0x1018/* USB EP10 Description Configs endpt */
#define   USB_MCU_EP10_NAK_Counter    0x101A             /*  USB EP10 NAK Counter */
#define   USB_MCU_EP11_DescriptConfig0x101C/* USB EP11 Description Configs endpt */
#define   USB_MCU_EP11_NAK_Counter    0x101E             /*  USB EP11 NAK Counter */
#define   USB_MCU_StallPolicy         0x1020              /*  USB EP STALL Policy */

#define   USB_MCU_EP1_DownloadStartAddress  0x1040
/*  USB EP1 Code Download Start address for code download Base Address on endpt 1 */
#define   USB_MCU_EP2_DownloadStartAddress  0x1044
/*  USB EP2 Code Download Start address for code download Base Address on endpt 2 */
#define   USB_MCU_EP3_DownloadStartAddress  0x1048
/*  USB EP3 Code Download Start address for code download Base Address on endpt 3 */
#define   USB_MCU_EP1_CurrentWrPtrOffset    0x1060
       /* USB EP1 Current Write Pointer Offset  */
#define   USB_MCU_EP2_CurrentWrPtrOffset    0x1064
       /*  USB EP2 Current Write Pointer Offset */
#define   USB_MCU_EP3_CurrentWrPtrOffset    0x1068
       /*  USB EP3 Current Write Pointer Offset */
#define   USB_MCU_IO_RegisterAddress        0x2000  /*  USB Register I/O Address */
#define   USB_MCU_IO_RegisterData           0x2002     /*  USB Register I/O Data */
#define   USB_MCU_ProgramMemory             0x3000       /*  USB MCU Program Mem */


#endif
```

9 AC'97 CODEC PORT

Overview

AC'97 is a digital interface for the transport of audio and modem samples that was originally developed by Analog Devices, Creative Labs, Intel, National Semiconductor, and Yamaha and documented in the AC'97 specification. For ADSP-2192, the AC'97 specification provides a high audio architecture for the 1997 and 1998 volume platform segments.

The AC'97 interface, which complies with the AC'97 specification, connects the host's Digital Controller (DC) chip set and one to four analog audio (AC), modem (MC), or Audio/Modem (AMC) codecs.

ADSP-2192 Features and Functionality

The AC'97 interface has the following features and functionality:

- Each DSP core within the ADSP-2192 has four FIFOs, which provide data communication paths to the remainder of the chip
- `TX0`, `RX0`, `TX1`, and `RX1` are the FIFO registers in the universal register map of the DSP

- Each FIFO is eight words deep and 16 bits wide:
  - Two FIFOs (RX0 and RX1) are inputs that receive data and send it to the DSP core
  - Two FIFOs (TX0 and TX1) are outputs that send data from the DSP to the AC’97 interface

The AC’97 interface read data is transmitted in a format of 20 bits per slot; however, the ADSP-2192 stores data in a 16-bit format. On slots 2 through 11, the AC’97 reads the 16 MSBs of the data and ignores the 4 LSBs.

- Interrupts can be generated when some number of words have been received in the receive FIFOs or when some number of words are empty in the transmit FIFOs
- FIFOs 0(TX0 and RX0) and 1 (TX1 and RX1) in each DSP core can be used to send and receive data to the AC'97 interface of the ADSP-2192
- Each FIFO has a 16-bit control register (STCTL0/1 and SRCTL0/1) associated with it:
  - STCTL0/1 are the transmit FIFO control and status registers
  - SRCTL0/1 are the receive FIFO control and status registers

Table 9-1. FIFO Receive and Control Status Registers

Address	Register
0x10	STCTL0
0x20	STCTL1
0x11	SRCTL0

Table 9-1. FIFO Receive and Control Status Registers

Address	Register
0x21	SRCTL1
0x12	TX0
0x22	TX1
0x13	RX0
0x23	RX1

FIFO Control and Status Register

FIFO Transmit Control and Status Register

All bits in this register are reset to zero.

The following are the bit descriptions for the STCTL0/1 register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TU	TFE	TFF	Reserved	DME	FIP[2:0]			SLOT[3:0]				SMSEL	Reserved	CE[1:0]	

Table 9-2. STCTL0/1Register Bit Description

Bit Position	Bit Name	Description
01:0	CE[1:0]	Connection Enable 00 = Disable 01 = Reserved 10 = Connect to AC'97 11 = Reserved
2	Reserved	
3	SMSEL	Stereo / Monaural Select - AC'97 Mode. 0 = Monaural Stream 1 = Stereo Stream
7:4	SLOT[3:0]	AC'97 Slot Select - AC'97 Mode Monaural Stereo 0000 -> 0010 = Reserved 0011 = Slot 3 Slots 3/4 0100 = Slot 4 Slots 4/5 0101 = Slot 5 Slots 5/6 0110 = Slot 6 Slots 6/7 0111 = Slot 7 Slots 7/8 1000 = Slot 8 Slots 8/9 1001 = Slot 9 Slots 9/10 1010 = Slot 10 Slots 10/11 1011 = Slot 11 Slots 11/12 1100 = Slot 12 Not Allowed 1101 -> 1111 = Reserved
10:8	FIP[2:0]	FIFO Interrupt Position. An interrupt is generated when FIP[2:0] +1 words are empty in the FIFO. The Interrupt is Level Sensitive.
11	DME	DMA Enable. 0 = DMA Disabled 1 = DMA Enabled

Table 9-2. STCTL0/1Register Bit Description (Continued)

Bit Position	Bit Name	Description
12	Reserved	
13	TFF	Transmit FIFO Full - Read Only. 0 = FIFO Not Full 1 = FIFO Full
14	TFE	Transmit FIFO Empty - Read Only. 0 = FIFO Not Empty 1 = FIFO Empty
15	TU	Transmit Underflow - Sticky, Write "1" Clear. 0 = FIFO Underflow has not occurred 1 = FIFO Underflow has occurred

FIFO Receive Control and Status Register

When communicating with AC'97 interface, the connection enable bits in the control register are bits 1-0. Bit 3 selects stereo or monaural transfers to and from the AC'97 interface. Bits 7-4 select the AC'97 slot associated with this FIFO.

When stereo is selected, the slot identified and the next slot will be associated with the FIFO. Typically, stereo is selected for left and right data. Both left and right must be associated with the same external AC'97 codec. It is important that these sample rates be locked together. In this case, left and right data will alternate in the FIFO with the left data coming first.

If FIFO is enabled and a valid request for data comes that the FIFO cannot fulfill, the transmitter underflow bit will be set. This indicates that an invalid value was sent over the selected slot. Similarly, on the receive side, if the FIFO is full and another valid word is received, the Overflow bit will be set to indicate the loss of data.

All bits in this register are reset to zero.

The following are the bit descriptions for the SRCTL0/1 register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RO	RFE	RFF	Reserved	DME	FIP[2:0]			SLOT[3:0]				SMSEL	Reserved	CE[1:0]	

Table 9-3. SRCTL0/1 Register Bit Descriptions

Bit Position	Bit Name	Description
01:0	CE[1:0]	Connection Enable. 00 = Disable 01 = Reserved 10 = Connect to AC'97 11 = Reserved
2	Reserved	
3	SMSel	Stereo / Monaural Select - AC'97 Mode Only. 0 = Monaural Stream 1 = Stereo Stream

Table 9-3. SRCTL0/1 Register Bit Descriptions (Continued)

Bit Position	Bit Name	Description
7:4	SLOT[3:0]	AC'97 Slot Select - AC'97 Mode Only. Monaural / Stereo 0000 -> 0010 = Reserved 0011 = Slot 3 / Slots 3/4 0100 = Slot 4 / Slots 4/5 0101 = Slot 5 / Slots 5/6 0110 = Slot 6 / Slots 6/7 0111 = Slot 7 / Slots 7/8 1000 = Slot 8 / Slots 8/9 1001 = Slot 9 / Slots 9/10 1010 = Slot 10 / Slots 10/11 1011 = Slot 11 / Slots 11/12 1100 = Slot 12 / Not Allowed 1101 -> 1111 = Reserved
10:8	FIP[2:0]	FIFO Interrupt Position. An interrupt is generated when FIP[2:0] + 1 words have been Received in the FIFO. The interrupt is level sensitive.
11	DME	DMA Enable. 0 = DMA Disabled 1 = DMA Enabled
12	Reserved	
13	RFF	Receive FIFO Empty - Read Only. 0 = FIFO Not Empty 1 = FIFO Empty
14	RFE	Receive FIFO Empty - Read Only. 0 = FIFO Not Empty 1 = FIFO Empty

Table 9-3. SRCTL0/1 Register Bit Descriptions (Continued)

Bit Position	Bit Name	Description
15	RO	Receive Overflow - Sticky, Write-One-Clear. 0 = FIFO Overflow has not occurred 1 = FIFO Overflow has occurred

FIFO DMA Address Registers

```
0x48    TX0ADDR
0x4C    RX0ADDR
0x50    TX1ADDR
0x54    RX1ADDR
```

This is the 16-bit register specifying the current address of the DMA channel. This will be the address used for the next DMA access. After each access, the address will be incremented. The register will be loaded from the channel's NextAddress register when the count for the channel reaches zero.

FIFO DMA Current Count Registers

```
0x4B    TX0CURCNT
0x4F    RX0CURCNT
0x53    TX1CURCNT
0x57    RX1CURCNT
```

This is the 16-bit register specifying the current count of the particular DMA channel. The count is decremented after each DMA transfer for that channel. When the count reaches zero, it is reloaded from the Count register, the Address register is reloaded from the NextAddress register, and an interrupt is generated.

FIFO DMA Count Registers

```
0x4A    TX0CNT
0x4E    RX0CNT
0x52    TX1CNT
0x56    RX1CNT
```

This 16-bit register specifies the number of DMA transfers for a channel between interrupts and reloading of the address and current count registers. Count is loaded into Current Count when Current Count reaches zero.

FIFO DMA Next Address Registers

```
0x49  TX0NXTADDR
0x4D  RX0NXTADDR
0x51  TX1NXTADDR
0x55  RX1NXTADDR
```

This is the 16-bit register specifying the next start address to be loaded into the Address register at the end of the current buffer.

```
0x12  TX0
0x22  TX1
```

16-bit Transmit Data Register

These are the destination registers when not using DMA to load data into transmit FIFO. TX0 and TX1 are also in the directly addressable register map of the DSP core.

```
0x13  RX0
0x23  RX1
```

16-bit Receive Data Register

These are the source registers when not using DMA to unload data from receive FIFO. RX0 and RX1 are also in the directly addressable register map of the DSP core.

AC-Link Digital Serial Interface Protocol

AC'97 incorporates a 5-pin digital serial interface that links it to the AC'97 Controller. AC-link is a bidirectional, variable rate, serial PCM digital stream. It handles multiple input and output audio streams, as well as control register accesses employing a time division multiplexed (TDM) scheme.

The AC-link architecture divides each audio frame into 12 outgoing and 12 incoming data streams, each with 20-bit sample resolution. With a minimum required DAC and Analog/Digital Converter (ADC) resolution of 16-bits, AC'97 could also be implemented with 18 or 20-bit DAC/ADC resolution, given the space that the AC-link architecture provides. The protocol has the following characteristics:

- Each core has two independent connections to the synchronous AC'97 (Revision 2.1) serial interface that supports external modem, handset, and audio peripherals.
- The AC-link implements a bi-directional, variable rate, serial pulse code modulated (PCM) digital stream.
- Handle multiple input and output audio streams as well as control and status registers accesses using a time division-multiplexing (TDM) scheme, as illustrated in Figure 9-1.

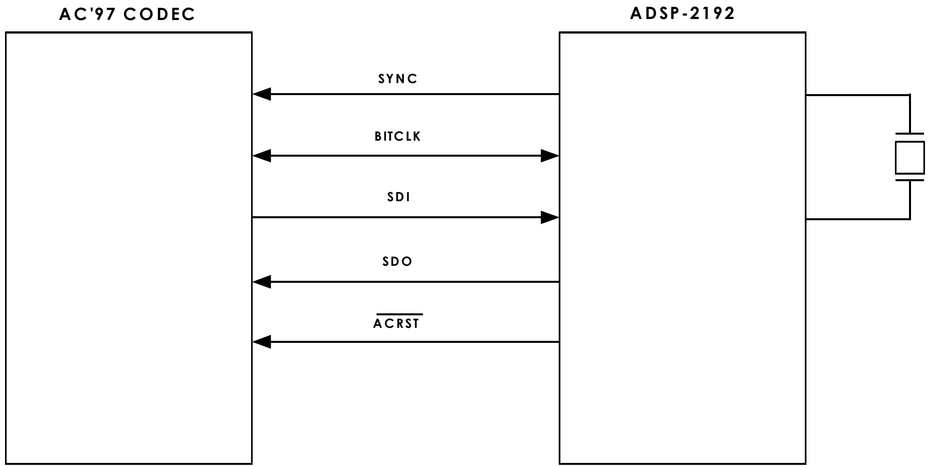


Figure 9-1. Codec to ADSP-2192 Communication

- The beginning of all audio sample packets, or Audio Frames, transferred over AC-link is synchronized to the rising edge of the `SYNC` signal. `SYNC` is driven by the ADSP-2192.
- `SYNC`, fixed at 48 kHz, is derived by dividing down the serial bit clock (`BITCLK`). `BITCLK`, fixed at 12.288 MHz, providing the necessary clocking granularity to support twelve 20-bit outgoing and incoming time slots and one 16-bit slot.
- AC-link serial data is transitioned on each rising edge of `BITCLK`. The receiver of AC-link data, AC'97 for outgoing data and ADSP-2192 for incoming data, samples each serial bit on the falling edges of `BITCLK`.

Resetting the AC'97

There are three types of AC'97 reset:

1. A cold reset where all AC'97 logic (registers included) is initialized to its default state.
2. A warm reset where the contents of the AC'97 register set are left unaltered.
3. A register reset which only initializes the AC'97 registers to their default states.

After signaling a reset to AC'97, the ADSP-2192 AC'97 controller will not attempt to play or capture audio data until it has sampled a "Codec Ready" indication from AC'97 (the AC'97 controller polls the `ACR` bit of the `AC97STAT` register).

ADSP-2192 AC'97 Control Registers

The ADSP-2192 contains an AC '97 Digital Controller that consists of dedicated hardware and customer provided DSP software. The ADSP-2192 can communicate with one to three codecs using one to eight sample transport channels (four separate monaural or stereo channels).

At power-on reset, the AC'97 Link is stopped with ACRST# asserted. When any other type of reset occurs, it may interrupt a running AC '97 link. In general, the hardware will not reset the link and the ADSP-2192 ROM code makes no attempt to return the AC '97 link to a known state. The host must download and run DSP code at reset time to stop (if desired) and restart the AC '97 link.

Refer to the Audio Hardware Developer section of Intel's web site (www.intel.com).

Table 9-4. AC'97 Control Registers

PCI Address	USB Address	DSP IO Page	DSP IO Address	Register Name	Description.
0x0C1-0x0C0	0x00C1-0x00C0	0x00	0xC0	AC97LCTL	Setup control for AC'97 interface. For more information, see "AC'97 Link Control/Status Register (AC97LCTL)" on page 9-15.
0x0C3-0x0C2	0x00C3-0x00C2	0x00	0xC2	AC97STAT	Setup control for AC'97 interface. For more information, see "AC'97 Link Status Register (AC97STAT)" on page 9-19.

Table 9-4. AC'97 Control Registers (Continued)

PCI Address	USB Address	DSP IO Page	DSP IO Address	Register Name	Description.
0x0C5-0x0C4	0x00C5-0x00C4	0x00	0xC4	AC97SEN	Setup control for AC'97 interface For more information, see "AC'97 Slot Enable Register (AC97SEN)" on page 9-21.
0x0C7-0x0C6	0x00C7-0x00C6	0x00	0xC6	AC97SVAL	Current status of valid frame from AC'97 link For more information, see "AC'97 Input Slot Valid Register (AC97SVAL)" on page 9-22.
0x0C9-0x0C8	0x00C9-0x00C8	0x00	0xC8	AC97SREQ	Current status of AC'97 slot requests For more information, see "AC'97 Slot Request Register (AC97SREQ)" on page 9-24.
0x0CB-0x0CA	0x00CB-0x00CA	0x00	0xCA	AC97SIF	GPIO slot 12 interface register For more information, see "AC'97 GPIO Status Register (AC97SIF)" on page 9-24.

AC'97 Link Control/Status Register (AC97LCTL)

The following are bit descriptions for the AC97LCTL register.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					ARPD	AGPE	ACWE	LKEN	BCEN	BCOE	Reserved	AFD	AFS	AFR	SYEN

Table 9-5. AC97LCTL Register Bit Definitions

Bit	Name	Description
0	SYEN	AC'97 Sync Generator Enable. This bit is automatically set to "1" upon link wakeup when enabled by the AC97LCTL:ACWE bit, resulting in immediate SYNC pulse generation upon resumption of bit clock. 1= Generate SYNC pulses and send and receive data over the AC'97 Link. 0= SYNC pulses are not generated. (default)
1	AFR	AC'97 Force Reset. This bit must remain set during the host computer boot sequence so that any attached audio codecs will enable the passive pass-through of PC_BEEP for audible POST error codes. The DSP must reset this bit before using the AC'97 Link the first time. To manually cold reset the AC'97 link, the DSP must write this bit to a 1 and then write it back to 0 after at least 1 us (or about 148 DSP clocks). BITCLK starts (BCEN is set) automatically on the Rising edge of ACRST# if BCOE = "1". 1= Forces ACRST# pin to "0". (default) 0= Releases ACRST#.

Table 9-5. AC97LCTL Register Bit Definitions (Continued)

Bit	Name	Description
2	AFS	AC'97 Force Sync. To manually warm reset the AC'97 link, the DSP must write this bit to a 1 and then write it back to 0 after at least 1 us (or about 148 DSP clocks). 1= Forces SYNC pin to "1" (allows AC'97 Vendor Test Mode). 0= SYNC pin is high only during Slot 0. (default)
3	AFD	AC'97 Force Data. 1= Forces SDO pin to "1" (allows AC'97 ATE Test Mode. 0= SDO pin drives serial data out. (default)
4	Reserved	Default value is 0.
5	BCOE	Bit Clock Output Enable. 1= BITCLK pin is an output, driving out the internally generated bit clock. 0= BITCLK pin is an input, taking in bit clock from the primary codec. (default)
6	BCEN	Bit Clock Generate Enable. 1= Internally generate a 12.288MHz bit clock from XTALI. Writing to "1" takes effect immediately. 0= Internally generated bit clock is "0" (default). Wait for two Frame Interrupts before writing to "0". The "0" then takes effect on the next External Codec Write.

Table 9-5. AC97LCTL Register Bit Definitions (Continued)

Bit	Name	Description
7	LKEN	AC'97 Link Enable. 1= When written from 0 to 1, bring the AC'97 link to a running state (see text). This will set the BCEN or SYEN bits and will clear the AFR bit. Writing to "1" takes effect immediately. 0= When written from 1 to 0, prepares to halt the AC'97 link to a "Warm" Reset State. The effect is delayed until the end of Slot 2 of the next AC'97 control register write (which must write either the PR4 or the MLNK bit to the primary codec). When that occurs, the sync generator is stopped (SYEN cleared) and, if applicable, the BITCLK generator is halted (BCEN cleared). Wait for two Frame Interrupts before writing to "0". The "0" then takes effect on the next External Codec Write.
8	ACWE	AC'97 Link Wakeup Enable. 1= Enable automatic restart of a powered-down AC'97 link on an AC'97 wake event (SDI=1). AC97LCTL:LKEN is set when a Wake event is detected, which will in turn de-assert ACRST#, set SYEN, etc. as needed. 0= The AC'97 link is not powered up in hardware on an AC'97 wake event. (default)

Table 9-5. AC97LCTL Register Bit Definitions (Continued)

Bit	Name	Description
9	AGPE	AC'97 GPIO Enable. 1= Sets slot 12 out valid and enables sending pin state data to the AC'97 GPIO pins using slot 12 (must write to the AC97SIF register first). Once enabled, GPIO Data is sent on every frame. To save power, enable AGPE, write AC97SIF, wait a frame and disable AGPE. 0= Slot 12 out is disabled. (default)
10	ARPD	AC Link Reset upon DSP Powerdown Enable. 1= Assert ACRST# when both DSPs are powered down. Do not set this Bit until the Link is in "Warm" Reset State, i.e. BITCLK is Stopped. 0= Do not automatically assert ACRST#. (default)

AC’97 Link Status Register (AC97STAT)

Table 9-6. AC’97 Link Status Register Bit Definitions

Bit	Name	Description
2:0	VGS[3:1]	Vendor-defined GPIO Status. Reports the state of the Vendor Optional GPIO bits (SDI Slot 12, bits 3-1) from the previous frame. Reflects data from all three SDIs ORed together.
5:3	AGI[2:0]	AC’97 Interrupt / Wakeup Detected. Reports “1” when the codec attached to the corresponding SDI[2:0] pin has signaled a wakeup or interrupt event. When the AC’97 Link is running, reports the status of the GPIO_INT bit (SDI Slot 12, Bit 0). When the link is stopped, the AGI bit is set immediately by an asynchronous HIGH state on the corresponding SDI pin. Switching between the two forms of reporting is automatic, based on the state of the AC’97 link. AGI<n> is valid only when either ACR<2:0> = 000 or when ACR<n> = 1. Between the time when the first codec reports Ready (after the Link starts) and when a given codec reports Ready, that codec's AGI bit may not be valid. Regardless, the AC'97 specification requires you to wait for a codec to report Ready before attempting to access its registers.
6	BCOK	AC’97 BITCLK OK. Reports “1” when the bit clock is running for both internally and externally generated bit clocks. Reverts to “0” within two bit clock periods after the clock stops.
7	LKOK	AC’97 Link OK. Reports “1” when the AC’97 Link is running. Reports “0” when either ACRST# is asserted, BITCLK is stopped or SYNCs are not Enabled.

Table 9-6. AC'97 Link Status Register Bit Definitions (Continued)

Bit	Name	Description
8	SYNC	AC'97 SYNC Status. Reports the current state of the AC'97 SYNC Signal, asserted during Slot 0.
9	REG	AC'97 Register Status. Use of this bit allows a DSP to efficiently time its accesses to controller and Codec registers. REG is asserted for exactly 20 BITCLKs, starting early in Slot 12 and ending early in Slot 0 of the next frame (after the rising edge of SYNC). The rising edge of REG occurs at the same point in time as the assertion of the AC'97 Frame Interrupt. Between then and the start of the next AC'97 Frame, there is enough time for at least four PDC transactions. This allows a DSP to examine some ADSP-2192 registers and then post an AC'97 Codec Register read or write and have it go out in the next Frame. REG is high during the time when AC'97 controller status registers update their values. You can inspect AC97STAT, AC97SVAL, AC97SREQ and AC97SIF anytime REG is low and get a synchronous snapshot of the previous Frame. A 1->0 transition on REG indicates that fresh status information is now available. For additional information, refer to "AC'97 AC97STAT:REG and Frame Interrupt Timing" on page 9-22.
12:10	Reserved	
15-13	ACR[2:0]	AC'97 Codec Ready. Reports "1" when the codec attached to the corresponding SDI[2:0] pin has set its Codec Ready bit (SDI Slot 0, Bit 15) in the previous frame. Always reports "0" when the AC'97 Link is disabled.

AC'97 Slot Enable Register (AC97SEN)

Table 9-7. AC'97 Slot Enable Register Bit Definitions

Bit	Name	Description
2:0	Reserved	
12:3	ACSE	Each bit enables the corresponding AC'97 slot to handle data. Effectively, by setting the bit, the DSP FIFO(s) commit to take RX data in every selected frame slot with an RX slot valid bit asserted, and they commit to transmitting data in every frame slot with a DAC request bit asserted. Overruns and underruns are possible but must be detected and tolerated by the FIFOs. The controller needs the ASCE register to process DAC request bits into TX Slot valid bits (like the RQE[1:0] bits) and to know when to generate internal SPORT framing signals as well. Setting ACSE[12] is supported only when AC97LCTL: AGPE=0. Each ACSE bit enables sample transfers in both directions. No independent control of transmit (SDO) and receive (SDI) is possible.
15:13	Reserved	

AC'97 Input Slot Valid Register (AC97SVAL)

The `AC97SVAL` and `AC97SREQ` registers are provided for diagnostic and debugging purposes only. All necessary processing of Slot Valid In/Out bits and Slot Request bits occurs automatically in dedicated hardware.

Table 9-8. AC'97 Slot Enable Register Bit Definitions

Bit	Name	Description
2:0	Reserved	
14:3	ACSV [1:12]	Each bit reports the state of the corresponding slot valid bit from the previous frame (SDI Slot 0 data).
15	ACR	AC'97 Codec Ready. Reports "1" if any codec asserted its Codec Ready bit in the previous frame.

AC'97 AC97STAT:REG and Frame Interrupt Timing

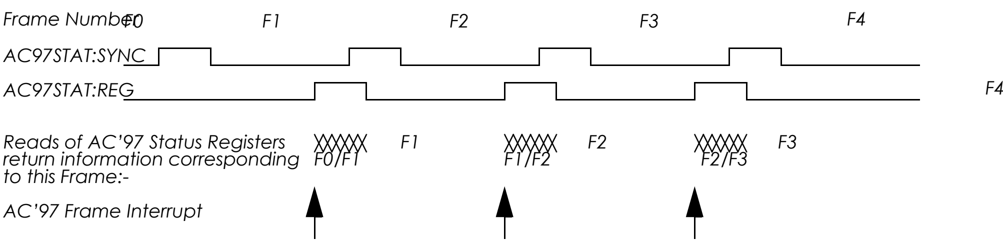


Figure 9-2. ASTST:REG and Frame Interrupt Timing

For low latency, the recommended method for code to determine if an AC'97 Frame Interrupt has occurred is for the Interrupt Service Routine to write a flag in memory. The alternative approach involves detecting a rising edge on AC97STAT:REG via PDC bus reads. A drawback to using this method is that it would take longer.

AC'97 External Codec Register Spaces

Table 9-9. AC'97 External Codec Register Spaces

IO Page	Address Range	Name
0x04	0-0x7E	AC97 External Primary Codec 00 Registers
0x05	0-0x7E	AC97 External Secondary Codec 01 Registers
0x06	0-0x7E	AC97 External Secondary Codec 10 Registers
0x07	0-0x7E	AC97 External Secondary Codec 11 Registers

Each external codec has a control/status register space, specified in the AC'97 / AC'97 2.1 Codec Specification. In the ADSP-2192, these register spaces are mapped into the DSP IO space. Each codec is placed in a separate Page in I/O space. This permits code that supports more than one codec to be shared, requiring simply a change of the IOPG register to select the codec desired.

AC’97 Slot Request Register (AC97SREQ)

The AC97SVAL and AC97SREQ registers are provided for diagnostic and debugging purposes only. All necessary processing of Slot Valid In/Out bits and Slot Request bits occurs automatically in dedicated hardware.

Table 9-10. AC97SREQ Register Bit Definitions

Bit	Name	Description
2:0	Reserved	
12:3	ACRQ [3:12]	Each bit reports the state of the corresponding slot request bit from the previous frame (SDI Slot 2 data). Bits are active low when the corresponding DAC is enabled and always low when the corresponding DAC is disabled.
15:13	Reserved	

AC’97 GPIO Status Register (AC97SIF)

Table 9-11. AC’97 GPIO Control / Status Register Bit Definitions

Bit	Name	Description
15-0	AGS [15:0]	**Reads.** Reports the state of the corresponding AC’97 GPIO pin during the previous frame. **Writes.** The AC97SIF register is sampled at the beginning of Slot 12 to provide pin state data to AC’97 GPIO pins programmed as outputs (provided AC97LCTL:AGPE=1).

ADSP-2192 AC'97 Audio Interface

The ADSP-2192 offers several features to support PC Audio requirements. The ADSP-2192 has an AC'97 2.1-compliant interface that supports up to three external codecs. These can be any combination of Audio or Modem/Handset Codecs.

External Audio Codec (AC'97) Subsystem

Resource Allocation

System design involves allocating the following resources among the supported devices:

- **SDI pins.** There are three SDI (Serial Data In) pins on the ADSP-2192. One SDI pin must be connected to each added codec.
- **AC'97 sample stream slots.** There are ten bidirectional sample slots per AC'97 frame (slots 3 through 12, although 12 is almost always used for GPIO.) Monaural streams use one slot, while stereo streams take two adjacent slots. While different streams may have different, unsynchronized sample rates, the left and right streams in a stereo pair are locked together. (The AC'97 2.1 specification suggests certain slot assignments for various functions. While external codecs may require such specific slots, the ADSP-2192 AC'97/FIFO hardware is general and may be programmed to any slot in the range 3 to 12).

- **DSP DMA FIFOs.** There are four FIFOs, two on each DSP. Each is capable of handling one monaural or stereo sample stream when assigned to AC’97 sample slots. (Rx and Tx may be assigned to different slots.)

  This implies that a maximum of eight AC’97 sample slots may be operated at any time.

- **Computational resources** (MIPS and DSP Memory), as appropriate.

Table 9-12. AC’97 Pin Listing

Pin Name	Function	Description	Connections
ACRST#	O	AC'97 Audio Reset.	To all external codecs
SYNC	O	AC'97 Sync. 48 KHz frame rate	To all external codecs
SDO	O	AC'97 Serial Data Out.	To all external codecs
SDI[2:0]	I	AC'97 Serial Data In Pins.	One input from each external. codec
BITCLK	I/O Output enabled if AC97LCTL:BCOE=1	AC'97 Bit Clock. 12.288 MHz	Connects to all external codecs; may be driven by ADSP-2192 or by one external codec

For most purposes, the AC’97 protocol describes SDI as a single input stream rather than three distinct streams. This stream is derived by ORing (combining by using the logical OR function) all three SDI pins. Unused SDI pins must be tied to GND for proper operation of the other devices.

BITCLK: On the ADSP-2192, the BITCLK signal may be generated internally or externally, as selected by the BCOE bit of the AC97LCTL register. If an external codec is configured as an AC'97 Primary codec, it is always (by definition) the generator of BITCLK. If all the external codecs are configured as secondary devices, the ADSP-2192 can generate the BITCLK signal.

When powering down the link, ensure that an AC'97 control register write be performed to the PR4 or MLNK bits of the nonexistent primary codec. The write function causes the external secondary devices to search the link for this control register write in order to time their own transition into powerdown properly.

AC'97 2.1 Protocol Summary

This is an introductory summary of the AC'97 2.1 serial interface protocol. For complete information, see the Audio Codec '97 Specification, from Intel Corporation.

The AC'97 Frame is structured as follows. Each frame is made up of one 16-bit tag slot (slot 0) and twelve 20-bit data slots (1 through 12), as shown Figure 9-3, for a total of 256 bits. Slots are numbered in increasing order (0 first), while bits are numbered in decreasing order (MSB first) See also the ADSP-2192 AC'97 Interface Bitmap Table, for a cross-reference between the ADSP-2192 register bits and the AC'97 serial data stream.)

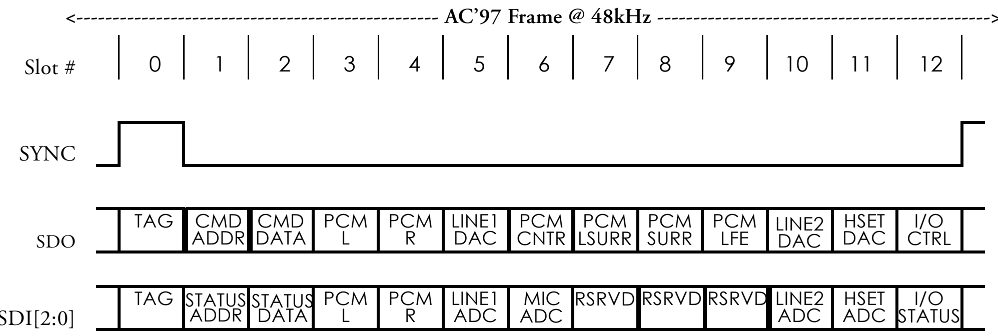


Figure 9-3. AC'97 frame structure

Access to AC'97 Codec Control/Status Registers

After a cold or warm restart of the AC'97 link, you must poll for the corresponding codec ready bit in AC97STAT to ensure that it has become asserted before attempting to access any AC'97 registers in that codec. A codec ready bit reads "1" only after the link begins to run (after `AC97STAT:LKOK` becomes "1") and after the corresponding codec asserts its Codec Ready bit in SDI Slot 0, Bit 15. Unused SDI pins should be tied low and their codec ready bits will always read "0".

Codec registers are memory-mapped into PDC bus address space. Each codec is assigned one IO page, so that Codec ID's 0-3 map to IO pages 4-7. All address offsets match the addresses in the AC'97 spec. For example, to write and read register `0x5E` on codec ID 1 from a DSP, do:

```
IOPG = 0x5;// CID 1
io(0x5E) = ...// AC97 Register 0x5E
...  = io(0x5E);
```

Note that, in addition to the DSPs, the PCI/USB/sub-ISA host may also directly access AC'97 Codec registers, to aid in debugging.

Reads and writes to AC’97 Codec Registers are automatically synchronized by hardware to place the proper information in Slots 0, 1 and 2 of the next available AC’97 Frame. Up to two AC'97 Codec Control/Status Register (CSR) writes may be posted and pending at any given time. If a third write is posted, PDC bus waitstates are inserted until the first can complete. Every CSR read inserts PDC waitstates until the end of Slot 2 in the Frame in which the read data is returned. When a DSP is waiting for completion of its PDC bus transaction, it is halted, unable to compute, DMA or interrupt. Excessive PDC waitstates should be avoided.

In order to minimize PDC waitstates, the recommended procedure is to time CSR accesses with the AC’97 Frame Interrupt. Wait for the interrupt and then post one read or write per interrupt. This gives the minimum latency until read data is returned. Writes may be posted with minimal PDC wait states at any time, but will take effect in the codec with minimum latency if posted following a Frame Interrupt. There is a DSP Flag In signal that tells whether a CSR access launched now will incur significant PDC waitstates.

Attempts to access an AC’97 Codec Register when the link is not running (when `AC97STAT:LKOK=0`) *are ignored*. Attempting to write an unpopulated Codec ID will send a write transaction over the AC’97 Link that is ignored by all present codecs. Attempting to read an unpopulated Codec ID will send a read request transaction over the AC’97 Link that is ignored by all present codecs. In the next Frame, the AC’97 controller will return all zeros to the PDC bus as long as no Codec sets Slot 2 Valid in.

There are three cases when you must synchronize state changes between the AC’97 controller and an AC’97 Codec:

- Enabling or disabling slots (writing the `AC97SEN` register)
- Disabling the link (writing `AC97LCTL:LKEN` to “0”)
- Disabling the locally generated `BITCLK` (writing `AC97LCTL:BCEN` to “0”)

Each case involves changing one controller register and one Codec register. The controller delays the effect in the controller register until the Codec receives its serial register write. Before launching the Codec register write, you must make sure that any previously posted Codec register writes have completed. Therefore, you must wait through two frames (two AC'97 Frame Interrupts) and then write the controller and Codec registers in close succession.

Changes to AC97SEN take effect at the next AC'97 Codec Control/Status Register (CSR) write. The value read back from AC97SEN is updated immediately after it is written by a DSP. This makes it possible, if timed properly, for DSP #1 and then DSP #2 to each read-modify-write AC97SEN to enable their respective allotted slots and have all the slot enable changes take effect in the same frame. Unfortunately, in the standard AC'97 protocol you can not enable all the codecs at the same time, as it takes multiple AC'97 writes-and therefore multiple Frames-to address them. It is possible to enable multiple codecs in one Frame if using ADI Chaining Mode.

AC'97 2.1 Link Powerdown States

As illustrated in Figure 9-4, the AC'97 2.1 interface may be powered down when inactive to save power, either to a Cold or Warm state. In the Cold state, ACRST# is asserted, and BITCLK and SYNC are halted. In the Warm state, BITCLK and SYNC are halted but ACRST# is deasserted.

Note: *On powerup, de-asserting ACRST# from the Cold state causes the BITCLK master to start, but the controller does not necessarily start generating SYNC pulses until it is enabled. This state in which BITCLK is running but SYNC is halted is called IDLE.*

When powered down, a wakeup protocol is defined using the SDI pins. A high level on SDI asynchronously signals a wakeup condition to the controller. If it is enabled to do so, the controller may then restart the link upon receiving this wakeup signal.

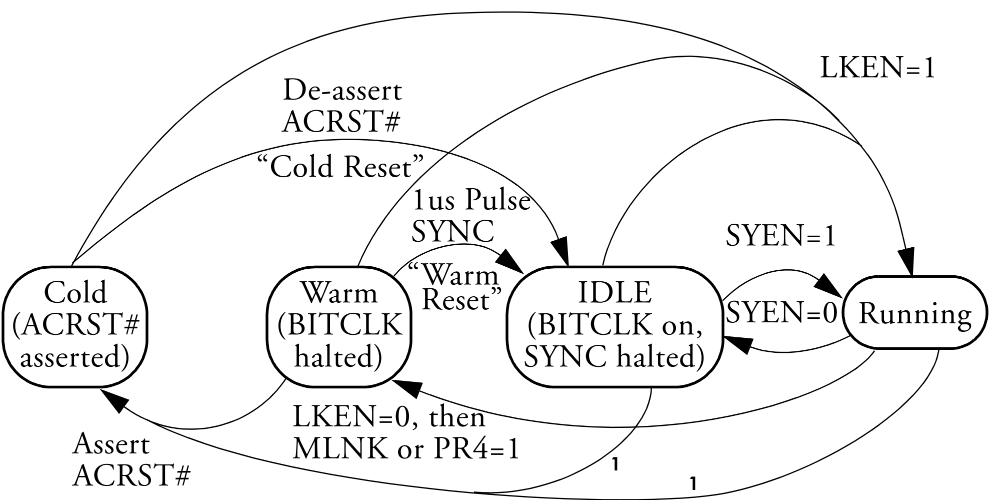
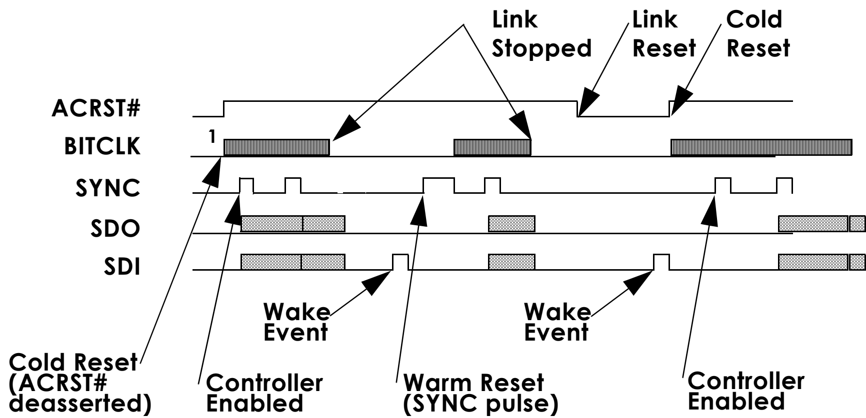


Figure 9-4. AC'97 Link Powerdown States, by Function



[1]BITCLK is stopped when ACRST#=0.

Figure 9-5. Link powerdown states, by signal

Important attributes of the powerdown states are as follows:

- AC'97 Frames, sample data, GPIO state, and register accesses can only be transferred in the Running state. The ADSP-2192 `AC97STAT:LKOK` (Link OK) bit reads 1 in this state only.
- The `AC97STAT:BCOK` (`BITCLK OK`) reads 1 in the Idle and Running states.
- Wakeups/Interrupts are signalled using SDI Slot 12 Bit 0 during the Running state.
- Wakeups are signalled using a high level on SDI during all other states.

State Transitions

The state transitions are: powerup and powerdown.

Power-Up Transitions

- The simplest way to power up the ADSP-2192 AC'97 link controller from any power-down state is to write AC97LCTL:LKEN=1, and then poll for AC97STAT:LKOK=1. The ADSP-2192 hardware will make the appropriate transitions to bring the interface to a running state.
- From the Cold state, de-assertion of ACRST# by the controller (for example, clearing AC97LCTL:AFR) causes the BITCLK generator to start, resulting in an IDLE state. Note that the BITCLK generator may be either in an external Primary codec, or the ADSP-2192's internal BITCLK generator, as selected by the AC97LCTL:BCOE bit.
- From Warm, assertion of SYNC for the bus by the controller causes the BITCLK to start (upon de-assertion of SYNC), resulting in the IDLE state.
- From IDLE, writing the bit AC97LCTL:SYEN=1 starts the SYNC pulse generator and places the link in a running state. Note that additional configuration is needed to power up devices, associate FIFOs with AC'97 Slots, and to enable sample transmission (see below).

The AC97LCTL:ACWE (AC'97 Wake Enable) bit enables automatically restarting the link (setting LKEN=1) when a wake event is detected in a non-Running (LKOK=0) state.

Power-Down Transitions

- The link must be powered down to a Warm state in this sequence:

  a. Power down all codec blocks by writing PRn bits to 1, except for the primary codec's PR4 or MLNK bit.

  b. Wait for two AC'97 Frame Interrupts.

  c. Write AC97LCTL:LKEN=0. This tells the controller that the link is about to be stopped.

  d. Write the primary codec's PR4 bit (if audio) or MLNK bit (if modem) to 1. At the end of slot 2 of this access, the primary codec will stop BITCLK; if BITCLK is internally generated (AC97LCTL:BCOE=1), the generator will also stop at this point (AC97LCTL:BCEN is cleared). External codecs will snoop the link watching for this register write, and will fully power down at this point. The controller's SYEN bit should now be cleared (it will be cleared automatically if the LKEN bit was used to power down the link).

- The link may then be brought from a Warm to a Cold state by asserting ACRST#, which is done by either writing AC97LCTL:AFR to 1, powering down the DSPs with the AC97LCTL:ARPD bit set to 1, or asserting the interface RST# pin while SCFG:RDIS is 0 (the power-on default).

  The recommended way to enable and disable the link is through use of the LKEN bit.

  When LKEN is written to 1 (from 0):

  - If the link was already running (LKOK=1), there is no effect.

  - If the link was in Cold reset, ACRST# is deasserted (clearing AC97LCTL:AFR if necessary).

- If the link was in Warm reset, SYNC is asserted for 1 us and then deasserted.
- If BITCLK is internal, the BITCLK generator starts (AC97LCTL:BCEN set to 1) when ACRST# deasserts (cold) or when SYNC deasserts (warm). If BITCLK is external, the external primary codec must start BITCLK in a similar manner.
- When BITCLK restarts (AC97STAT:BCOK reads 1), the sync pulse generator is automatically enabled (AC97LCTL:SYEN set to 1). At this point, LKOK reads 1.

When LKEN is written to 0 (from 1):

The controller will wait for the end of Slot 2 of the next control register write (which must set PR4 or MLNK) and then disable the BITCLK generator (clear AC97LCTL:BCEN) and SYNC generation (Clear AC97LCTL:SYEN). At this point, LKOK will read 0.

When LKEN is written with the same value (0==>0 or 1==>1), there is no effect. The safest way to restart the Link when it is in an unknown state (such as at a reset other than power on) is to write LKEN to a zero (0) and then to one (1).

Configuring AC'97 Sample Data Streams

DAC and ADC sample streams are conveyed to external AC'97 codec devices by assigning DSP FIFOs to AC'97 Data Slots.

In order to enable an AC'97 sample stream, follow this sequence:

1. Set up the DMA channel for the correct DSP FIFO.
2. Program the Transmit and/or Receive FIFO Control Register (STCTL/SRCTL) in the correct DSP FIFO. Program the desired slot number into the AC'97 Slot Select field, set the Stereo/Monaural select bit as required and set the Connection Enable field to 10 for AC'97. See the DSP FIFO section for more details.
3. Pre-fill the TX FIFO, if needed.
4. Enable RX/TX interrupts in the corresponding DSP, if needed.
5. Wait for two AC'97 Frame interrupts in order to flush out any pending CSR writes.
6. Write the appropriate AC'97 Codec Control/Status Register to enable the DAC/ADC.
7. Set the bit or adjacent pair of bits in the AC97SEN register corresponding to the slot(s) to be enabled.

In order to disable an AC'97 sample stream, follow this sequence:

1. Wait for two AC'97 Frame Interrupts in order to flush out any pending CSR writes.
2. Write the appropriate AC'97 Codec Control/Status Register to disable the DAC/ADC.
3. Clear the bit or adjacent pair of bits in the AC97SEN register corresponding to the slot(s) to be disabled.

4. Wait for one AC'97 Frame Interrupt in order to let the CSR write complete.
5. Disable RX/TX interrupts in the corresponding DSP, if needed.
6. Drain the RX FIFO, if needed.
7. Clear the Connection Enable bits in the Transmit and/or Receive FIFO Control Register (STCTL/SRCTL) in the correct DSP FIFO to 00.

10 JTAG TEST-EMULATION PORT

The ADSP-2192 contains a JTAG test access port. The emulator uses JTAG logic for ADSP-2192 communications and control. This JTAG logic consists of a state machine, a five pin Test Access Port (TAP), and shift registers. Note that the ADSP-2192 JTAG does not support boundary scan.

The TAP pins appear in Table 10-1.

Table 10-1. JTAG Test Access Port (TAP) Pins

Pin	Function
TCK	(input) Test Clock: pin used to clock the TAP state machine.[1]
TMS	(input) Test Mode Select: pin used to control the TAP state machine sequence.[1]
TDI	(input) Test Data In: serial shift data input pin.
TDO	(output) Test Data Out: serial shift data output pin.
$\overline{\text{TRST}}$	(input) Test Logic Reset: resets the TAP state machine

[1] Asynchronous with XTALI

For more information about JTAG, see application note EE-68. Engineering application notes are available at `www.analog.com`.

11 SYSTEM DESIGN

Overview

This chapter describes the basic system interface features of the ADSP-219x family processors, including the ADSP-2192. The system interface includes various hardware and software features used to control the DSP processor. Processor control pins include a $\overline{\text{PORST}}$ (power on reset) signal, clock signals, flag inputs and outputs, and interrupt requests. This chapter describes only the logical relationships of control signals; consult individual processor data sheets (including the data sheet for the ADSP-2192) for actual timing specifications.

Sources for Additional Information

Some ADSP-2192 system interfaces are documented in other chapters in this book, as follows:

- "Host (PCI/USB) Port" on page 8-1 discusses the PCI, CardBus, and Sub-ISA interfaces, and includes information about how these applications use the Serial EEPROM port and the DMA (Direct Memory Access) controller. That chapter also discusses the USB interface and includes information about how USB applications use the Serial EEPROM port and the DMA controller.
- "AC'97 Codec Port" on page 9-1 discusses the AC'97 interface.
- "JTAG Test-Emulation Port" on page 10-1 discusses the JTAG Test Access Port and how it is used during emulation.

In addition, the Analog Devices web site contains a series of engineering application notes. These documents describe topics related to the ADSP-2192, including:

- Hints for porting code from ADSP-218x chips to ADSP-219x chips (described in application note EE-121)
- Tips for maximizing performance on the ADSP-219x family of processors (described in application note EE-122)
- An overview of the ADSP-219x pipeline (described in application note EE-123),
- Mechanism for booting on the ADSP-2192 (described in application note EE-124).

Engineering application notes, including any new ones that may have been added since this manual was published, are available at `www.analog.com`.

Pin Descriptions

The ADSP-2192 processor comes in a 144-LQFP package configuration. This section provides functional descriptions of the ADSP-2192 pins.

Table 11-1 through Table 11-7 provide ADSP-2192 processor pin descriptions.

Table 11-1. PCI/USB Bus Interface Pin Descriptions

Pin Name(s)	Number of Pins	I/O	Description
AD0 - AD31	32	I/O	Address and Data Bus
CBE0 - CBE3	4	I/O	PCI Command/Byte Enable
CLK	1	I	PCI Clock
CLKRUN	1	O	Clock Run
DEVSEL	1	I/O	PCI Target Device Select
FRAME	1	I/O	PCI Frame Select
GNT	1	I	PCI Grant
IDSEL	1	I	PCI Initiator Device Select
INTAB	1	O	PCI / ISA Interrupt
IRDY	1	I/O	PCI Initiator Ready
PAR	1	I/O	PCI Bus Parity/
PCIGND	7	I	PCI Ground
PCIVDD	7	I	PCI VDD supply
PERR	1	I/O	PCI Parity Error/USB- (Inverting input)

Table 11-1. PCI/USB Bus Interface Pin Descriptions (Continued)

Pin Name(s)	Number of Pins	I/O	Description
$\overline{PME}$	1	O	PCI Power Management Event
$\overline{REQ}$	1	O	PCI Request
$\overline{RST}$	1	I	PCI Reset
$\overline{SERR}$	1	O	PCI System Error/USB+ (Non-inverting input)
$\overline{STOP}$	1	I/O	PCI Target Stop
$\overline{TRDY}$	1	I/O	PCI Target Ready

Table 11-2. Crystal/Configuration Pin Descriptions

Pin Name(s)	Number of Pins	I/O	Description
BUS0 - BUS1	2	I	PCI/ Sub-ISA/CardBus Select pins
CLKSEL	1	I/O	Clock Select
IGND	1	I	IGND
NC	1	O	No Connect
$\overline{PORST}$	1	I	Power On Reset
XTALI	1	I	Crystal Input Pin (24.576 MHz)
XTALO	1	O	Crystal Output Pin

Table 11-3. AC'97 Interface Pin Descriptions

Pin Name(s)	Number of Pins	I/O	Description
$\overline{\text{ACRST}}$	1	O	AC'97 Reset
ACVAUX	1	I	AC'97 Vaux Input
ACVDD	1	I	AC'97 VDD Input
BITCLK	1	O	AC'97 Bit Clock
SDI0 - SDI2	3	I	AC'97 Serial Data Input
SDO	1	O	AC'97 Serial Data Output
SYNC	1	O	AC'97 Sync

Table 11-4. Serial EEPROM Pin Descriptions

Pin Name(s)	Number of Pins	I/O	Description
SCK	1	I	Serial EEPROM Clock
SDA	1	I	Serial EEPROM Data
SEN	1	I	Serial EEPROM Enable

Table 11-5. Emulator Pin Descriptions

Pin Name(s)	Number of Pins	I/O	Description
$\overline{\text{EMU}}$	1	O	Emulator Event Pin
TCK	1	I	Emulator Clock Input
TDI	1	I	Emulator Data Input

Table 11-5. Emulator Pin Descriptions (Continued)

Pin Name(s)	Number of Pins	I/O	Description
TDO	1	O	Emulator Data Output
TMS	1	I	Emulator Mode Select
$\overline{\text{TRST}}$	1	I	Emulator Logic Reset

Table 11-6. I/O Pin Descriptions

Pin Name(s)	Number of Pins	I/O	Description
AIOGND	2		IO Ground
IO0 - IO7	8	I/O	IO Pin, Bits 0-7
IOVDD	2		IO Vdd

Table 11-7. Power Supply Pin Descriptions

Pin Name(s)	Number of Pins	I/O	Description
ACVAUX	1		AC'97 VAUX Input
AIOGND	1		IO Ground
AVDD	1		Analog VDD Supply
CTRLAUX	1		AUX Control
CTRLVDD	1		Control VDD
IGND	9		Digital Ground

Table 11-7. Power Supply Pin Descriptions (Continued)

Pin Name(s)	Number of Pins	I/O	Description
IVDD	9		Digital VDD
RVAUX	1		AUX supply
RVDD	1		Analog VDD Supply

Clock Signals

The ADSP-2192 can be clocked by a crystal oscillator. If a crystal oscillator is used, the crystal should be connected across the XTALI/O pins, with two capacitors connected as shown in Figure 11-1 on page 11-8. Capacitor values are dependent on crystal type and should be specified by the crystal manufacturer. A parallel-resonant, fundamental frequency, microprocessor-grade 24.576 MHz crystal should be used for this configuration.

The ADSP-2192 processor can also be operated with an external frequency generator or integrated oscillator supplying the clock signal. If an external clock is used, the input should be connected to the XTALI pin, and the XTALO pin must be left unconnected. If the input waveform exceeds a 2.5V signal level, the CLKSEL pin should be tied to VBYP (2.5V) to protect the internal oscillator. The XTALI signal may not be halted, changed, or operated below the specified frequency during normal operation.

The internal phase locked loop (PLL) of the processors generates an internal clock that is, by default, six times the input frequency. This clock rate is configurable, and the multiplier's default value of six can be changed, according to the bit settings of DPLLN, DPLLK, and DPLLM in the DSP's PLL control register.

For more information about the programmable PLL, see “Setting Dual DSP Core Features” on page 6-3.

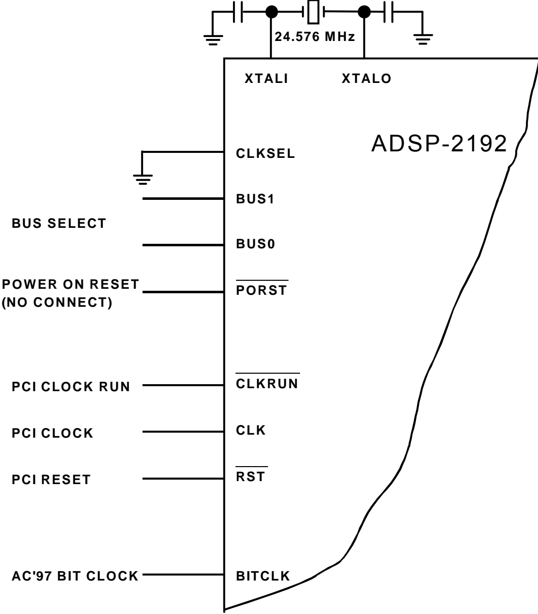


Figure 11-1. External Crystal Connections

The ADSP-2192 defines the following clock domains: PCI, USB, AC’97, DSP clock, and the Peripheral Device Control (PDC) bus. Figure 11-2 on page 11-9 shows these clock domains and their relationships to each other.

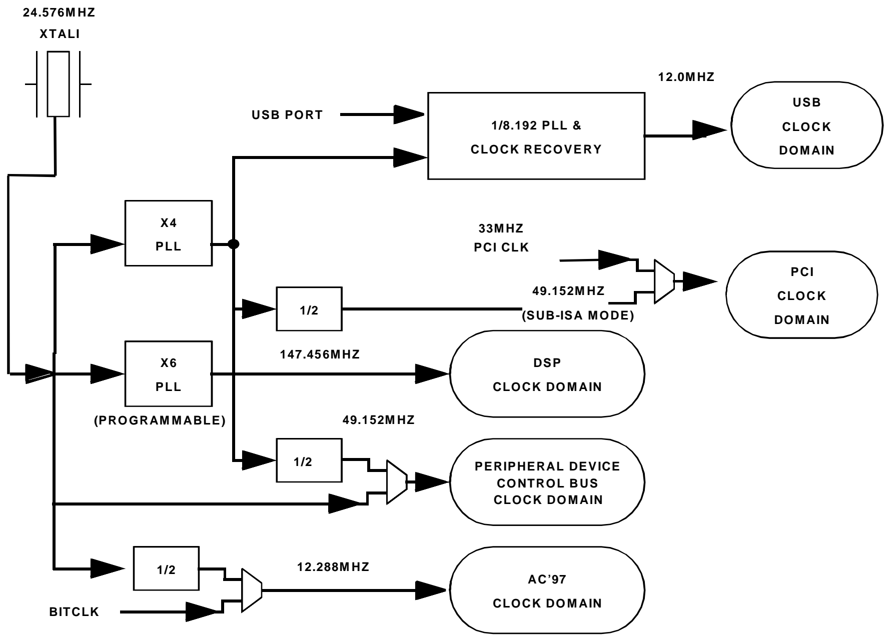


Figure 11-2. ADSP-2192 Clock Domains

Synchronization Delay

Each peripheral has several asynchronous inputs (interrupt requests, for example), which can be asserted in arbitrary phase to the processor clock. The processor synchronizes such signals before recognizing them. The delay associated with signal recognition is called *synchronization delay*.

Different asynchronous inputs are recognized at different points in the processor cycle. Any asynchronous input must be valid prior to the recognition point to be recognized in a particular cycle. If an input does not meet the setup time on a given cycle, it is recognized either in the current cycle or during the next cycle if it remains valid.

Edge-sensitive interrupt requests are latched internally so that the request signal only has to meet the pulse width requirement. To ensure the recognition of any asynchronous input, however, the input must be asserted for at least one full processor cycle plus setup and hold time. Setup and hold times are specified in the data sheet for the ADSP-2192 or data sheets for other products in the ADSP-219x family.

Configurable Clock Multiplier Considerations

The processors on the ADSP-2192 use an on-chip phase-locked loop (PLL) to generate the internal clock signals. Because these clocks are generated based on the rising edge of XTALI, there is no ambiguity about the phase relationship of two processors sharing the same input clock. Multiple processor synchronization is simplified as a result.

Using an input clock with more than one possible frequency (with the phase-locked loop generating the internal clock cycles, based on the configurable value of the clock multiplier) imposes certain restrictions. The XTALI signal must be valid long enough to achieve phase lock before $\overline{\text{PORST}}$ can be deasserted. Also, the clock input frequency cannot be changed unless the processor is in $\overline{\text{PORST}}$.

The PLLCTL register controls clock multiplier modes for the ADSP-2192. See “ADSP-2192 DSP Peripheral Registers” on page B-1 for more information about the bits in PLLCTL. These modes affect operations for both DSP cores. See “Clock Multiplier Mode Control” on page 6-10 for more instructions on how to change the clock multiplier.

Maximizing Performance of DSP Algorithms

When writing DSP algorithms, you can significantly improve performance by executing many instructions in parallel, or within the same clock cycle. This is called *parallel operation*, and the instructions that execute multiple operations within a single cycle are called *multifunction instructions*.

Multifunction operations available on the ADSP-2192 include:

- ALU/MAC with DM and PM dual read using DAGS 1and2 post-modify
- Multifunction ALU/MAC with DM/PM read or write using DAG post-modify
- ALU/MAC/Shift and any DREG-to-DREG transfer
- Conditional ALU/MAC/Shift
- Unconditional Register File ALU/MAC

Table 11-8 lists the registers available for multifunction instructions.

Table 11-8. Available Registers for Multifunction Instructions

Operation	Available XOPs	Available YOPs
ALU	AX0, AX1, AR, MR0, MR1, MR2, SR0, SR1	AY0, AY1, AF, the value zero
MAC	MX0, MX1, AR, MR0, MR1, MR2, SR0, SR1	MY0, MY1, SR1, the value zero
Shifts	SI, SR0, SR1, SR2, AR, AX0, AX1, AY0, AY1, MX0, MX1 MY0, MY1, MR0, MR1, MR2	N/A

For more information about these performance enhancements and instructions for optimizing the code in your DSP algorithms, see Application Note EE-122, available from the ADSP-2192 product page on the `www.analog.com` web site.

Resetting the Processor

The ADSP-2192 supports booting via either the PCI interface or the USB interface. The boot loader kernel, located in the DSP ROM, determines how the DSP boots (PCI or USB). The kernel then sets up and initializes appropriate DSP registers to facilitate the booting.

Three methods for resetting the processor on the ADSP-2192 are discussed in this section: Power On Reset, Forced Reset Via PCI/USB, and Software Reset. The reset type is specified by bits 8 and 9 (CRST<1:0>) of the Chip Mode/Status Register (CMSR), as follows:

- CRST<1:0>=00—Power On Reset
- CRST<1:0>=01—(Reserved)
- CRST<1:0>=10—PCI/USB Hard Reset
- CRST<1:0>=11—Soft Reset from CMSR RST bit.

For information about resetting the AC'97 link, refer to "Resetting the AC'97" on page 9-12.

Power On Reset

The ADSP-2192 has an internal Power On Reset circuit that resets the DSP when power is applied. A Power On Reset ($\overline{\text{PORST}}$) signal can also initiate this master reset. When the Power On Reset is invoked, program flow jumps to the first location of the loader kernel at address 0x14000 and begins execution.

Forced Reset Via PCI/USB

In addition to the Power On Reset ($\overline{\text{PORST}}$), the ADSP-2192 can also be reset by the PCI or USB interfaces. These interfaces reset the DSP under their control as needed. A reset via the PCI or USB device causes program flow to jump to the command monitor that is part of the loader kernel, bypassing the serial EEPROM detection/reading subroutines.

For more information about resets via PCI or USB, see "Resets" on page 8-14.

Software Reset

The DSP can generate a software reset by using the RSTD bit in the DSP Interrupt/Powerdown Registers. (See "ADSP-2192 DSP Peripheral Registers" on page B-1 for more information about the RSTD bit.) Generally, reset conditions are handled by forcing the DSPs to execute ROM- or RAM-based Reset Handler code. The Reset Handler to be executed can be dictated by the Reset Source as defined by the CRST[1:0] bits in the Chip Mode/Status Register (CMSR). If not otherwise defined, the loader kernel jumps to the first location of internal PM memory at address 0x10000 and commences execution.

The exact Reset Functionality is therefore defined by the ROM and RAM Reset Handler Code and as such is programmable.

Reset Progression

Once a reset has occurred and the loader kernel begins running, it does the following:

- Determines the type of reset (Power On Reset, PCI/USB Reset, or Software Reset),
- Configures interrupts

- Reads data from the serial EEPROM if appropriate
- Sets up bus configurations
- Transfers control to PCI or USB

At this point the loader kernel enters into an infinite loop, waiting for instructions. Once the PCI or USB device has completed booting the DSP, the device can write an instruction to a pre-defined location (`0x000000` in Data Memory), at which point the DSP will execute any one of a list of supported commands. These supported commands include the following:

- Jump to program memory without returning (to leave the loader kernel and begin user code)
- Read a word from EEPROM
- Enable write mode on EEPROM
- Write a word to EEPROM
- Re-read patch block from EEPROM (if bus configuration may have overwritten locations)
- Enter power-down state

You can also write code to perform these operations during runtime by writing the appropriate value into the pre-defined memory address and then by performing a `CALL` to address `0x014F00` (a location in the ROM). The loader kernel performs the requested command and then returns control to the user code.

Table 11-9 shows the values and descriptions for the supported functions. For commands that require multiple arguments, the arguments are placed in Data Memory addresses `0x0001`, `0x0002`, and `0x0003` respectively.

Table 11-9. User-Defined Loader Kernel Function Values

		0x0001	0x0002	0x0003
Value	**Description**	**Argument 1**	**Argument 2**	**Argument 3**
0x0000	nop	N/A	N/A	N/A
0x0004	jump_to_code	N/A	dest. address	N/A
0x0006	eeprom_write_enable	N/A	N/A	N/A
0x0002	eeprom_write_word	eeprom address	source address	source page
0x0005	eeprom_write_imm_data	eeprom address	imm. value	N/A
0x0001	eeprom_read_word	eeprom address	dest. address	dest. page
0x0003	read_patch_block	eeprom address	N/A	N/A
0x0007	powerdown_dsp	N/A	N/A	N/A

Resets and Software-Forced Rebooting

Table 11-10 on page 11-17 shows the state of the processor registers after a reset or a software-forced reboot. The values of any registers not listed are unchanged by a reset or a reboot. The contents of on-chip memory are unchanged after $\overline{\text{PORST}}$, except for the data-memory-mapped control/status registers, as shown in Table 11-10.

During booting (and rebooting), all interrupts are masked, and auto-buffering is disabled. The timers run during a reboot. If a timer interrupt occurs during the reboot, it is masked. Thus, if more than one timer interrupt occurs during the reboot, the processor latches only the first, and a timer overrun can occur.

Table 11-10. ADSP-2192 Register State after Reset or Software Reboot

Register Name	Value Changed after Reset or Software Reboot?	New Value
AX0	no	unchanged
AX1	no	unchanged
MX0	no	unchanged
MX1	no	unchanged
AY0	no	unchanged
AY1	no	unchanged
MY0	no	unchanged
MY1	no	unchanged
MR2	no	unchanged
SR2	no	unchanged
AR	no	unchanged
SI	no	unchanged
MR1	no	unchanged
SR1	no	unchanged
MR0	no	unchanged

Table 11-10. ADSP-2192 Register State after Reset or Software Reboot

Register Name	Value Changed after Reset or Software Reboot?	New Value
SR0	no	unchanged
I0	no	unchanged
I1	no	unchanged
I2	no	unchanged
I3	no	unchanged
M0	no	unchanged
M1	no	unchanged
M2	no	unchanged
M3	no	unchanged
L0	no	unchanged
L1	no	unchanged
L2	no	unchanged
L3	no	unchanged
IMASK	yes	0
IRPTL	yes	0
ICNTL	yes	0
STACKA	no	unchanged
I4	no	unchanged

Table 11-10. ADSP-2192 Register State after Reset or Software Reboot

Register Name	Value Changed after Reset or Software Reboot?	New Value
I5	no	unchanged
I6	no	unchanged
I7	no	unchanged
M4	no	unchanged
M5	no	unchanged
M6	no	unchanged
M7	no	unchanged
L4	no	unchanged
L5	no	unchanged
L6	no	unchanged
L7	no	unchanged
TX0	no	unchanged
TX1	no	unchanged
CNTR	no	unchanged
LPSTCKA	no	unchanged
ASTAT	yes	0
MSTAT	yes	0
SSTAT	yes	0x55

Table 11-10. ADSP-2192 Register State after Reset or Software Reboot

Register Name	Value Changed after Reset or Software Reboot?	New Value
LPSTCKP	no	unchanged
CCODE	yes	0x8
SE	no	unchanged
SB	no	unchanged
PX	no	unchanged
DMPG1	yes	0
DMPG2	yes	0
IOPG	yes	0
IJPG	yes	0
RX0	no	unchanged
RX1	no	unchanged
STACKP	no	unchanged
AF	no	unchanged
B0	no	unchanged
B1	no	unchanged
B2	no	unchanged
B3	no	unchanged
B4	no	unchanged

Table 11-10. ADSP-2192 Register State after Reset or Software Reboot

Register Name	Value Changed after Reset or Software Reboot?	New Value
B5	no	unchanged
B6	no	unchanged
B7	no	unchanged
SYSCTL	no	unchanged
DMAPAGE	no	unchanged
CACTL	yes	0
STCTL0	no	unchanged
SRCTL0	no	unchanged
TX0	no	unchanged
RX0	no	unchanged
STCTL1	no	unchanged
SRCTL1	no	unchanged
TX1	no	unchanged
RX1	no	unchanged
TPERIOD	no	unchanged
TCOUNT	no	unchanged
TSCALE	no	unchanged

Table 11-10. ADSP-2192 Register State after Reset or Software Reboot

Register Name	Value Changed after Reset or Software Reboot?	New Value
TSCALECNT	no	unchanged
FLAGS	yes	0

Interrupts

See “ADSP-2192 Interrupts” on page E-1 for information about interrupts on the ADSP-2192.

Flag Pins

The ADSP-2192 processor has eight dedicated general-purpose flag pins, IO0-7. These flags can be programmed as either inputs or outputs; they default to inputs following reset. The IOx pins are programmed with the use of two memory-mapped registers. The value of the GPIO configuration register determines the flag direction: `0`=output and `1`=input. The Programmable Flag Data register is used to read and write the values on the pins. (Refer to “ADSP-2192 DSP Peripheral Registers” on page B-1 and “Setting Dual DSP Core Features” on page 6-3 for more information about these registers.)

Data being read from a pin configured as an input is synchronized to the processor’s clock. Pins configured as outputs drive the appropriate output value. When the GPIO status register is read, any pins configured as outputs will read back the value being driven out; the status is “sticky”; writing a zero clears it, but writing a one has no effect.

Powerup and Powerdown

This section discusses all possible power states on the ADSP-2192, including the PCI, USB, and AC'97 peripheral interfaces.

The power states of the two DSPs are independent of each other. Each DSP can be in one of the following states: running, in idle with memory clocks running, in idle with memory clocks stopped, or completely powered down.

Each AC'97 codec can be powered up, powered down, or in one of several power levels in between (including DACs only powered down, ADCs only powered down, or the entire link powered down). See "AC'97 Codec Port" on page 9-1 for more information about AC'97 and its power states.

The PCI interface supports the following PCI power management states: `D0`, `D1`, `D2`, `D3hot`, and `D3cold`. See "Host (PCI/USB) Port" on page 8-1 for more information about PCI and its power management states.

The USB interface supports the following power management states: running, reset, or suspended. See "Host (PCI/USB) Port" on page 8-1 for more information about USB and its power management sates.

If everything else on the ADSP-2192 is powered down, the clock crystal can either be running or stopped.

See "Power Management Description" on page 11-28 for more information about power issues on the ADSP-2192.

Powerup Issues

The ADSP-219x dual-voltage processor (ADSP-2192) has special issues related to powerup. These issues include the powerup sequence and the dual-voltage power supplies. This section discusses both of these issues. It also gives information about reset generators, which provide a reliable active reset once the power supplies and internal clock circuits have stabilized.

Powerup Sequence

Each of the DSPs on the ADSP-2192 has a register to control powerup and powerdown functions. These registers are PWRP0 (the DSP1 Interrupt/Powerdown Register) and PWRP1 (the DSP2 Interrupt/Powerdown Register).

To power up one of the DSPs, write a 1 to the PU (power up) control bit of that DSP's Interrupt/Powerdown Register. Writing this value causes the DSP to exit the IDLE within its powerdown handler, effectively powering up. The same process can also be used to abort a powerdown; if the DSP is in the powerdown handler prior to the IDLE, writing a 1 causes execution to immediately continue through the IDLE without stopping the clocks.

To power down one of the DSPs, write a 1 to the PD (power down) control bit of that DSP's Interrupt/Powerdown Register. Writing this value causes the DSP to enter its powerdown handler. The same process can be used to abort a powerup. If the DSP is in the powerdown handler after executing an IDLE, writing a 1 causes the DSP to immediately re-enter the powerdown handler after executing the RTI.

The current value of the PD and PU control bits indicate the current state of the DSP. If PD=1, this DSP is powered down; either it is in the powerdown handler and has executed an IDLE instruction, or the DSP Clock Generator (PLL) is not running and stable. If PU=1, this DSP is in the powerdown interrupt handler, whether or not it has executed the powerdown IDLE.

If both DSPs are powered down, the DSP clock generator is also powered down. The DSP clock generator restarts automatically when either DSP wakes up. DSP memory cannot be accessed via PCI when the DSP clock generator is powered down, and memory reads must not be performed while the DSPs are powering up.

The following recommendations should be observed when powering up dual-voltage DSPs. Ideally, the two supplies, V_{DDEXT} and V_{DDINT}, should be powered up together. If they cannot be powered up together, the internal (core) supply should be powered up first to reduce the risk of latchup events.

As shown in Figure 11-3 on page 11-26, a network of protection diodes, isolates the internal supplies and provides ESD protection for the IO pins. When applying power separately to the V_{DDINT} or V_{DDEXT} pins, limit the maximum supply current and duration that would be conducted through the isolation diodes if the unpowered pins are at ground potential.

If an external master clock is used, it should not be driving the XTALI pin when the DSP is unpowered. The clock must be driven immediately after powerup; otherwise, internal gates stay in an undefined (hot) state and can draw excess current. After powerup, there should be sufficient time for the internal PLL to stabilize (2000 clock cycles) before the reset is released.

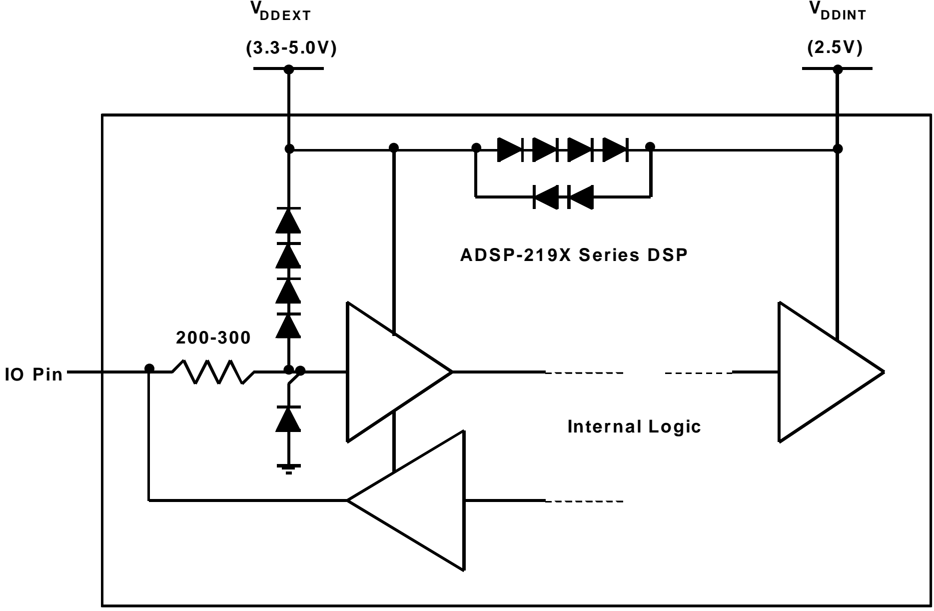


Figure 11-3. Protection Diodes and IO Pin ESD Protection

Power Regulators

The ADSP-2192 is intended to operate in a variety of different systems. These include PCI, CardBus, USB and embedded (Sub-ISA) applications. The PCI and USB specifications define power consumption limits that constrain the ADSP-2192 design; see "Host (PCI/USB) Port" on page 8-1 for more information.

2.5V Regulator Options

In 5V and 3.3V PCI applications the ADSP-2192 2.5V IVDD supply will be generated by an on-chip regulator. The internal 2.5V supply (IVDD) can be generated by the on-chip regulator combined with an external power transistor as shown in Figure 11-4 on page 11-27. To support the PCI specification's power down modes, the two transistors control the primary and auxiliary supply. If the reference voltage on RVDD (typically the same as PCIVDD) drops out, the VCTRLAUX will switch on the device connected to PCIVAUX and VCTRLVDD will switch off the primary supply. USB applications may require an external high-efficiency switching regulator to generate the 2.5V supply for the ADSP-2192.

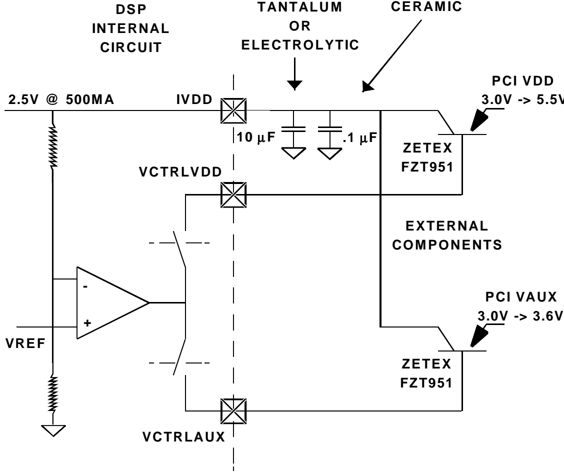


Figure 11-4. ADSP-2192 2.5V Regulator Options

Power Management Description

The ADSP-2192 supports several hardware and software states with distinct power management and functionality capabilities.

The driver and DSP code take responsibility for detailed power management of the modem, so minimum power levels are achieved regardless of OS or BIOS. The driver and DSPs manage power by changing platform states as necessary in response to events.

Each DSP can be in one of several power management states. A DSP can be running, in idle mode with memory clocks running, in idle mode with memory clocks stopped, or powered down. If everything else on the chip is powered down, the crystal can be running or not running.

In addition to the power management states of each DSP, the ADSP-2192 interfaces each have their own power management states. These states are not totally dependent on the hardware but are controlled by the driver software and can be changed to reduce overall power on the chip. The PCI interface supports the following power management states: `D0`, `D1`, `D2`, `D3hot`, and `D3cold`. The USB can be running, reset, or suspended. The AC'97 codecs can be powered up or powered down. Additionally, they have power management options to power down only DACs or ADCs, or to power down the link.

Powerdown

In addition to supporting powerdown modes for the PCI, USB, and AC'97 standards, the ADSP-2192 supports additional powerdown modes for the DSP cores and peripheral buses. The powerdown modes are controlled by the DSP1 and DSP2 Interrupt/Powerdown registers.

The ADSP-2192 processor provides a powerdown feature that allows the processor to enter a very low power dormant state through hardware or software control. (Refer to the processor data sheet for exact power consumption specifications.)

The powerdown feature is useful for applications where power conservation is necessary (for example in battery-powered operation).

The powerdown feature has the following effects:

- Internal clocks are disabled
- Processor registers and memory contents are maintained
- The chip can recover from powerdown in less than 100 `XTALI` cycles
- The chip can disable internal oscillator when using crystal
- Processor does not need to shut down clock for lowest power when using external oscillator
- Interrupt support enables "housekeeping" code to execute before entering powerdown and after recovering from powerdown
- User-selectable powerup context is provided

Even though the processor is put into the powerdown mode, the lowest level of power consumption still might not be achieved if certain guidelines are not followed. Lowest possible power consumption requires no additional current flow through processor output pins and no switching activity on active input pins. Therefore, a careful analysis of pin loading in the circuit is required.

The following sections detail the proper powerdown procedure and provide guidelines for clock and output pin connections required for optimum low-power performance. Refer to "AC'97 Codec Port" on page 9-1 for more information about powering the ADSP-2192 up or down through the AC'97 interface, or to "Host (PCI/USB) Port" on page 8-1 for more information about doing this through the USB or PCI interfaces.

Powerdown Control

The ADSP-2192 supports two states with distinct power management and functionality capabilities. These states are referred to as *Platform States* and are denoted PS0 and PS1.

These platform states encompass both hardware and software states. The driver and DSP code take responsibility for detailed power management, and minimum power levels are achieved regardless of OS or BIOS. The driver and DSPs manage power by changing platform states as necessary in response to events. Such events may include changes in the function's PCI/USB power management D-state or PME_Enable state (set via the external PME pin), or external wakeup events detected on the external dedicated general-purpose flag pins (IO0-7). See Table B-5 on page B-18 for more information about the PME bit.

The PS0 platform state indicates that the platform is running and is operational. The power state is D0 and the chip is at full power.

The PS1 platform state indicates that the platform is shut down and is in the lowest power state. The power state may be D0, D3, or D3cold. The DSPs are powered down, the AC'97 is shut down, and XTAL and the clocks to the DSPs are stopped. No wakeup is enabled, and any enabled wake events signal $\overline{\text{PME}}$ directly without DSP intervention.

Entering and Exiting Powerdown

Each of the DSPs on the ADSP-2192 has a register to control powerdown and powerup functions. These registers are PWRP1 (the DSP1 Interrupt/Powerdown Register) and PWRP2 (the DSP2 Interrupt/Powerdown Register).

To power down one of the DSPs, write a 1 to the PD (power down) control bit of that DSP's Interrupt/Powerdown Register. Writing a 1 causes the DSP to enter its powerdown handler. The same process can be used to abort a powerup; if the DSP is in the powerdown handler after executing an IDLE, writing a 1 causes the DSP to re-enter the powerdown handler immediately after executing the RTI.

To power up one of the DSPs, write a 1 to the PU (power up) control bit of that DSP's Interrupt/Powerdown Register. Writing a 1 causes the DSP to exit the IDLE within its powerdown handler, effectively powering up. The same process can also be used to abort a powerdown. If the DSP is in the powerdown handler prior to the IDLE, writing a 1 causes execution to continue immediately through the IDLE without stopping the clocks.

The current value of the PD and PU control bits indicate the current state of the DSP. If PD=1, this DSP is powered down; either it is in the powerdown handler and has executed an IDLE instruction, or the DSP Clock Generator (PLL) is not running and stable. If PU=1, this DSP is in the powerdown interrupt handler, whether or not it has executed the powerdown IDLE.

If both DSPs are powered down, the DSP clock generator is also powered down; the DSP clock generator restarts automatically when either DSP wakes up. DSP memory cannot be accessed via PCI when the DSP clock generator is powered down, and memory reads must not be performed while the DSPs are powering up.

While the processor is in the powerdown mode, the processor is in CMOS standby. This feature allows the lowest level of power consumption where most input pins are ignored. Active inputs need to be held at CMOS levels to achieve lowest power. More information can be found in the section "Processor Operation During Powerdown" on page 11-36.

Powering Down the USB

The ADSP-2192 can be powered down through the USB interface. To do this, the software driver sends `USB REGWR` commands. The `USB REGWR` command comes down on the control pipe (Endpoint `0`) and writes the appropriate bits to registers `PWRP1` (DSP 1 Interrupt/Powerdown Register at Page `0x00`, Address `0x08`) and `PWRP2` (DSP 2 Interrupt/Powerdown Register at Page `0x00`, Address `0x0A`).

Once these bits have been written to the `PWRP1` and `PWRP2` registers, the DSPs park in their idle states. If a full USB powerdown is desired, the USB host then signals the `SUSPEND` state to the ADSP-2192. When the `SUSPEND` state is signaled, the USB interface sees this bus state and notifies the ADSP-2192, causing the internal USB clocks to stop, at which time the device enters its lowest power state.

Powering Down the PCI

Refer to "Power Management Functions" on page B-18 for information about powering down the PCI.

Powering Down the AC'97 Link

The recommended way to enable and disable the link is through use of the LKEN bit. When LKEN is set to 1, the following results can occur:

- If the link was already running (LKOK=1), there is no effect.
- If the link was in cold reset, $\overline{\text{ACRST}}$ is deasserted (clearing ACTL:AFR if necessary).
- If the link was in warm reset, SYNC is asserted for 1 µs and is then deasserted.
- If BITCLK is internal, the BITCLK generator starts (ACTL:BCEN set to 1) when $\overline{\text{ACRST}}$ deasserts (cold) or when SYNC deasserts (warm). If BITCLK is external, the external primary codec must start BITCLK in a similar manner.
- When BITCLK restarts (ASTAT:BCOK reads 1), the sync pulse generator is automatically enabled (ATL:SYEN set to 1). At this point, LKOK reads 1.

For more information about AC'97 power modes, refer to "AC'97 Codec Port" on page 9-1.

Entering Powerdown

The powerdown sequence is defined as follows:

1. Initiate the powerdown sequence by writing a 1 to the PD bits of the PWRP1 and PWRP2 registers.
2. The processor vectors to the non-maskable powerdown interrupt vector at address 0x0004.

The powerdown interrupt is never masked. You must be careful not to cause multiple powerdown interrupts to occur, or stack overflow may result. Multiple powerdown interrupts can occur if the PWD input is pulsed while the processor is already servicing the powerdown interrupt.)

3. Any number of housekeeping instructions, starting at location 0x002C, can be executed prior to the processor entering the powerdown mode. Typically, this section of code is used to configure the powerdown state, disable on-chip peripherals, and clear pending interrupts.
4. The processor enters powerdown mode when it executes an IDLE instruction (while the PD bit is set). The processor may take either one or two cycles to power down, depending on internal clock states during the execution of the IDLE instruction. All register and memory contents are maintained while in powerdown. Also, all active outputs are held in whatever state they were in before going into powerdown.

Similarly, the powerdown sequence can be aborted by writing a 1 to the PU bits of the PWRP1 and PWRP2 registers. If an RTI is executed before the IDLE instruction, then the processor returns from the powerdown interrupt and the powerdown sequence is aborted.

Exiting Powerdown

The powerdown mode can be exited with the use of the PU bit of the PWRP1 and PWRP2 registers. Writing a 1 to that bit causes the powerdown sequence to be aborted. There are also several user-selectable modes for start-up from powerdown which specify a start-up delay and also specify the program flow after start-up. This feature allows the program to resume from where it left off before powerdown, or the program context to be cleared.

Ending Powerdown

Applying a low-to-high transition to the PU bits of the PWRP1 and PWRP2 registers takes the processor out of powerdown mode. The processor automatically selects the amount of time to wait before coming out of the powerdown mode. The PLL waits until it is stable before starting the clocks to the rest of the system; it stabilizes more quickly when the XON bit (the Xtal Force On bit in the CMSR register) is set because the crystal oscillator remains active. For more information, see "Using an External TTL/CMOS Clock" on page 11-36.

Ending Powerdown with the PORST Pin

If $\overline{\text{PORST}}$ (power on reset) is asserted while the processor is in the powerdown mode, the processor is reset and instructions are executed from address 0x10000. A boot is performed if the boot mode is set. If the $\overline{\text{PORST}}$ pin is used to exit powerdown, it must be held low for the appropriate number of cycles. If the clock is stopped at powerup or is operating at a different frequency at powerup than it was before powerdown, $\overline{\text{PORST}}$ must be held long enough for the oscillator to stabilize, plus an additional 1000 XTALI cycles for the phase locked loop (PLL) to lock. The time required for the oscillator to stabilize depends upon the type of crystal used and capacitance of the external crystal circuit. Typically 2000 XTALI cycles is adequate for clock stabilization time.

If the clock was not stopped at powerup and is at a stable frequency at powerup (same as before powerdown), only 5 cycles of PORST are required.

Startup Time after Powerdown

The time required to exit the powerdown state depends on whether an internal or external oscillator is used, and the method used to exit powerdown.

Using an External TTL/CMOS Clock

When in PCI or CardBus mode, the external clock signal is ignored if both DSPs and the AC'97 link are powered down and the XON (Xtal Force On) bit in the CMSR register has not been set. When in USB mode, the USB interface must also be suspended for the clock to be ignored. In Sub-ISA mode, the clock must remain running.

The external clock is ignored internally; you do not need to stop it, since no power is wasted while it is running. Since the PCI and USB interfaces can restart the ADSP-2192 without warning, we recommend that the external clock remain running as long as there is power in the system.

Processor Operation During Powerdown

Some processor circuitry may still be active during powerdown mode. Also, some output pins remain active. A good understanding of these states will allow you to determine the best low-power configuration for your system. By keeping output loading and input switching to a minimum, the lowest possible power consumption can be achieved.

Interrupts And Flags

DSP interrupts are latched and will be serviced when the processor exits powerdown. Any activity on the flag input pins during a low power state may increase the power consumption. There should also be no resistive load on the flag output pins (as with any active output pin) if lowest power is desired.

Conditions for Lowest Power Consumption

All pins on the ADSP-2192 remain active as long as power is maintained to the chip. This chip does not have a specifically-defined powerdown state; at any time either or both of the two processors can be in a low power state, and any or all of the interfaces can be in a low power state. Because of this wide variety of power state options, each interface (and its associated pins) must follow a bus standard power state specific to that interface. Each interface is maintained in a power state as defined by the standard for that interface.

To assure the lowest power consumption, all active input pins should be held at a CMOS level (to ground level, if possible). All active output pins should be free of resistive load, since load current increases power dissipation. You must perform a careful analysis of each input and output pin in order to ensure the lowest power dissipation.

Some inputs are active but are ignored. The state of these inputs does not matter as long as they are at a CMOS level.

Additionally, each peripheral interface (USB, PCI, and AC'97) can be put into a low power mode, as described in the following sections.

AC'97 Low Power Mode

The AC'97 link is powered down or is put into a low power state by commands issued to the external AC'97 codec. The AC'97 link can be in a cold powerdown state or a warm powerdown state. In the cold powerdown state, $\overline{\text{ACRST}}$ is asserted, and BITCLK and SYNC are halted. In the warm powerdown state, BITCLK and SYNC are also halted, but $\overline{\text{ACRST}}$ is deasserted.

At powerup, deasserting $\overline{\text{ACRST}}$ from the cold state causes the BITCLK master to start, but the controller does not necessarily start generating SYNC pulses until it is enabled. The state in which BITCLK is running but SYNC is halted is called *IDLE*.

When powered down, a wakeup protocol is defined using the SDI pins. Wakeups are signalled using SDI Slot 12 Bit 0 during the running state; wakeups are signalled using a high level on SDI during all other states. If it is enabled to do so, the controller can restart the link upon receiving this wakeup signal.

For more information about AC'97 power modes, refer to "AC'97 Codec Port" on page 9-1.

Using Powerdown as A Non-Maskable Interrupt

The powerdown interrupt is never masked, although it can be disabled with the `DIS INTS` instruction. It is possible to use this interrupt for other purposes if desired. The processor does not go into powerdown until an `IDLE` instruction is executed. If an `RTI` is executed before the `IDLE` instruction, then the processor returns from the powerdown interrupt and the powerdown sequence is aborted.

It is possible to place a series of instructions at the powerdown interrupt vector location `0x002C`. This routine should end with an `RTI` instruction and should not contain an `IDLE` instruction if the interrupt is to be used for purposes other than powerdown.

Emulation

Analog Devices DSP emulators use the JTAG test access port of the ADSP-2192 processor to monitor and control the target board processor during emulation. The emulator provides full-speed emulation, allowing inspection and modification of memory, registers, and processor stacks. Non-intrusive in-circuit emulation is assured by the use of the processor's JTAG interface; the emulator does not affect target system loading or timing.

Note that the ADSP-2192 JTAG port does not support boundary scan.

For more information about JTAG emulation, see the "JTAG Test-Emulation Port" on page 10-1.

EZ-KIT Lite

To make it easier to evaluate the ADSP-219x DSP family for your application, Analog Devices sells the ADSP-2192 EZ-KIT Lite™. This kit provides developers with a cost-effective method for evaluating of the ADSP-219x family of DSPs.

The EZ-KIT Lite includes an ADSP-2192 DSP evaluation board and fundamental debugging software. The evaluation board in this kit contains an ADSP-2192 digital signal processor, Audio type Codec, breadboard area, Flag LED, Reset/Interrupt/Flag push buttons, and ADSP-2192 peripheral port connectors. The peripheral connectors include a JTAG test and emulation port connector that supports the Analog Devices emulators and PCI and USB connections.

The ADSP-2192 EZ-KIT Lite comes with an evaluation suite of the VisualDSP++ integrated development environment with the C/C++ compiler, assembler, and linker that supports typical debug functions, including memory/register read and write, halt, run, and single step. The use of all software tools is limited to the EZ-KIT Lite product.

For more information, refer to the documentation shipped with the EZ-KIT Lite.

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, and is an excellent source of information and practical ideas. Topics covered in the book include:

- High-Speed Properties of Logic Gates
- Measurement Techniques
- Transmission Lines
- Ground Planes and Layer Stacking
- Terminations
- Vias
- Power Systems
- Connectors
- Ribbon Cables
- Clock Distribution
- Clock Oscillators

Reference: Johnson and Graham, *High-Speed Digital Design: A Handbook of Black Magic*, Prentice Hall, Inc., ISBN 0-13-395724-1

A ADSP-219X DSP CORE REGISTERS

Overview

The DSP core 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 (for system control registers) memory-mapped addresses. Information on each type of register is available at the following locations:

- "Core Status Registers" on page A-8
- "Computational Unit Registers" on page A-15
- "Program Sequencer Registers" on page A-18
- "Data Address Generator Registers" on page A-24
- "Memory Interface Registers" on page A-26

Outside of the DSP core, a set of registers control I/O peripherals. For information on these product-specific registers, see "ADSP-2192 DSP Peripheral Registers" on page B-1.

When writing DSP programs, it is often necessary to set, clear, or test bits in the DSP'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 names of bits or registers.

For convenience and consistency, Analog Devices provides a header file that provides bit and register symbols and their corresponding names (bit and register definitions). For core register definitions, see the "Register and Bit #Defines File" on page A-27. For off-core register definitions, see the "Register and Bit #Defines File" on page B-95.

Many registers have reserved bits. When writing to a register, programs may only clear (write zero to) the register's reserved bits.

Core Registers Summary

The DSP has three categories of registers: core registers, system control registers, and I/O registers. Table A-1 lists and describes the DSP's core registers. The DSP core registers divide into register group (DREG, REG1, REG2, and REG3) based on their opcode identifiers and functions. Table A-2 shows these groups. For more information on how registers may be used within instructions, see the *ADSP-219x DSP Instruction Set Reference*.

Table A-1. Core Registers

Type	Registers	Function
Status	ASTAT MSTAT SSTAT (read-only)	Arithmetic status flags Mode control and status flags System status
Computational Units	AX0, AX1, AY0, AY1, AR, AF, MX0, MX1, MY0, MY1, MR0, MR1, MR2, SI, SE, SB, SR0, SR1, SR2	Data register file registers provide Xop and Yop inputs for computations. AR, SR, and MR receive results. In this text, the word Dreg denotes unrestricted use of data registers as a data register file, while the words XOP and YOP denote restricted use. The data registers (except AF, SE, and SB) serve as a register file, for unconditional, single-function instructions.
Shifter	SE SB	Shifter exponent register Shifter block exponent register

Table A-1. Core Registers (Continued)

Type	Registers	Function
Program flow	CCODE LPSTACKA LPSTACKP STACKA STACKP	Software condition register Loop stack address register, 16 address LSBs Loop stack page register, 8 address MSBs PC stack address register, 16 address LSBs PC stack page register, 8 address MSBs
Interrupt	ICNTL IMASK IRPTL	Interrupt control register Interrupt mask register Interrupt latch register
DAG address	I0, I1, I2, I3 I4, I5, I6, I7	DAG1 index registers DAG2 index registers
	M0, M1, M2, M3 M4, M5, M6, M7	DAG1 modify registers DAG2 modify registers
	L0, L1, L2, L3 L4, L5, L6, L7	DAG1 length registers DAG2 length registers
System control	B0, B1, B2, B3, B4, B5, B6, B7, CACTL	DAG1 base address registers (B0-3), DAG2 base address registers (B4-7), Cache control
Page	DMPG1 DMPG2 IJPG IOPG	DAG1 page register, 8 address MSBs DAG2 page register, 8 address MSBs Indirect jump page register, 8 address MSBs I/O page register, 8 address MSBs
Bus exchange	PX	Holds eight LSBs of 24-bit memory data for transfers between memory and data registers only

Table A-2. ADSP-219x DSP Core Registers

RGP/Address	Register Groups (RGP)			
Address	**00 (DREG)**	**01 (REG1)**	**10 (REG2)**	**11 (REG3)**
0000	AX0	I0	I4	ASTAT
0001	AX1	I1	I5	MSTAT
0010	MX0	I2	I6	SSTAT
0011	MX1	I3	I7	LPSTACKP
0100	AY0	M0	M4	CCODE
0101	AY1	M1	M5	SE
0110	MY0	M2	M6	SB
0111	MY1	M3	M7	PX
1000	MR2	L0	L4	DMPG1
1001	SR2	L1	L5	DMPG2
1010	AR	L2	L6	IOPG
1011	SI	L3	L7	IJPG
1100	MR1	IMASK	Reserved	Reserved
1101	SR1	IRPTL	Reserved	Reserved
1110	MR0	ICNTL	CNTR	Reserved
1111	SR0	STACKA	LPCSTACKA	STACKP

Register Load Latencies

An effect latency occurs when some instructions write or load a value into a register, which changes the value of one or more bits in the register. Effect latency refers to the time it takes after the write or load instruction for the effect of the new value to become available for other instructions to use.

Effect latency values are given in terms of instruction cycles. A 0 latency means that the effect of the new value is available on the next instruction following the write or load instruction. For register changes that have an effect latency greater than 0, you should try not to use the register immediately after writing or loading a new value; you may end up using the old value rather than the new value. Table A-3 gives the effect latencies for writes or loads of various interrupt and status registers.

Table A-3. Effect Latencies for Register Changes

Register	Bits	REG = value	ENA/DIS mode	POP STS	SET/CLR INT
ASTAT	All	1 cycle	NA	0 cycles	NA
CCODE	All	1 cycle	NA	NA	NA
CNTR	All	1 cycle[1]	NA	NA	NA
ICNTL	All	1 cycle	NA	NA	0 cycles
IMASK	All	1 cycle	NA	0 cycles	NA

Table A-3. Effect Latencies for Register Changes (Continued)

Register	Bits	REG = value	ENA/DIS mode	POP STS	SET/CLR INT
MSTAT	SEC_REG	1 cycle	0 cycles	1 cycle	NA
	BIT_REV	3 cycles	0 cycles	3 cycles	NA
	AV_LATCH	0 cycles	0 cycles	0 cycles	NA
	AR_SAT	1 cycle	0 cycles	1 cycle	NA
	M_MODE	1 cycle	0 cycles	1 cycle	NA
	TIMER	1 cycle	0 cycles	1 cycle	NA
	SEC_DAG	3 cycles	0 cycles	3 cycles	NA
CACTL	CPE	5 cycles	NA	NA	NA
	CDE	5 cycles	NA	NA	NA
	CFZ	4 cycles	NA	NA	NA

1 This latency applies only to IF COND instructions, not to the DO UNTIL instruction. Loading the CNTR register has 0 effect latency for the DO UNTIL instruction.

A PUSH or POP PC has one cycle of latency for all SSTAT register bits, but a PUSH or POP LOOP or STS has one cycle of latency only for the STKOVERFLOW bit in the SSTAT register.

When loading some Group 2 and 3 registers (see Table A-3 on page A-5), the effect of the new value is not immediately available to subsequent instructions that might use it. For interlocked registers (DAG address and page registers, IOPG, IJPG), the DSP automatically inserts stall cycles as needed. However, for non-interlocked registers (to accommodate the required latency), programs must insert either the necessary number of NOP instructions or other instructions that are not dependent upon the effect of the new value.

The non-interlocked registers are:

- Status registers ASTAT and MSTAT
- Condition code register CCODE
- Interrupt control register ICNTL

The number of NOP instructions to insert is specific to the register and the load instruction, as shown in Table A-3. A zero (0) latency indicates that the new value is effective on the next cycle after the load instruction executes. An *n* latency indicates that the effect of the new value is available up to *n* cycles after the load instruction executes. When using a modified register before the required latency, the DSP provides the register's old value.

Since unscheduled or unexpected events (interrupts, DMA operations, etc.) often interrupt normal program flow, do not rely on these load latencies when structuring program flow. A delay in executing a subsequent instruction based on a newly loaded register could result in erroneous results—whether the subsequent instruction is based on the effect of the register's new or old value.

Load latency applies only to the time it takes the loaded value to effect the change in operation, not to the number of cycles required to load the new value. A loaded value is always available to a read access on the next instruction cycle.

Core Status Registers

The DSP's control and status system registers configure how the processor core operates and indicate the status of many processor core operations. Table A-4 lists the processor core's control and status registers with their initialization values. Descriptions of each register follow.

Table A-4. Core Status Registers

Register Name and Page Reference	Initialization After Reset
"Arithmetic Status (ASTAT) Register" on page A-9	b#0 0000 0000
"Mode Status (MSTAT) Register" on page A-11	b#000 000
"System Status (SSTAT) Register" on page A-14	b#0101 0101

Arithmetic Status (ASTAT) Register

The ASTAT register is a non-memory-mapped, register group 3 register (REG3). The reset value for this register is b#0 0000 0000. The DSP updates the status bits in ASTAT, indicating the status of the most recent ALU, multiplier, or shifter operation.

Table A-5. ASTAT Register Bit Definitions

Bit	Name	Description
0	AZ	ALU result zero. Logical NOR of all bits written to the ALU result register (AR) or ALU feedback register (AF). 0 = ALU output ≠ 0 1 = ALU output = 0
1	AN	ALU result negative. Sign of the value written to the ALU result register (AR) or ALU feedback register (AF). 0 = ALU output positive (+) 1 = ALU output negative (–)
2	AV	ALU result overflow. 0 = No overflow 1 = Overflow
3	AC	ALU result carry. 0 = No carry 1 = Carry
4	AS	ALU *x* input sign. Sign bit of the ALU *x*-input operand; set by the ABS instruction only. 0 = Positive (+) 1 = Negative (–)

Table A-5. ASTAT Register Bit Definitions (Continued)

Bit	Name	Description
5	AQ	ALU quotient. Sign of the resulting quotient; set by the DIVS or DIVQ instructions. 0 = Positive (+) 1 = Negative (–)
6	MV	Multiplier overflow. Records overflow/underflow condition for MR result register. 0 = No overflow or underflow 1 = Overflow or underflow
7	SS	Shifter input sign. Sign of the shifter input operand. 0 = Positive (+) 1 = Negative (–)
8	SV	Shifter overflow. Records overflow/underflow condition for SR result register. 0 = No overflow or underflow 1 = Overflow or underflow

Mode Status (MSTAT) Register

The MSTAT register is a non-memory-mapped, register group 3 register (REG3). The reset value for this register is b#000 0000.

Table A-6. MSTAT Register Bit Definitions

Bit	Name	Description
0	SEC_REG	Secondary data registers. Determines which set of data registers is currently active. 0 = Deactivate secondary set of data registers (default). Primary register set (set that is active at reset) enabled and used for normal operation; secondary register set disabled. 1 = Activate secondary set of data registers. Secondary register set enabled and used for alternate DSP context (for example, interrupt servicing); primary register set disabled, current contents preserved. For details, see "Switching Contexts" in the *ADSP-219x DSP Instruction Set Reference*.
1	BIT_REV	Bit-reversed address output. Enables and disables bit-reversed addressing on DAG1 index registers only. 0 = Disable 1 = Enable For details, see "Bit-Reversed Addressing" in the *ADSP-219x DSP Instruction Set Reference*.

Table A-6. MSTAT Register Bit Definitions (Continued)

Bit	Name	Description
2	AV_LATCH	ALU overflow latch mode. Determines how the ALU overflow flag, AV, gets cleared. 0 = Disable Once an ALU overflow occurs and sets the AV bit in the ASTAT register, the AV bit remains set until explicitly cleared or is cleared by a subsequent ALU operation that does not generate an overflow. 1 = Enable Once an ALU overflow occurs and sets the AV bit in the ASTAT register, the AV bit remains set until the application explicitly clears it. For details on clearing the AV bit, see "Bit Manipulation: TSTBIT, SETBIT, CLRBIT, TGLBIT" and "Register to Register Move" in the *ADSP-219x DSP Instruction Set Reference*.
3	AR_SAT	ALU saturation mode. For signed values, determines whether ALU AR results that overflowed or underflowed are saturated or not. Enables or disables saturation for all subsequent ALU operations. 0 = Disable AR results remain unsaturated and return as is. 1 = Enable AR results saturated according to the state of the AV and AC status flags in ASTAT. AV AC AR register 0 0 ALU output 0 1 ALU output 1 0 0x7FFF 1 1 0x8000 Only the results written to the AR register are saturated. If results are written to the AF register, wraparound occurs, but the AV and AC flags reflect the saturated result.

Table A-6. MSTAT Register Bit Definitions (Continued)

Bit	Name	Description
4	M_MODE	MAC result mode. Determines the numeric format of multiplier operands. For all MAC operations, the multiplier adjusts the format of the result according to the selected mode. 0 = Fractional mode, 1.15 format. 1 = Integer mode, 16.0 format. For details, see "Data Format Options" in the *ADSP-219x DSP Instruction Set Reference*.
5	TIMER	Timer enable. Starts and stops the timer counter. 0 = Stops the timer count. 1 = Starts the timer count. For details on timer operation, see the "ADSP-2192 Timer" on page D-1.
6	SEC_DAG	Secondary DAG registers. Determines which set of DAG address registers is currently active. 0 = Primary registers. 1 = Secondary registers. For details, see "Secondary DAG Registers" and "Switching Contexts" in the *ADSP-219x DSP Instruction Set Reference*.

System Status (SSTAT) Register

The SSTAT register is a non-memory-mapped, register group 3 register (REG3). The reset value for this register is b#0000 0000.

Table A-7. SSTAT Register Bit Definitions

Bit	Name	Description
0	PCSTKEMPTY or PCE	PC stack empty. 0 = PC stack contains at least one pushed address. 1 = PC stack is empty.
1	PCSTKFULL or PCF	PC stack full. 0 = PC stack contains at least one empty location. 1 = PC stack is full.
2	PCSTKLVL or PCL	PC stack level. 0 = PC stack contains between 3 and 28 pushed addresses. 1 = PC stack is at or above the high-water mark (28 pushed addresses), or it is at or below the low-water mark (3 pushed addresses).
3	Reserved	
4	LPSTKEMPTY or LSE	Loop stack empty. 0 = Loop stack contains at least one pushed address. 1 = Loop stack is empty.
5	LPSTKFULL or LSF	Loop stack full. 0 = Loop stack contains at least one empty location. 1 = Loop stack is full.

Table A-7. SSTAT Register Bit Definitions (Continued)

Bit	Name	Description
6	STSSTKEMPTY or SSE	Status stack empty. 0 = Status stack contains at least one pushed status. 1 = Status stack is empty.
7	STKOVERFLOW or SOV	Stacks overflowed. 0 = Overflow/underflow has not occurred. 1 = At least one of the stacks (PC, loop, counter, status) has overflowed, or the PC or status stack has underflowed. This bit cleared only on reset. Loop stack underflow is not detected because it occurs only as a result of a POP LOOP operation.

Computational Unit Registers

The DSP's computational registers store data and results for the ALU, multiplier, and shifter. The inputs and outputs for processing element operations go through these registers.

The PX register lets programs transfer data between the data buses, but the data cannot be an input or output in a calculation.

Table A-8. Computational Unit Registers

Register	Initialization After Reset
"Data Register File (DREG) Registers" on page A-16	Undefined
"ALU X- and Y-Input (AX0, AX1, AY0, AY1) Registers" on page A-16	Undefined
"ALU Results (AR) Register" on page A-17	Undefined

Table A-8. Computational Unit Registers (Continued)

Register	Initialization After Reset
"Multiplier X- and Y-Input (MX0, MX1, MY0, MY1) Registers" on page A-17	Undefined
"Multiplier Results (MR2, MR1, MR0) Registers" on page A-17	Undefined
"Shifter Input (SI) Register" on page A-17	Undefined
"Shifter Exponent (SE) and Block Exponent (SB) Registers" on page A-18	Undefined

Data Register File (DREG) Registers

The DREG registers are non-memory-mapped, register group 0 registers. For unconditional, single-function instructions, the DSP has a data register file—a set of 16-bit data registers that transfer data between the data buses and the computation units. These registers also provides local storage for operands and results. For more information on how to use these registers, see "Data Register File" on page 2-57. The registers in the data register file include: AX0, AX1, MX0, MX1, AY0, AY1, MY0, MY1, MR2, SR2, AR, SI, MR1, SR1, MR0, and SR0.

ALU X- and Y-Input (AX0, AX1, AY0, AY1) Registers

The AX0, AX1, AY0, AY1 registers are non-memory-mapped, register group 0 registers. For conditional and/or multifunction instructions, some restrictions apply to data register usage. The registers that may provide Xop and Yop input to the ALU for conditional and/or multifunction instructions include: AX0, AX1, AY0, and AY1. For more information on how to use these registers, see "Multifunction Computations" on page 2-60.

ALU Results (AR) Register

The AR register is a non-memory-mapped, register group 0 register.The ALU places its results in the 16-bit AR register. For more information on how to use this registers, see "Arithmetic Logic Unit (ALU)" on page 2-17.

Multiplier X- and Y-Input (MX0, MX1, MY0, MY1) Registers

The MX0, MX1, MY0, and MY1 registers are non-memory-mapped, register group 0 registers. For conditional and/or multifunction instructions, some restrictions apply to data register usage.

The registers that may provide Xop and Yop input to the multiplier for conditional and/or multifunction instructions include: MX0, MX1, MY0, and MY1. For more information on how to use these registers, see "Multifunction Computations" on page 2-60.

Multiplier Results (MR2, MR1, MR0) Registers

The MR2, MR1, and MR0 registers are non-memory-mapped, register group 0 registers. The multiplier places results in the combined multiplier result register, MR. For more information on result register fields, see "Multiply—Accumulator (Multiplier)" on page 2-28.

Shifter Input (SI) Register

The SI register is a non-memory-mapped, register group 0 registers. For conditional and/or multifunction instructions, some restrictions apply to data register usage. The only registers that may provide input to the shifter for conditional and/or multifunction instructions is SI. For more information on how to use this registers, see "Multifunction Computations" on page 2-60.

Shifter Exponent (SE) and Block Exponent (SB) Registers

The SE and SB registers are non-memory-mapped, register group 3 registers. These register hold exponent information for the shifter. For more information on how to use these registers, see "Barrel-Shifter (Shifter)" on page 2-37.

The SB and SE registers are 16 bits in length, but all shifter instructions that use these registers as operands or update these registers with result values do not use the full width of these registers. Shifter instructions treat SB as being a 5 bit 2's complement register and treat SE as being an 8 bit 2's complement register.

Program Sequencer Registers

The DSP's Program Sequencer registers hold page addresses, stack addresses, and other information for determining program execution.

Table A-9. Program Sequencer Registers

Register	Initialization After Reset
"Interrupt Mask (IMASK) and Interrupt Latch (IRPTL) Registers" on page A-19	0x0000
"Interrupt Control (ICNTL) Register" on page A-20	0x0000
"Indirect Jump Page (IJPG) Register" on page A-21	0x00
"PC Stack Page (STACKP) and PC Stack Address (STACKA) Registers" on page A-21	Undefined
"Loop Stack Page (LPSTACKP) and Loop Stack Address (LPSTACKA) Register" on page A-22	Undefined

Table A-9. Program Sequencer Registers (Continued)

Register	Initialization After Reset
"Counter (CNTR) Register" on page A-22	Undefined
"Condition Code (CCODE) Register" on page A-23	Undefined
"Cache Control (CACTL) Register" on page A-23	b#101n nnnn

Interrupt Mask (IMASK) and Interrupt Latch (IRPTL) Registers

The IMASK and IRPTL registers are non-memory-mapped, register group 1 registers (REG1). The reset value for these registers is 0x0000.

Table A-10. IMASK and IRPTL Register Bit Definitions

Bit	Name	Description
0	EMU	Emulator. Nonmaskable. Highest priority
1	PWDN	Powerdown. Maskable only with GIE bit in ICNTL.
2	SSTEP	Single-step (during emulation)
3	STACK	Stack interrupt. Generated from any of the following stack status states: (if PCSTKE enabled) PC stack is pushed or popped and hits high-water mark, any stack overflows, or the status or PC stacks underflow.
4–14	User-defined	
15	User-defined	Lowest priority

Interrupt Control (ICNTL) Register

The ICNTL register is a non-memory-mapped, register group 1 register (REG1). The reset value for this register is 0x0000.

Table A-11. ICNTL Register Bit Definitions

Bit	Name	Description
0	reserved	write 0
1	reserved	write 0
2	reserved	write 0
3	reserved	write 0
4	INE	Interrupt nesting mode enable. 0 = Disabled 1 = Enabled
5	GIE	Global interrupt enable. 0 = Disabled 1 = Enabled
6	reserved	write 0
7	BIASRND	MAC biased rounding mode. 0 = Disabled 1 = Enabled
8–9	reserved	write 0
10	PCSTKE	PC stack interrupt enable. 0 = Disabled 1 = Enabled

Table A-11. ICNTL Register Bit Definitions (Continued)

Bit	Name	Description
11	EMUCNTE	Emulator cycle counter interrupt enable. 0 = Disabled 1 = Enabled
12–15	reserved	write 0

Indirect Jump Page (IJPG) Register

The IJPG register is a non-memory-mapped, register group 3 register (REG3). The reset value for this register is 0x00. For information on using this register, see "Indirect Jump Page (IJPG) Register" on page 3-15.

PC Stack Page (STACKP) and PC Stack Address (STACKA) Registers

The STACKP and *STACKPA* are non-memory-mapped, register group 1 and 3 registers (REG1, REG3). The PC Stack Page (STACKP) and PC Stack Address (STACKA) registers hold the top entry in the Program Counter (PC) address stack. The upper 8 bits of the address go into STACKP, and the lower 16 bits go into STACKA. The PC stack is 33 levels deep.

On JUMP, CALL, DO…UNTIL (loop), and PUSH PC instructions, the DSP pushes the PC address onto this stack, loading the STACKP and STACKA registers. On RTS/I (return) and POP PC instructions, the DSP pops the STACKP:STACKA address off of this stack, loading the PC register.

For information on using these registers, see "Stacks and Sequencing" on page 3-34.

Loop Stack Page (LPSTACKP) and Loop Stack Address (LPSTACKA) Register

The LPSTACKP and LPSTACKA registers are non-memory-mapped, register group 2 and 3 registers (REG2, REG3). The Loop Stack Page (LPSTACKP) and Loop Stack Address (LPSTACKA) registers hold the top entry in the loop stack. The upper 8 bits of the address go into LPSTACKP, and the lower 16 bits go into LPSTACKA. The loop stack is 8 levels deep.

On DO…UNTIL (loop) instructions, the DSP pushes the end of loop address onto this stack, loading the LPSTACKP and LPSTACKA registers. On PUSH LOOP instructions, the DSP pushes the (explicitly loaded) contents of the LPSTACKP and LPSTACKA registers onto this stack.

At the end of a loop (counter decrements to zero), the DSP pops the LPSTACKP:LPSTACKA address off of this stack, loading the PC register with the next address after the end of the loop. On POP LOOP instructions, the DSP pops the contents of the LPSTACKP and LPSTACKA registers off of this stack.

At the start of a loop the PC (start of loop address) is pushed onto the loop begin stack (STACKP:STACKA registers) and the end of loop address is pushed onto the loop end stack (LPSTACKP:LPSTACKA registers). If it is a counter-based loop (DO…UNTIL CE), the loop count (CNTR register) is pushed onto the counter stack.

For information on using these registers, see "Stacks and Sequencing" on page 3-34.

Counter (CNTR) Register

The CNTR register is a non-memory-mapped, register group 2 register (REG2). The DSP loads the loop counter stack from CNTR on Do/Until or Push Loop instructions. For information on using this register, see "Loops and Sequencing" on page 3-20 and "Stacks and Sequencing" on page 3-34.

Condition Code (CCODE) Register

The CCODE register is a non-memory-mapped, register group 3 register (REG3). Using the CCODE register, conditional instructions may base execution on a comparison of the CCODE value (user-selected) and the SWCOND condition (DSP status). The CCODE register holds a value between 0x0 and 0xF, which the instruction tests against when the conditional instruction uses SWCOND or NOT SWCOND. Note that the CCODE register has a one cycle effect latency. The CCODE register bits hold the following data:

CCODE[7:0] holds the same value as FLAGS[15:8]
CCODE[15:8] is RESERVED

Refer to Table 6-2 on page 6-14 for more information regarding the FLAGS register.

Cache Control (CACTL) Register

The CACTL register is a register-memory-mapped register at address 0x0F. The reset value for this register is b#101n nnnn.

Table A-12. CACTL Registers Bit Definitions

Bit	Name	Description
4-0	Reserved	Reserved
5	CDE	Enable caching of instructions that conflict with DMDAs (=1, enabled on reset)
6	CFZ	Cache freeze (fills disabled while set)
7	CPE	Cache enable (=1, enabled on reset)

Data Address Generator Registers

The DSP's Data Address Generator (DAG) registers hold data addresses, modify values, and circular buffer configurations. Using these registers, the DAGs can automatically increment addressing for ranges of data locations (a buffer).

Table A-13. Data Address Generator Registers

Register	Initialization After Reset
"Index Registers (Ix)" on page A-24	Undefined
"Modify Registers (Mx)" on page A-24	Undefined
"Length and Base (Lx,Bx) Registers" on page A-25	Undefined
"Data Memory Page (DMPGx) Register" on page A-25	0x00

Index Registers (Ix)

The `Ix` register are non-memory-mapped, Register Group 1 and 2 registers (REG1 and REG2). The Data Address Generators store addresses in Index registers (`I0-I3` for DAG1 and `I4-I7` for DAG2). An Index register holds an address and acts as a pointer to memory. For more information, see "DAG Operations" on page 4-9.

Modify Registers (Mx)

The `Mx` registers are non-memory-mapped, Register Group 1 and 2 registers (REG1 and REG2). The Data Address Generators update stored addresses using Modify registers (`M0-M3` for DAG1 and `M4-M7` for DAG2). A Modify register provides the increment or step size by which an Index register is pre- or post-modified during a register move. For more information, see "DAG Operations" on page 4-9.

Length and Base (Lx,Bx) Registers

The Length registers are non-memory-mapped, Register Group 1 and 2 registers (REG1 and REG2). The Base registers are memory-mapped in register-memory at addresses: B0=0x00 through B7=0x07.

The Data Address Generators control circular buffering operations with Length and Base registers (`L0-L3` and `B0-B3` for DAG1 and `L4-L7` and `B4-B7` for DAG2). Length and Base registers setup the range of addresses and the starting address for a circular buffer. For more information, see "DAG Operations" on page 4-9.

Data Memory Page (DMPGx) Register

The `DMPGx` registers are a non-memory-mapped, register group 3 registers (REG3). The reset value for these registers is 0x00. For information on using this registers, see "DAG Page Registers (DMPGx)" on page 4-6.

Memory Interface Registers

The DSP's memory interface registers set up page access to I/O memory and provide an interface between the 24-bit and 16-bit data buses.

Table A-14. Memory Interface Registers

Register	Initialization After Reset
"PM Bus Exchange (PX) Register" on page A-26	Undefined
"I/O Memory Page (IOPG) Register" on page A-26	0x00

PM Bus Exchange (PX) Register

This is a non-memory-mapped, register group 3 register (REG3). The PM Bus Exchange (PX) register permits data to flow between the PM and DM data buses. For more information on PX register usage, see "Internal Data Bus Exchange" on page 5-5.

I/O Memory Page (IOPG) Register

This is a non-memory-mapped, register group 3 register (REG3). The reset value for this register is 0x00.

Register and Bit #Defines File

The following example definitions file is for the items that are common to all ADSP-219x DSPs. For the most current definitions file, programs should use the version of this file that comes with the software development tools. The version of the file that appears here is included as a guide only.

```
/* -----------------------------------------------------------------------------
def2192_core.h - SYSTEM REGISTER BIT & ADDRESS DEFINITIONS FOR ADSP-219x DSPs

Created November 21, 2000. Copyright Analog Devices, Inc.

   Changes: Added ADSP-219x common items to def2192_core.h file


The def2192_core.h file defines ADSP-219x DSP family common symbolic names; for
names that are unique to particular ADSP-219x family DSPs, see that DSP's
definitions file (such as the def2191.h file) instead. This include file
(def2192_core.h) contains a list of macro "defines" that let programs use symbolic
names for the following ADSP-219x facilities:

- system register bit definitions
- system register map

These registers use the REG() command.

Here is an example use:

ax0 = 0x0800;
REG(B0) = ax0;                >>> this uses the define for the B0 register's address

-----------------------------------------------------------------------------*/
#ifndef __DEF2192_core_H_
#define __DEF2192_core_H_
/*---------------------------------------------------------------------------*/

#define  B0       0x00  /* Base Register0 */
#define  B1       0x01  /* Base Register1 */
#define  B2       0x02  /* Base Register2 */
#define  B3       0x03  /* Base Register3 */
#define  B4       0x04  /* Base Register4 */
#define  B5       0x05  /* Base Register5 */
#define  B6       0x06  /* Base Register6 */
#define  B7       0x07  /* Base Register7 */
#define  DMAPAGE  0x0C  /* DMA Page Register */
#define  CACTL    0x0F  /* Cache Control Register */
#define  STCTL0   0x10  /* FIFO0 Transmit Control Register */
#define  SRCTL0   0x11  /* FIFO0 Receive Control Register */
#define  TX0      0x12  /* FIFO0 Transmit Data (TX) register */
#define  RX0      0x13  /* FIFO0 Receive Data (RX) register */
#define  STCTL1   0x20  /* FIFO1 Transmit Control Register */
#define  SRCTL1   0x21  /* FIFO1 Receive Control Register */
#define  TX1      0x22  /* FIFO1 Transmit Data (TX) register */
```

```
#define  RX1      0x23  /* FIFO1 Receive Data (RX) register */
#define  TPERIOD  0x30  /* Timer Period Register */
#define  TCOUNT   0x31  /* Timer Counter Register */
#define  TSCALE   0x32  /* Timer Scaling Register */
#define  TSCALECNT   0x33  /* Timer Scale Count Register */
#define  FLAGS       0x34  /* Flags Register */
#define  MASTADDR    0x44  /* DMA Address, DSP Master DMA */
#define  MASTNXTADDR 0x45  /* DMA Next Address, DSP Master DMA */
#define  MASTCNT     0x46  /* DMA Count, DSP Master DMA */
#define  MASTCURCNT  0x47  /* DMA Current Count, DSP Master DMA */
#define  TX0ADDR     0x48  /* DMA Address, Fifo0 Transmit */
#define  TX0NXTADDR  0x49  /* DMA Next Address, Fifo0 Transmit */
#define  TX0CNT      0x4A  /* DMA Count, Fifo0 Transmit */
#define  TX0CURCNT   0x4B  /* DMA Current Count, Fifo0 Transmit */
#define  RX0ADDR     0x4C  /* DMA Address, Fifo0 Receive */
#define  RX0NXTADDR  0x4D  /* DMA Next Address, Fifo0 Receive */
#define  RX0CNT      0x4E  /* DMA Count, Fifo0 Receive */
#define  RX0CURCNT   0x4F  /* DMA Current Count, Fifo0 Receive */
#define  TX1ADDR     0x50  /* DMA Address, Fifo1 Transmit */
#define  TX1NXTADDR  0x51  /* DMA Next Address, Fifo1 Transmit */
#define  TX1CNT      0x52  /* DMA Count, Fifo1 Transmit */
#define  TX1CURCNT   0x53  /* DMA Current Count, Fifo1 Transmit */
#define  RX1ADDR     0x54  /* DMA Address, Fifo1 Receive */
#define  RX1NXTADDR  0x55  /* DMA Next Address, Fifo1 Receive */
#define  RX1CNT      0x56  /* DMA Count, Fifo1 Receive */
#define  RX1CURCNT   0x57  /* DMA Current Count, Fifo1 Receive */
#define  DBGCTRL     0x60  /* Test and Emulation Debug Control Register */
#define  DBGSTAT     0x61  /* Test and Emulation Debug Status Register */
#define  CNT0        0x62  /* Cycle Counter 0 Register (lsb) */
#define  CNT1        0x63  /* Cycle Counter 1 Register */
#define  CNT2        0x64  /* Cycle Counter 2 Register */
#define  CNT3        0x65  /* Cycle Counter 3 Register (msb) */


/************* SRCTLx and STCTLx Bit definitions ************/

#define SPEN   0   /* AC'97 FIFO Connection Enable */
#define SSEL3  7   /* AC'97 Slot Select */
#define SSEL2  6   /* AC'97 Slot Select */
#define SSEL1  5   /* AC'97 Slot Select */
#define SSEL0  4   /* AC'97 Slot Select */
#define FIP2  10   /* AC'97 FIFO Interrupt Position */
#define FIP1   9   /* AC'97 FIFO Interrupt Position */
#define FIP0   8   /* AC'97 FIFO Interrupt Position */
#define SDEN  11   /* AC'97 Port DMA Enable */
#define FULL  13   /* FIFO Full, read-only) */
#define EMPTY 14   /* FIFO Empty, read-only) */
#define FLOW  15   /* FIFO Over/Underflow, sticky, write-one-clear) */
/**********************************************************/

#endif
```

B ADSP-2192 DSP PERIPHERAL REGISTERS

Overview

The DSP 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 (for I/O processor registers) memory-mapped addresses. Information on each type of register is available at the following locations:

- "Core Status Registers" on page A-8
- "Computational Unit Registers" on page A-16
- "Program Sequencer Registers" on page A-19
- "Data Address Generator Registers" on page A-26
- "Peripheral Registers" on page B-2

When writing DSP programs, it is often necessary to set, clear, or test bits in the DSP's registers. While these bit operations can 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 header files that define these bit and register definitions (def2192_IO.h, def2192_PCI.h, def2192_USB.h, def2192-12.h, and def219x.h). Note that the def2192-12.h file also contains the definitions from the IO, PCI, and USB header files.

A sample of def219x.h is shown in “Register and Bit #Defines File” on page A-27, and a sample of def2192-12.h is shown in “Register and Bit #Defines File” on page B-95.

Many registers have reserved bits. When writing to a register, programs may clear (write zero to) the register’s reserved bits only.

Peripheral Registers

There are three groups of registers for the ADSP-2192:

- “ADSP-219x DSP Core Registers” on page A-1
- “ADSP-2192 System Control Registers” on page B-6
- “ADSP-2192 Peripheral Device Control Registers” on page B-11

This appendix describes system control registers and peripheral device control registers. A general description of DSP peripheral architecture, which follows, provides an overview of peripheral registers.

DSP Peripherals Architecture

Figure B-1 shows the DSP's on-chip peripherals, which include the Host port (PCI or USB), AC'97 port, JTAG test and emulation port, flags, and interrupt controller.

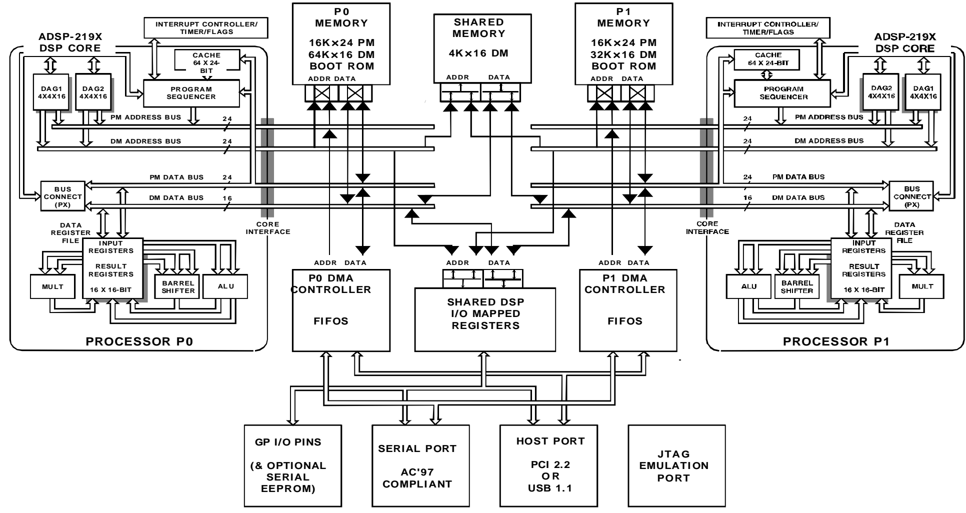


Figure B-1. ADSP-2192 Dual-Core DSP Block Diagram

The ADSP-2192 can respond to up to fourteen interrupts at any given time. A list of these interrupts can be found in the table "Interrupt Vectors for an ADSP-2192 DSP Core" on page 6-13.

The AC'97 codec port on the ADSP-2192 provides a complete synchronous, full-duplex serial interface. This interface completely supports the AC'97 standard.

The ADSP-2192 provides up to eight general-purpose I/O pins, which are programmable as either inputs or outputs. These pins are dedicated general-purpose programmable flag pins.

Peripheral Device Register Groups

The registers that control FIFO DMA transfers are accessible only from within the DSP. They are defined as part of the Core Register Space.

Summary

Each of the DSPs integrated within the ADSP-2192 and the interfaces (PCI, USB Sub-ISA, Cardbus) needs to be capable of controlling and monitoring a variety of registers external to the DSP core. This section describes how the DSPs access these Peripheral Device Control (PDC) registers. The operation of the Peripheral Device Control (PDC) Bus that connects the DSPs and Interfaces to the PDC Registers is also described in this section.

Writes to AC'97 codec registers are posted, but only one may complete per AC'97 frame. Up to two writes may be pending at any one time. The first write completes with zero PDC wait states. A second write launched immediately after the first incurs PDC wait states equivalent to a few AC'97 BITCLKs. A third write in a row blocks for an entire AC'97 frame. Use the Frame interrupt to time AC'97 codec writes out to one per frame, assuring that they will all complete with zero wait states.

Reads from AC'97 codec registers must always wait for the data to be returned. A read must also wait for any pending AC'97 codec register writes to complete before it can begin. In the best case, a read takes one full AC'97 frame plus another three AC'97 slots (25.39 µs, or approximately 3,744 DSP cycles). This is also the typical case when the AC'97 Frame Interrupt is used to time the Read.

The worst case AC'97 read time is four frames plus three slots (87.89 µs, or approximately 12,960 DSP cycles). This occurs only when there are already two AC'97 codec register writes pending just after the start of a frame.

Most AC'97 codec registers may be shadowed, and actual reads should be rare.

Example

In the worst case, DSP core P1 posts two AC'97 codec register writes just after the start of a new Frame. DSP core P0 immediately follows with a read to an AC'97 codec register. DSP core P0 will be unable to compute, DMA, or interrupt for 87.89 µs. DSP core P1 can compute with data in its own memory, but cannot communicate with DSP core P0 or access any PDC bus register for 87.89 µs. The external bus interface can communicate with DSP core P1, but cannot communicate with DSP core P0 or access any PDC bus register for 87.89 µs. In the state, the entire ADSP-2192 system is highly constrained.

ADSP-2192 System Control Registers

The following tables show the System Control Registers in each DSP core.

Table B-1. ADSP-2192 System Control Registers

Address	Register	Function
00	B0	Base Register0
01	B1	Base Register1
02	B2	Base Register2
03	B3	Base Register3
04	B4	Base Register4
05	B5	Base Register5
06	B6	Base Register6
07	B7	Base Register7
08 - 0B		Reserved
0C	DMAPAGE	DMA Page Register
0D - 0E		Reserved
0F	CACTL	Cache Control Register
10	STCTL0	FIFO0 Transmit Control Register
11	SRCTL0	FIFO0 Receive Control Register
12	TX0	FIFO0 Transmit Data (TX) register
13	RX0	FIFO0 Receive Data (RX) register

Table B-1. ADSP-2192 System Control Registers (Continued)

Address	Register	Function
14 - 1F		Reserved
20	STCTL1	FIFO1 Transmit Control Register
21	SRCTL1	FIFO1 Receive Control Register
22	TX1	FIFO1 Transmit Data (TX) register
23	RX1	FIFO1 Receive Data (RX) register
24 - 2F		Reserved
30	TPERIOD	Timer Period Register
31	TCOUNT	Timer Counter Register
32	TSCALE	Timer Scaling Register
33	TSCALECNT	Timer Scale Count Register
34	FLAGS	Flags Register
35 - 3F		Reserved
40 - 43		Reserved
44	MASTADDR	DMA Address, DSP Master DMA
45	MASTNXTADDR	DMA Next Address, DSP Master DMA
46	MASTCNT	DMA Count, DSP Master DMA
47	MASTCURCNT	DMA Current Count, DSP Master DMA
48	TX0ADDR	DMA Address, FIFO0 Transmit
49	TX0NXTADDR	DMA Next Address, FIFO0 Transmit

Table B-1. ADSP-2192 System Control Registers (Continued)

Address	Register	Function
4A	TX0CNT	DMA Count, FIFO0 Transmit
4B	TX0CURCNT	DMA Current Count, FIFO0 Transmit
4C	RX0ADDR	DMA Address, FIFO0 Receive
4D	RX0NXTADDR	DMA Next Address, FIFO0 Receive
4E	RX0CNT	DMA Count, FIFO0 Receive
4F	RX0CURCNT	DMA Current Count, FIFO0 Receive
50	TX1ADDR	DMA Address, FIFO1 Transmit
51	TX1NXTADDR	DMA Next Address, FIFO1 Transmit
52	TX1CNT	DMA Count, FIFO1 Transmit
53	TX1CURCNT	DMA Current Count, FIFO1 Transmit
54	RX1ADDR	DMA Address, FIFO1 Receive
55	RX1NXTADDR	DMA Next Address, FIFO1 Receive
56	RX1CNT	DMA Count, FIFO1 Receive
57	RX1CURCNT	DMA Current Count, FIFO1 Receive
58-5F		Reserved
60	DBGCTRL	Test and Emulation Debug Control Register
61	DBGSTAT	Test and Emulation Debug Status Register
62	CNT0	Cycle Counter 0 Register (LSB)
63	CNT1	Cycle Counter 1 Register

Table B-1. ADSP-2192 System Control Registers (Continued)

Address	Register	Function
64	CNT2	Cycle Counter 2 Register
65	CNT3	Cycle Counter 3 Register (MSB)
66-FF		Reserved

STCTLx FIFO Transmit Control Register

These include the STCTL0 and STCTL1 registers in each DSP core.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FLOW	EMPTY	FULL	LOOP	SDEN	FIP2	FIP1	FIP0	SSEL3	SSEL2	SSEL1	SSEL0	DSP	FLSH	SDEN	SPEN

SRCTLx FIFO Receive Control Register

These include the SRCTL0 and SRCTL1 registers in each DSP core.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FLOW	EMPTY	FULL	LOOP	SDEN	FIP2	FIP1	FIP0	SSEL3	SSEL2	SSEL1	SSEL0	DSP	FLSH	SDEN	SPEN

xxxADDR DMA Address Register

This group of registers include Rx0ADDR, Rx1ADDR, Tx0ADDR, Tx1ADDR, and MASTADDR registers for each DSP core. Each register is a 16-bit register containing a 16-bit word.

xxxNXTADDR DMA Next Address Register

This group of registers include Rx0NXTADDR, Rx1NXTADDR, Tx0NXTADDR, Tx1NXTADDR, and MASTNXTADDR for each DSP core. Each register is a 16-bit register containing a 16-bit word.

xxxCNT DMA Count Register

This group of registers include Rx0CNT, Rx1CNT, Tx0CNT, Tx1CNT, and MASTCNT for each DSP core. Each register is a 16-bit register containing a 16-bit word.

xxxCURCNT DMA Current Count Register

This group of registers include Rx0CURCNT, Rx1CURCNT, Tx0CURCNT, Tx1CURCNT, and MASTCURCNT for each DSP core. Each register is a 16-bit register containing a 16-bit word.

ADSP-2192 Peripheral Device Control Registers

The following tables show the Peripheral Device Control Registers accessible by both DSP cores and the PCI and USB interfaces.

The following is a summary of the various classes of I/O registers and their organization within DSP I/O pages.

Addresses are 8-bit values. The Page is also an 8-bit value.

Table B-2. Register Group Descriptions

Page	Addresses	Descriptions	Access permitted by	Refer to
0x00	0x00-0x0F	ADSP-2192 Chip Control Registers	DSP / PCI / USB	page B-13
	0x10-0x1F	General-purpose I/O (GPIO) Control Registers	DSP / PCI / USB	page B-24
	0x20-0x2F	Host Mailbox Registers	DSP / PCI / USB	page B-30
	0x30	EEPROM Register	DSP / PCI / USB	page B-28
	0xA0-0xBF	JTAG ID Registers	DSP Only	page B-32
	0xC0-0xFF	AC'97 Controller Registers	DSP / PCI / USB	page B-41
0x01	0x00-0x2D	CardBus Function Event Registers	DSP / PCI	page B-32
0x02-0x03		Reserved		
0x04	0x00-0x7E	AC'97 Codec Register Space, Primary Codec 0	DSP / PCI / USB	page B-45

Table B-2. Register Group Descriptions (Continued)

Page	Addresses	Descriptions	Access permitted by	Refer to
0x05	0x00-0x7E	AC'97 Codec Register Space, Secondary Codec 1	DSP / PCI / USB	page B-45
0x06	0x00-0x7E	AC'97 Codec Register Space, Secondary Codec 2	DSP / PCI / USB	page B-46
0x07	Reserved			
0x08	0x00-0x7F	DMA Address, Count Registers	DSP / PCI	page B-46
	0x80-0x87	DMA Control Registers	DSP / PCI	page B-46
	0x88-0x8A	PCI Interrupt, Control Registers	DSP / PCI	page B-47
0x09	0x00-0xFF	PCI Configuration Register Space, Function 0	DSP[1] / PCI	page B-56
0x0A	0x00-0xFF	PCI Configuration Register Space, Function 1	DSP[1] / PCI	page B-58
0x0B	0x00-0xFF	PCI Configuration Register Space, Function 2	DSP[1] / PCI	page B-59
0x0C	0x00-0x4F	USB DSP Registers	DSP / USB	page B-71
0x0D-0xFF	Reserved			

1 PCI configuration spaces should be accessed only by the DSP, and only during the boot process. After the PCI interface has been configured, bit 2 (ConfRdy) of the PCI_CFGCTL register should be set by the DSP. This allows the PCI interface access to these registers while at the same time denying the DSP access.

ADSP-2192 Chip Control Registers

The chip control registers provide support for the following:

- General status for the chip as a whole
- Power-down operations
- Other control functions

The following table lists the PDC register space. The register addresses from PCI space, USB space, and DSP I/O space are listed.

Table B-3. ADSP-2192 Chip Control Registers

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
SYSCON	Chip Mode/Status	0x01-0x00	0x01-0x00	0x00	0x00
PWRCFG0	Function 0 Power Management	0x03-0x02	0x03-0x02	0x00	0x02
PWRCFG1	Function 1 Power Management	0x05-0x04	0x05-0x04	0x00	0x04
PWRCFG2	Function 2 Power Management	0x07-0x06	0x07-0x06	0x00	0x06
PWRP0	DSP 0 Interrupt/Powerdown	0x09-0x08	0x09-0x08	0x00	0x08
PWRP1	DSP 1 Interrupt/Powerdown	0x0B-0x0A	0x0B-0x0A	0x00	0x0A
PLLCTL	DSP PLL Control	0x0D-0x0C	0x0D-0x0C	0x00	0x0C

Chip Control (SYSCON) Registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PCI RST	VAUX	PCI_5V	B5V	BUS<1:0>		CRST<1:0>		REGD	VXPD	VXPW	ACVX	XON	RDIS	Reserved	RST

Table B-4. SYSCON Register Bit Descriptions

Bit Position	Bit Name	Description
0	RST	Soft Chip Reset. A write of 1 causes a soft reset to the ADSP-2192. A write of 0 has no effect. Always reads 0. Soft Reset affects the DSPs and the GPIOs. Soft Reset does not affect the PCI, USB, Mailboxes, AC'97, or EEPROM. Note that the DSP memory pipeline (last 2 writes per bank) is lost upon reset. If desired, it may be flushed by three writes in a row to the same location. Note: This bit resets to zero.
1	Reserved	

Table B-4. SYSCON Register Bit Descriptions (Continued)

Bit Position	Bit Name	Description
2	RDIS	Reset Disable. When 1, disables a PCI/ISA/CBUS bus reset from affecting any portions of the ADSP-2192 except the bus interface itself. When 0 (default), a bus reset causes the DSPs and AC'97 subsystem to be reset. Note: If RDIS is set, the DSP can detect that the bus is in reset by the PCIRST bit in the CMSR register. Un-masked Bus Reset affects the DSPs, the GPIOs, the AC'97, and the PCI/USB interface. Un-masked Bus Reset does not affect the Mailboxes or EEPROM. Note that the DSP memory pipeline (last 2 writes per bank) is lost upon reset. If desired, it may be flushed by three writes in a row to the same location. Note: This bit resets to zero.
3	XON	XTAL Force On. When 1, causes the XTAL oscillator to run even if all other subsystems are powered down. This permits access to the on-chip control registers when the part is powered down. If the chip and the XTAL oscillator are powered off, attempting to write PDC registers including this one will result in powering up the XTAL and setting the XON bit. The write will succeed, after a delay for the oscillator to stabilize. Subsequent writes or reads should not be attempted until the oscillator has stabilized, about 8K clocks or 333us. When 0, the XTAL oscillator stops whenever it is not needed by any on-chip subsystem. Note: This bit resets to zero.

Table B-4. SYSCON Register Bit Descriptions (Continued)

Bit Position	Bit Name	Description
4	ACVX	AC’97 External Devices Vaux Powered. Controls the AC’97 interface during D3cold ($\overline{RST}$ asserted). 0 = Disable the interface (drive 0, disable all inputs). This is used if external AC’97 devices are NOT powered during d3cold, and protects the ADSP-2192 from floating inputs and from outputs driving input clamps on an external device. (default) 1 = Interface enabled during $\overline{RST}$. Note: This bit resets to zero.
5	Reserved	Reserved
6	Reserved	Reserved
7	REGD	2.5V Regulator Control Disable. Disables the on-chip 2.5V Regulator controller when the 2.5V (IVDD) supply is derived from an external regulator (e.g. in USB and Mini-PCI applications). 0 = On-Chip 2.5V Regulator Control Enabled. (default) 1 = On-Chip 2.5V Regulator Control Disabled. Note: This bit resets to zero.
9:8	CRST<1:0>	Chip Reset Source. Indicates the source of the last reset to the chip (Read-Only) 00 = Power-On Reset 01 = Reserved. 10 = PCI/ISA/CBUS/USB bus interface hard reset 11 = Soft Reset from the CMSR:RST bit Note: the fifth possible reset source, DIP Soft Reset, is indicated by DIP1/2:RD = 1. Each DSP must check its DIP<n>:RD bit and clear it to zero upon reset.

Table B-4. SYSCON Register Bit Descriptions (Continued)

Bit Position	Bit Name	Description
11:10	BUS<1:0>	Bus Mode. Mode Pin status. Sampled at Power-On Reset (Read-Only). 00= PCI 01= CardBus 10= USB 11= Sub-ISA
12	B5V	**AC'97 5V level.** 1= If the AC'97 interface is powered from nominal 5V; 0 if nominal 3.3V.
13	PCI 5V	**PCI 5V level.** 1= If the PCI/ISA/CBUS interface is powered from nominal 5V; 0 if nominal 3.3V. Monitors the level of the `PCIVDD` pins (Read Only).
14	Vaux	Vaux Present. 1= If Vaux is currently powered (Read-Only).
15	PCIRST	PCI Reset. 0= If PCI/CBUS/ISA $\overline{\text{RST}}$ is asserted (which may indicate `D3Cold` powerdown state). Note: This bit resets to one.

Power Management Functions

Power management registers share the same bit specifications. Each register corresponds to one of the PCI functions:

15	14	13:9	8	7	6	5	4	3	2	1	0
PME	SPME	Reserved	PME EN	Reserved	GPME	APME	Reserved			PWRST[1:0]	

All bits in this register reset to zero.

Table B-5. Bit Descriptions for PWRCFG0, PWRCFG1, and PWRCFG2 Registers

Bit Position	Bit Name	Description
1:0	PWRST<1:0>	PCI Function Power State. Reports this function's PCI Power Management state from its PMCSR register in PCI Configuration Space. (Read Only)
4:2	Reserved	
5	APME	AC'97 Power Management Event Enable. 1= Enables setting this function's PME bit upon an AC'97 interrupt/wake event. (Read/Write)
6	GPME	GPIO Power Management Event Enable. 1= Enables setting this function's PME bit upon a GPIO Wakeup event. (Read/Write)
7	Reserved	Reserved
8	PME_EN	Power Management Event Enable. 1= PME_EN bit is set in this function's PMCSR register in PCI Configuration space.
13:9	Reserved	

Table B-5. Bit Descriptions for PWRCFG0, PWRCFG1, and PWRCFG2 Registers (Continued)

Bit Position	Bit Name	Description
14	SPME	Power Management Event (Set). A write of 1 to this bit sets the PME bit for this function. A write of 0 has no effect. Always reads 0.
15	PME	Power Management Event (Status/Clear). 1= A power management event has been detected for this function. This is an alias of the PME bit in the Power Management Control/Status Register in PCI Configuration Space for this function. A write of 1 to this bit clears PME. 0= A write of 0 has no effect.

DSP Powerdown (PWRPx) Registers

These two registers share the following bit layout. One register corresponds to each DSP.

All bits in this register reset to zero.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RINT	GINT	AINT	PMINT	RIEN	GIEN	AIEN	PMIEN	RWE	GWE	AWE	PMWE	FIEN	RSTD	PU	PD

Table B-6. DSP Interrupt/Powerdown (PWRPx) Register Bit Descriptions

Bit Position	Bit Name	Description
0	PD	DSP PowerDown. When written to a 1, causes the DSP to power down (enter its power-down handler). Can also be used to abort a power-up: if the DSP is in the power-down handler after executing an IDLE, writing a 1 will cause the DSP to immediately re-enter the PowerDown interrupt handler after it executes the RTI. When read, PD=1 indicates that this DSP is powered down: either (a) it is in the powerdown handler and has executed an IDLE instruction), and/or (b) the DSP Clock Generator (PLL) is not running and stable. When both DSPs are powered down, the DSP Clock Generator is powered down, and automatically restarts when either DSP wakes up. Note: DSP memory cannot be accessed via PCI or USB when the DSP is powered down. There is a delay after powering up the DSPs with the PU bit during which memory reads must not be performed, because the XTAL or the DSP PLL is not yet running and stable. After powering up by writing a 1 to the PU bit, the PD bit must be polled until it becomes 0, after which the clock generator will be running and it is safe to access DSP memory again.
1	PU	DSP PowerUp. When written to a 1, causes the DSP to power up (exit the IDLE within its power-down handler). Can also be used to abort a powerdown: when written to 1 while the DSP is within its powerdown handler prior to the IDLE, writing a 1 will cause execution to immediately continue through the IDLE without stopping clocks. When read, PU=1 indicates that this DSP is in the power-down interrupt handler, whether or not it has executed the powerdown IDLE.

Table B-6. DSP Interrupt/Powerdown (PWRPx) Register Bit Descriptions (Continued)

Bit Position	Bit Name	Description
2	RSTD	DSP Soft Reset When written to a 1, causes a soft reset to this DSP. Retains a 1 until cleared by writing to a 0. If the DSP core is powered down, it must be powered up first (DSP:PU bit written to 1) before resetting.
3	FIEN	DSP Interrupt Enable: AC'97 Frame When 1, enables an AC'97 Frame interrupt (IMASK bit 15) to this DSP from the AC'97 Interface. If 0, no interrupt is signalled (Read/Write). The actual interrupt occurs once per AC'97 Frame, at the second bit of Slot 12.
4	PMWE	Power Management Wakeup Enable. When 1, enables waking the respective DSP on a Power Management State Change event (Read/Write).
5	AWE	DSP Wakeup Enable: GPIO Interrupt, AC'97 Interrupt. When 1, enables this DSP to wake from powerdown upon an event from the indicated source. (Read/Write).
6	GWE	DSP Wakeup Enable: GPIO Interrupt, AC'97 Interrupt. When 1, enables this DSP to wake from powerdown upon an event from the indicated source. (Read/Write).
7	RWE	DSP Wakeup Enable: GPIO Interrupt, AC'97 Interrupt. When 1, enables this DSP to wake from powerdown upon an event from the indicated source. (Read/Write).
8	PMIEN	Power Management Interrupt Enable. When 1, enables interrupting the respective DSP on a Power Management State Change event. (The interrupt level is the same as used for GPIO and AC'97 interrupt) (Read / Write).

Table B-6. DSP Interrupt/Powerdown (PWRPx) Register Bit Descriptions (Continued)

Bit Position	Bit Name	Description
9	AIEN	DSP Interrupt Enable: AC'97 Interrupt. When 1, enables an IO interrupt to this DSP from the AC'97 port. If 0, no interrupt will be signalled and the corresponding Interrupt Pending bit will not be set upon an event. (Read/Write).
10	GIEN	DSP Interrupt Enable: GPIO Interrupt. When 1, enables an IO interrupt to this DSP from the GPIO. If 0, no interrupt will be signalled and the corresponding Interrupt Pending bit will not be set upon an event. (Read/Write).
11	Reserved	
12	PMINT	Power Management Interrupt Pending. When 1, indicates an interrupt is pending for the respective DSP from a Power Management State Change event. A write of 1 clears this interrupt flag. A write of 0 has no effect.
13	AINT	When 1, an IO interrupt (IMASK bit 6) to this DSP is pending from the AC'97 port. A write of 1 clears this interrupt flag. A write of 0 has no effect. The AC'97 port should be cleared prior to clearing this interrupt flag, or it may be re-triggered. Similarly, this interrupt flag must be cleared prior to executing an RTI from the DSP interrupt handler routine, or the DSP may immediately take another interrupt.
14	GINT	When 1, an IO interrupt (IMASK bit 6) to this DSP is pending from the GPIO. A write of 1 clears this interrupt flag. A write of 0 has no effect. The GPIO should be cleared first (e.g., clearing a GPIO Status Bit) prior to clearing this interrupt flag, or it may be re-triggered. Similarly, this interrupt flag must be cleared prior to executing an RTI from the DSP interrupt handler routine, or the DSP may immediately take another interrupt.
15	Reserved	

DSP PLL Control (PLLCTL) Register

The DSP PLL control register controls the frequencies of the PLL (Phase Locked Loop) clock generator. Do not write to this register unless the PLL is powered down.

The register is controlled by an Adjust bit. When the Adjust bit is zero, default values for the settings in that segment are used by the PLL. These default values are also returned upon a register read. The default values are subject to change.

Writing this register to 0 resets the register to its factory defaults.

$F_{out} = 6 * F_{in} = 98.304$ MHz

$6 * F_{in} = 147.456$ MHz

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CselR	CselC	Cboost	CAdj	DPLLK		DPLLN		DPLLM				DselR	DselC	Dboost	DAdj

General-purpose I/O (GPIO) Control Registers

Eight pins support general-purpose I/O to registers that control them.

Table B-7 lists the Peripheral Device Control Register Space for GPIO Control registers:

Table B-7. GPIO Control Registers

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
GPIOCFG	GPIO Configuration Direction Control (1 = input, 0 = output)	0x010	0x0010	0x00	0x10
GPIOPOL	GPIO Polarity Inputs: 0 = active high 1 = active low. Outputs: 0 = CMOS 1 = Open Drain	0x012	0x0012		0x12
GPIOSTKY	GPIO Sticky: 1 = sticky, 0 = not sticky	0x014	0x0014	0x00	0x14
GPIOWCTL	GPIO Wake Control 1 = wake-up enabled (requires sticky set)	0x016	0x0016	0x00	0x16
GPIOSTAT	GPIO Status Read = Pin state Write: 0 = clear sticky status, 1 = no effect	0x018	0x0018	0x00	0x18
GPIOCTL	GPIO Control (w), Init (r) Read = Power-on state; Write: Set state of output pins	0x01A	0x001A	0x00	0x1A

Table B-7. GPIO Control Registers (Continued)

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
GPIOPUP	GPIO Pullup Pull-up enable (if input): 1 = enable, 0 = Hi-Z	0x01C	0x001C	0x00	0x1C
GPIOPDN	GPIO Pulldown Pull-down enable (if input): 1 = enable, 0 = Hi-Z	0x01E	0x001E	0x00	0x1E

GPIO Configuration (GPIOCFG) Register

This register resets to 0x7F.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								GCFG[7:0]							

GPIO Polarity (GPIOPOL) Register

This register resets to 0xFF.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								GPOL[7:0]							

GPIO Sticky (GPIOSTKY) Register

This register resets to zero.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								GSTKY							

GPIO Wakeup Control (GPIOWAKECTL) Register

This register resets to zero.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								GWAK[7:0]							

GPIO Status (GPIOSTAT) Register

This register resets to 0xFF.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								GSTAT[7:0]							

GPIO Control (GPIOCTL) Register

This register resets to 0x7F.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								GCTL[7:0]							

GPIO Pullup (GPIOPUP) Register

This register resets to 0xFF.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								GPU[7:0]							

GPIO Pulldown (GPIOPDN) Register

This register resets to zero.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								GPD[7:0]							

EEPROM I/O Control/Status (SPROMCTL) Register

The EEPROM register controls access to the serial EEPROM. Table B-8 lists the Peripheral Device Control Register Space for EEPROM Control Register.

Table B-8. SPROMCTL Control Register

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
SPROMCTL	EEPROM I/O Control/Status Controls the direction and status for SEN, SCK, SDA pins.	0x30	0x30	0x00	0x30

This register is reset by any of the following:

- Power-On Reset
- $\overline{\text{SYSRST}}$ asserted
- Soft Reset using the PCC:RST bit
- PCI $\overline{\text{RST}}$ asserted when the AC97LCTL:DSPR bit is 0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCKI	SENI	SDAI	Reserved					SCK	SEN	SDA	Reserved				

Table B-9. SPROMCTL Register Bit Descriptions

Bit Position	Bit Name	Description
4:0	Reserved	
5	SDA	SDA pin status. Default = 0. **Note:** *This bit resets to zero.*
6	SEN	SEN pin status. Default = 0 (output driving 0). **Note:** *This bit resets to zero.*
7	SCK	SCK pin status. Default = 0 (output driving 0) **Note:** *This bit resets to zero.*
12:8	Reserved	
13	SDAI	SDA pin input enable. 1=input Default=1 (input) 0=output. **Note:** *This bit resets to one.*
14	SENI	SEN pin input enable. 1=input 0=output (default) **Note:** *This bit resets to zero.*
15	SCKI	SCK pin input enable. 1=input 0=output (default) **Note:** *This bit resets to zero.*

Host Mailbox Registers

The Host Mailbox registers control communication between the DSP and host (PCI host or USB Host), depending on which one is turned on. Only one can be active at time.

Overview

DSP Mailbox registers allow you to construct an efficient communications protocol between the PCI device driver and the DSP code. The mailbox functions consist of an InBox0, InBox1, OutBox0, OutBox1, a control register, and a status register.

InBoxes. The incoming mailboxes (`InBox0` and `InBox1`) are 16 bits wide. They may be read or written by the PCI device or the DSP core. PCI writes to the InBoxes may generate DSP interrupts. DSP reads of InBoxes may generate PCI interrupts.

OutBoxes. The outgoing mailboxes (`OutBox0` and `OutBox1`) are 16 bits wide. They may be read or written by the PCI device or the DSP core. DSP writes to the OutBoxes may generate PCI interrupts.

PCI reads of OutBoxes may generate DSP interrupts with special handling. The PC host must perform the following sequence when reading an OutBox: (1) read OutBox, (2) write a 1 to the OutBox `Valid` bit to clear it. (PCI reads of OutBoxes cannot generate interrupts directly, as they would be "read side-effects" which are prohibited by system design considerations in the PCI Specification.)

Control. This register consists of read/write interrupt enable control bits. (denoted R/W).

Status. This register consists of read/write-one-clear status bits (denoted R/WC). A read/write-one-clear bit is cleared when a one is written to it. Writing a zero has no effect.

Table B-10 lists the Peripheral Device Control Register Space for PCI/USB Mailbox registers. For register bit names and descriptions for each register, see the topic "Using DSP and PCI Mailbox Registers" in Chapter 6 Dual DSP Cores.

Table B-10. PCI / USB Mailbox Registers

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
MBXSTAT	Mailbox Status	0x021-0x020	0x0021-0x0020	0x00	0x20
MBXCTL	Mailbox Control	0x023-0x022	0x0023-0x0022	0x00	0x22
MBX_IN0	Incoming Mailbox 0 PCI/USB to DSP mailbox	0x025-0x024	0x0025-0x0024	0x00	0x24
MBX_IN1	Incoming Mailbox 1 PCI/USB to DSP mailbox	0x027-0x026	0x0027-0x0026	0x00	0x26
MBX_OUT0	Outgoing Mailbox 0 DSP to PCI/USB mailbox	0x029-0x028	0x0029-0x0028	0x00	0x28
MBX_OUT1	Outgoing Mailbox 0 DSP to PCI/USB mailbox	0x02B-0x02A	0x002B-0x002A	0x00	0x2A

CardBus Function Event Registers

Of the four function modes, PCI, USB, sub-ISA, and Cardbus, these function event registers are used only in CardBus mode to provide status registers for power management.

In a CardBus system (specified by BUSMODE=01), the operating system handles Power Management in one of two ways. If the O/S is ACPI-compliant, the OS uses the Power Management registers in PCI Configuration space in the normal fashion. If the O/S is an older legacy system, it looks for a set of four 32-bit Function Event registers, one set per card function. (The upper 16 bits of each register are reserved and are implemented as read-only with 0s.) These registers are used only in CardBus systems, and have no effect on operation when the ADSP-2192 is not in CardBus mode (BUSMODE=01). The registers for each of the three functions are largely independent.

The CardBus Function Event registers contain bits similar to those in the PCI Power Management registers:

- Function Event Register GWAKE-E == PCI PME_Status
- Function Event Mask register GWAKE-M == PCI PME_Enable

In addition, the CardBus registers define the following:

- Interrupt Mask INTR M - global mask bit for both interrupt and wakeup ($\overline{\text{PME}}$) signalling
- Wakeup Mask WKUPM - additional PCI PME_Enable
- Event Force bits for interrupt (INTRF) and wakeup (GWAKEF) for software development

The CardBus Function Event registers are defined to inter-operate with the PCI Power Management Control/Status (SYSCON) register as follows:

- PCI updates PME_EN bit.

  CardBus GWAKM and WKUPM take the new value.

- PCI clears PME_Status (writes 1).

  CardBus GWAKE is cleared.

- CardBus updates GWAKM or WKUP.

  No effect in PCI PMCSR.

- CardBus clears GWAKE_E (writes 1).

  PCI PME_Status bit is cleared.

- Power Management event occurs.

  Both GWAKE and PME_Status are set; $\overline{\text{PME}}$ may assert.

- Host Interrupt occurs.

  INTRE is set, $\overline{\text{INTA}}$ may assert.

CSTSCHG Signal

In CardBus systems, power management events are signaled to the host by an active-high signal called CSTSCHG. An external FET or inverter is used with the ADSP-2192's active-low $\overline{\text{PME}}$ signal to create CSTSCHG. In CardBus mode, $\overline{\text{PME}}$ is asserted under the following conditions:

- The function's INTRM master interrupt/wakeup mask bit is 1
- The function's WKUPM master wakeup mask bit is 1

- The function's GWAKEM general wakeup bit is 1
- The function's GWAKEE general-wake event pending bit is 1

GWAKE is an alias of the function's SYSCON:PME_Status latch. GWAKEE is set to 1 when any of the conditions occurs which would set PME_Status for that function, according to the masks in the function's Function Power Management register (PWRCFG0/1/2). In addition, writing a 1 to the GWAKE_F bit in the function's Function Event Force register sets the GWAKEE bit (and PME_Status bit) for that function to 1.

$\overline{\text{INTA}}$ Signal

In CardBus systems, assertion of the $\overline{\text{INTA}}$ pin is controlled by the INTRM master interrupt mask bit, in addition to the other interrupt control registers on the ADSP-2192. The INTR_E bit indicates if an interrupt is pending. The $\overline{\text{INTA}}$ pin is asserted under the following conditions:

- The function's INTRM master interrupt/wakeup mask bit is 1
- The function's INTRE interrupt pending bit is 1

INTRE is set when any of the conditions occurs which would cause $\overline{\text{INTA}}$ to be asserted in PCI systems, according to the settings in the PCI Interrupt Register. All three functions are controlled by the same interrupt-detect signal. Additionally, writing a 1 to the INTRF bit in a function's Function Event Force register sets that function's INTRE bit and, if the corresponding INTRM bit is set, the $\overline{\text{INTA}}$ pin is asserted. Writing INTRF directly is the only way for an individual function to set its INTRE bit and hence signal an interrupt independently of the other functions.

CIS Tuple Requirements

The four Function Event Registers for each function are pointed to by a data structure in CIS (Card Information Services) RAM, which must be initialized by the DSP from ROM at power-up. A CISTPL_CONFIG_CB CIS tuple must be provided for each function to point to the function event registers in BAR1 at the appropriate offset:

Table B-11. CIS Tuple Requirements

Function	Value of TPCC_ADDR in CISTPL_CONFIG_CB	Meaning
0	0x0000_0101	Offset 0x0000_0100 within BAR 1
1	10x0000_0111	Offset 0x0000_0110 within BAR 1
2	0x0000_0121	Offset 0x0000_0120 within BAR 1

CardBus Function Event (CB_FE0) Register

All bits in this register are reset to zero.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTE	Reserved										GWKE	Reserved			

Table B-12. CB_FE0 Register Bit Description

Bit Position	Bit Name	Description
3:0	Reserved	
4	GWKE	General Wakeup Event Pending. This bit is equivalent to the PME_Status bit. It reads 1 if CB_FPS0:GWAKE has been set by either a wakeup event on AC'97 as enabled by APME, or by a wakeup event on GPIOs enabled by GPME. A write of a 1 clears this bit. This nonvolatile bit is reset by power-on reset only, and is not affected by PCI RST, $\overline{\text{SYSRST}}$ or Soft Reset.
14:5	Reserved	
15	INTE	Interrupt Event Pending. Reads 1 if CB_FPS0:INTR is set and CB_FEM0:INTRM is 1. Default=0. A write of 1 clears all of the interrupts DSPI, WKI, GPI, and TABI corresponding to bits 15:12 of the PCS register. This bit is cleared by power-on reset and PCI RST. It is not affected by $\overline{\text{SYSRST}}$ or the Soft Reset bit PCC:RST.

CardBus Function Event Mask (CB_FEM0) Register

All bits in this register are reset to zero.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTM	WKUP	Reserved							PWM	BAM		Reserved			

Table B-13. CB_FEM0 Register Bit Descriptions

Bit Position	Bit Name	Description
3:0	Reserved	
4	GWKM	General Wakeup Mask. This bit is equivalent to PME_Enable. Enables assertion of CSTSCHG in Cardbus mode (see above).
5	Reserved	
6	Reserved	
13:7	Reserved	
14	WKUP	Wakeup Enable. Master wakeup enable for assertion of CSTSCHG/$\overline{PME}$ when in CardBus mode. When not in CardBus mode, this has no effect. This nonvolatile bit is reset by power-on reset only, and is not affected by PCI RST, $\overline{SYSRST}$ or Soft Reset.

Table B-13. CB_FEM0 Register Bit Descriptions (Continued)

Bit Position	Bit Name	Description
15	INTM	Interrupt / Wakeup Mask. Enables assertion of INTA and PME//CSTSCHG when in CardBus mode (CBUS = low). Has no effect upon INTA or PME/CSTSCHG when not in CardBus mode. In CardBus mode: INTA is asserted when: CB_FEM0:INTM=1 and CB_FE0:INTR=1. CSTSCHG is asserted high when: CB_FEM0:INTRM=1 and CB_FEM0:WKUP=1 and CB_FEM0:GWKM=1 and CB_FE0:GWKE=1. This bit is cleared by power-on reset and PCI RST. It is not affected by SYSRST or Soft Reset.

CardBus Function Event Present State (CB_FPS0) Register

All bits in this register are reset to zero.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTR	Reserved										GWAKE	Reserved			

Table B-14. CB_FPS0 Register Bit Descriptions

Bit Position	Bit Name	Description
3:0	Reserved	
4	GWAKE	Current wakeup state. This bit reflects the current state of the wakeup condition from either AC'97 or the local GPIOs, if enabled by ACPU or GPU.
14:5	Reserved	

Table B-14. CB_FPS0 Register Bit Descriptions (Continued)

Bit Position	Bit Name	Description
15	INTR	Current Interrupt State. This bit reflects the combined state of the current interrupt inputs to DSPI, WKI, GPI, and TABI.

Table B-15. CardBus Function Event Registers

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
CBEVENT0	Function 0 Event	0x103-0x100	n/a	0x01	0x00
CBEVENT_MSK0	Function 0 Event Mask	0x107-0x104	n/a	0x01	0x04
CBPRES_STATE0	Function 0 Present State	0x10B-0x108	n/a	0x01	0x08
CBEVENT_FORCE0	Function 0 Event Force	0x10F-0x10C	n/a	0x01	0x0C
CBEVENT1	Function 1 Event	0x113-0x110	n/a	0x01	0x10
CBEVENT_MSK1	Function 1 Event Mask	0x117-0x114	n/a	0x01	0x14
CBPRES_STATE1	Function 1 Present State	0x11B-0x118	n/a	0x01	0x18
CBEVENT_FORCE1	Function 1 Event Force	0x11F-0x11C	n/a	0x01	0x1C
CBEVENT2	Function 2 Event	0x123-0x120	n/a	0x01	0x20
CBEVENT_MSK2	Function 2 Event Mask	0x127-0x124	n/a	0x01	0x24

Table B-15. CardBus Function Event Registers

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
CBPRES_STATE2	Function 2 Present State	0x12B-0x128	n/a	0x01	0x28
CBEVENT_FORCE2	Function 2 Event Force	0x12F-0x12C	n/a	0x01	0x2C

CardBus Function Event Force (CB_FEFx) Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTF	Reserved										GWKF	Reserved			

Table B-16. CB_FEFx Register Bit Descriptions

Bit Position	Bit Name	Description
3:0	Reserved	
4	GWKF	Wakeup Force. Sets the CB_FE0:GWAKE bit in the Function Event register (equivalent to PME_Status). Does not affect the state of CB_FPS0:GWAKE in the Function Present State register.
14:5	Reserved	
15	INTF	Interrupt Force. Sets the CB_FE0:INTRE bit in the Function Event register. Does not affect the CB_FPS0:INTR bit in the Function Present State register.

AC'97 Controller Registers

These control registers for the serial port are used for audio (sound) and modem, specifically V.90 modems under codec control.

Table B-17. AC'97 Control Registers

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
AC97LCTL	AC'97 Link Control Setup control for AC'97 interface	0x0C1-0x0C0	0x00C1-0x00C0	0x00	0xC0
AC97STAT	AC'97 Link Status Setup control for AC'97 interface	0x0C3-0x0C2	0x00C3-0x00C2	0x00	0xC2
AC97SEN	AC'97 Slot Enable Register Setup control for AC'97 interface	0x0C5-0x0C4	0x00C5-0x00C4	0x00	0xC4
AC97SVAL	AC'97 Input Slot Valid Current status of valid frame from AC'97 link	0x0C7-0x0C6	0x00C7-0x00C6	0x00	0xC6
AC97SREQ	AC'97 Slot Request Current status of AC'97 slot requests	0x0C9-0x0C8	0x00C9-0x00C8	0x00	0xC8

Table B-17. AC'97 Control Registers (Continued)

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
AC97SIF	AC'97 External GPIO Status Register GPIO slot 12 interface register	0x0CB-0x0CA	0x00CB-0x00CA	0x00	0xCA

AC'97 Link Control/Status Register (AC97LCTL)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					ARPD	AGPE	ACWE	LKEN	BCEN	BCOE	EYE-BOX	AFD	AFS	AFR	SYEN

AC'97 Link Status Register (AC97STAT)

The following illustration shows the AC'97 Link Status Register Bit Definitions

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACR[2:0]			Reserved			REG	SYNC	LCOK	BCOK	AGI[2:0]			BGS[3:1]		

All the bits in this register reset to zero.

AC'97 Slot Enable Register (AC97SEN)

The numbers indicated after the bit name (ACSE12, for example) indicate the relative slot number. Slots are numbered in increasing order (0 first), while bits are numbered in decreasing order (MSB first).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			ACSE3	ACSE4	ACSE5	ACSE6	ACSE7	ACSE8	ACSE9	ACSE10	ACSE11	ACSE12	Reserved		

The bits in the ACSE register reset to zero.

AC'97 Input Slot Valid Register (AC97SVAL)

The numbers indicated after the bit name (ACSV12, for example) indicate the relative slot number. Slots are numbered in increasing order (0 first), and bits are numbered in decreasing order (MSB first).

The bits in the AC97SVAL register reset to zero.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACR	ACSV1	ACSV2	ACSV3	ACSV4	ACSV5	ACSV6	ACSV7	ACSV8	ACSV9	ACSV10	ACSV11	ACSV12	Reserved		

AC'97 Slot Request Register (AC97SREQ)

The numbers indicated after the bit name (ACRQ12, for example) indicate the relative slot number. Slots are numbered in increasing order (0 first), and bits are numbered in decreasing order (MSB first).

The bits in the AC97SREQ register reset to zero.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			ACRQ3	ACRQ4	ACRQ5	ACRQ6	ACRQ7	ACRQ8	ACRQ9	ACRQ10	ACRQ11	ACRQ12	Reserved		

AC'97 GPIO Status Register (AC97SIF)

The bits in the AC97SIF register reset to zero.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AGS15	AGS14	AGS13	AGS12	AGS11	AGS10	AGS9	AGS8	AGS7	AGS6	AGS5	AGS4	AGS3	AGS2	AGS1	AGS0

AC'97 Codec Registers

These register spaces are used to access registers in external codecs that are connected to the AC'97 port. Refer to the standard list provided in documentation for your codec. The IDs associated with primary, secondary, and tertiary registers report addressing information available for accessing the codecs.

AC'97 Codec Register Space-Primary Codec 0 (AC97EXT0) Register

Table B-18. AC'97 External Codec Space 0 Registers

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
AC97EXT0	AC'97 External Codec Space 0. External Primary Codec 0 Register	0x47F-0x400	0x047F-0x0400	0x04	0x7F-0x00

AC'97 Codec Register Space, Secondary Codec 1 (AC97EXT1) Register

Table B-19. AC'97 External Codec Space 1 Registers

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
AC97EXT1	AC'97 External Codec Space 1. External Secondary Codec 1 Register	0x57F-0x500	0x057F-0x0500	0x05	0x7F-0x00

AC'97 Codec Register Space, Secondary Codec 2 (AC97EXT2) Register

Table B-20. AC'97 External Codec Space 2 Registers

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
AC97EXT2	AC'97 External Codec Space 2. External Secondary Codec 2 Register	0x67F-0x600	0x067F-0x0600	0x06	0x7F-0x00

PCI DMA Address, Count Registers

The PCI DMA registers described in the following sections refer to the Address generator block (shown in Figure B-1 on page B-3) for PCI Bus Mastering. Use these registers to specify general addressing information.

DMA Control Registers

These registers control bus mastering transactions.

PCI DMA Control Registers

All four PCI DMA control registers, listed below, share the same bit structure and bit descriptions. Refer to "Setting I/O Processor—Host Port Modes" on page 7-12 for descriptions of the bits in these registers.

- DMA PCI Control/Status - PCI DMA Channel Control (`Rx0`) Register
- DMA PCI Control/Status-Transmit (`Tx0`)

- DMA PCI Control/Status (Rx1)
- DMA PCI Control/Status (Tx1)

None of the PCI DMA control registers can be reset.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								EOL	FLG	SGVL		INTMODE		LP EN	SGDEN

PCI Interrupt, Control Registers

Use the PCI registers to access the PCI DSP interface.

Table B-21. PCI Interrupt Control Registers

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
PCI_MSTRCNT0	DMA Transfer Count0 Bus master sample transfer count 0.	0x881-0x880	No Access.	0x08	0x80
PCI_MSTRCNT1	DMA Transfer Count1 Bus master sample transfer count 1.	0x883-0x882	No Access.	0x08	0x82
PCI_MSTRCTL0	DMA Control0 Bus master control and status 0.	0x885-0x884	No Access.	0x08	0x84
PCI_MSTRCTL1	DMA Control1 Bus master control and status 1.	0x887-0x886	No Access.	0x08	0x86

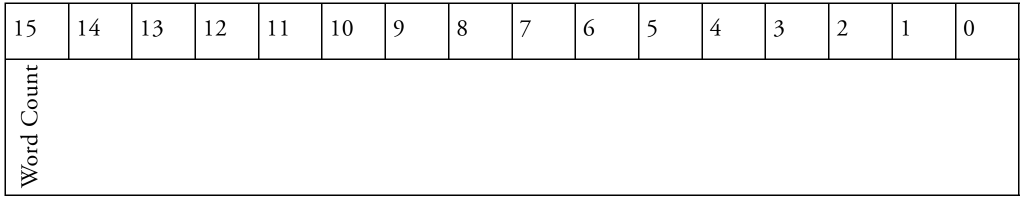
Table B-21. PCI Interrupt Control Registers (Continued)

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
PCI_IRQSTAT	Interrupt Register Status bits for all PCI interrupt sources.	0x889-0x888	No Access.	0x08	0x88
PCI_CFGCTL	PCI Control Includes configuration register read/write control.	0x88A	No Access.	0x08	0x8A

DMA Transfer Count 0 - Bus Master Sample Transfer Count (PCI_MSTRCNT0) Register

This 16-bit register contains a count of the number of words to be transferred between PCI address space and the DSP internal memory.

All bits in this register reset to 0.



DMA Transfer Count 1 - Bus Master Sample Transfer Count (PCI_MSTRCNT1) Register

This 16-bit register contains a count of the number of words to be transferred between PCI address space and the DSP internal memory.

All bits in this register reset to 0.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word Count															

DMA Control X - Bus Master Control and Status (PCI_DMACx) Register

All bits in this register reset to 0.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					LOOP	HALT	EMPTY	FUNCTION <2:0>			PACK DIS	DSP2/$\overline{\text{DSP1}}$	Flush FIFO	WR/$\overline{\text{RD}}$	DMA EN

Table B-22. PCI_DMACx Register Bit Descriptions

Bit position	Bit name	Description
0	DMA EN	DMA Enable.
1	WR/$\overline{\text{RD}}$	DMA Write / Read.
2	Flush FIFO	Flush Master FIFO.
3	DSP2/$\overline{\text{DSP1}}$	Select DSP2 / DSP1.
4	PACK DIS	DMA Packing Disable (DWORD Mode).
7:5	FUNCTION <2:0>	Function Select (0, 1, and 2).

Table B-22. PCI_DMACx Register Bit Descriptions (Continued)

Bit position	Bit name	Description
8	EMPTY	DMA FIFO Empty Status (1 = Empty).
9	HALT	DMA Channel Halt Status (1 = Halted).
10	LOOP	DMA Channel Loop Status (1 = Looping Occurred).
15:11	Reserved	

PCI Interrupt (PCI_IRQSTAT) Register

There are a variety of potential sources of interrupts to the PCI host besides the bus master DMA interrupts. A single interrupt pin, $\overline{\text{INTA}}$, is signals these interrupts back to the host. The PCI Interrupt Register consolidates all of the possible interrupt sources; the bits of this register are shown in Table B-28 on page B-60. The register bits are set by the various sources and can be cleared by writing a 1 to the bits to be cleared.

Interrupts may be sensitive either to edges or levels, as indicated in Table B-28. In particular, the PCI GPIO interrupt is level sensitive, and is asserted when any of the GPIO's individual sticky status bits is true. If an interrupt service routine is in the process of acknowledging one GPIO interrupt (by clearing its sticky status and then writing a 1 to PCI_IRQSTAT:GPIO) while an event occurs on another GPIO, it is possible for the ISR to miss the second event, should it occur between the time the ISR reads the GPIOs' status and when the ISR clears the PCI_IRQSTAT:GPIO bit. The GPIO interrupt is level sensitive to accommodate this case; the PCI_IRQSTAT:GPIO interrupt bit and the $\overline{\text{INTA}}$ pin immediately reassert after clearing. The ISR may be written in two ways to detect this case: it may either exit and be immediately re-triggered, or it may read back the PCI_IRQSTAT register after the clear to see if any bit has been set again indicating the occurrence of some new interrupt.

Status bits for all possible PCI interrupt sources (read/write-1-clear).

All bits in this register reset to 0.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	Target Abort	Master Abort	AC'97	GPIO	Reserved	Reserved	MBox 1 OUT	MBox 0 OUT	MBox 1 IN	MBox 0 IN	TX1 DMA	TX0 DMA	RX1 DMA	RX0 DMA	Reserved

Table B-23. PCI_IRQSTAT Register Bit Descriptions

Bit position	Bit name	Description
0	Reserved	
1	RX0 DMA	Rx0 DMA Channel Interrupt. Receive Channel 0 Bus Master Transactions Sensitivity: Edge
2	RX1 DMA	Rx1 DMA Channel Interrupt. Receive Channel 1 Bus Master Transactions Sensitivity: Edge
3	TX0 DMA	Tx0 DMA Channel Interrupt. Transmit Channel 0 Bus Master Transactions Sensitivity: Edge
4	Tx1 DMA	Tx1 DMA Channel Interrupt. Transmit Channel 1 Bus Master Transactions Sensitivity: Edge

Table B-23. PCI_IRQSTAT Register Bit Descriptions (Continued)

Bit position	Bit name	Description
5	MBox 0 IN	Incoming Mailbox 0 PCI Interrupt. PCI to DSP Mailbox 0 Transfer Sensitivity: Edge
6	MBox 1 IN	Incoming Mailbox 1 PCI Interrupt. PCI to DSP Mailbox 1 Transfer Sensitivity: Edge
7	MBox 0 OUT	Outgoing Mailbox 0 PCI Interrupt. DSP to PCI Mailbox 0 Transfer Sensitivity: Edge
8	MBox 1 OUT	Outgoing Mailbox 1 PCI Interrupt. DSP to PCI Mailbox 1 Transfer Sensitivity: Edge
9	Reserved	
10	Reserved	Reserved
11	GPIO	General-purpose I/O Pin Initiated. Sensitivity: Level
12	AC'97	AC'97 Interface Initiated. Sensitivity: Edge
13	Master Abort	PCI Interface Master Abort Detected. Sensitivity: Edge
14	Target Abort	PCI Interface Target Abort Detected. Sensitivity: Edge
15	Reserved	

PCI Control (PCI_CFGCTL) Register

All bits in this register reset to 0.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	TAbort IEN	MAbort IEN	AC'97 IEN	GPIO INE	Reserved	Reserved	D2PM1 IENE	D2PM0 IEN	P2DM1 IEN	P2DM0 IEN	Reserved		Conf Rdy	PCIF1	PCIF0

Table B-24. PCI_CFGCTL Register Bit Descriptions

Bit position	Bit name	Description
1-0	PCIF[1:0]	PCI Functions Configured. 00 = One PCI Function enabled 01= Two functions 10= Three functions
2	Conf Rdy	Configuration Ready. When 0, disables PCI accesses to the ADSP-2192 (terminated with Retry). Must be set to 1 by DSP ROM code after initializing configuration space. Once 1, cannot be written to 0.
4:3	Reserved	
5	P2DM0 IEN	PCI to DSP Mailbox 0 Transfer Interrupt Enabled.
6	P2DM1 IEN	PCI to DSP Mailbox 1 Transfer Interrupt Enabled.
7	D2PM0 IEN	DSP to PCI Mailbox 0 Transfer Interrupt Enabled.
8	D2PM1 IEN	DSP to PCI Mailbox 1 Transfer Interrupt Enabled.

Table B-24. PCI_CFGCTL Register Bit Descriptions (Continued)

Bit position	Bit name	Description
9	Reserved	
10	Reserved	Reserved
11	GPIO IEN	General-purpose I/O Pin Initiated Interrupt Enabled.
12	AC'97 IEN	AC'97 Interface Initiated Interrupt Enabled.
13	MAbort IEN	PCI Interface Master Abort Detect Interrupt Enabled.
14	TAbort IEN	PCI Interface Target Abort Detect Interrupt Enabled.
15	Reserved	

PCI Configuration Register Space

The ADSP-2192 PCI Interface requires separate configuration space for each function due to operating system requirements. This section describes the registers in each function, their reset conditions, and interaction between the functions to access and control the ADSP-2192 hardware.

Commonalities Between the Three Functions

Each function contains a complete set of registers in the predefined header region, as defined in PCI Local Bus Specification, Revision 2.2. In addition, each function contains optional registers to support PCI Bus Power Management. Registers that are unimplemented or read-only in one function are similarly defined in the other functions.

Each function contains four base address registers that access ADSP-2192 control registers and DSP memory. Base address register (BAR1) accesses the ADSP-2192 control registers. Accesses to the control registers via BAR1 use PCI memory accesses. BAR1 requests a memory allocation of 1024 bytes. Access to DSP memory occurs via BAR2 and BAR3. BAR2 is accesses 24-bit DSP memory (i.e. for DSP program downloading) and BAR3 accesses 16-bit DSP memory. BAR4 provides I/O space access to both the control registers and the DSP memory.

The configuration space headers are defined by Function 0 (register information shown in Table B-28 on page B-60), Function 1 (register information shown in Table B-29 on page B-63), and Function 2 (register information shown in Table B-30 on page B-65).

Each function is defined by writing to the class code register of that function during bootup. Additionally, during boot time, the DSP will have the possibility of disabling one or more of the functions. If only two functions are enabled, they will be functions zero and one. If only one function is enabled, it will be function zero.

Interactions Between the Three Functions

Because all functions access and control a single set of resources, potential conflicts occur in the control specified by the configuration. For each of the potential conflicts, a resolution is proposed. Table B-31 on page B-67 and Table B-32 on page B-70 identify the proposed resolutions (interactions). Table B-31 covers the registers in the predefined header space and Table B-32 covers the Power Management registers.

Target accesses to registers and DSP memory can go through any function. As long as the Memory Space access enable bit is set in that function, then PCI memory accesses whose address matches the locations programmed into a function's BARs 1-3 will be able to read or write any visible register or memory location within the ADSP-2192. Similarly, if I/O Space access enable is set, then PCI I/O accesses can be performed via BAR4.

Within the Power Management section of the configuration blocks, there are a few interactions. The part will stay in the highest power state between the three functions. Thus if a modem is requested to be powered down to state D2, but Function 0 is set for power state D0, the overall chip will remain in state D0. When one or the other of the functions is in a low power state, they can only respond to configuration accesses, regardless of the power state of the other functions. Similarly, when a function transitions from D3hot to D0, that function's configuration space will be re-initialized. Each function has a separate PME enable and PME status bit. Whenever possible, the hardware will identify Function 0 wakeup from wakeup and set the appropriate PME status. When no determination is possible, both PME status bits will be set.

PCI Configuration Register Space, Function 0

PCI Configuration Spaces should only be accessed by the DSP, and only during the boot process. After the PCI interface has been configured, bit 2 of the PCI_CFGCTL register (ConfRdy) should be set by the DSP. This allows the PCI interface access to these registers while at the same time denying the DSP access.

Access to these registers is controlled by the PCI RDY bit in the Chip Mode/Status Register (Page 0x00, Address 0x00).

Table B-25. Function 0 Registers

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
PCI_CFG0_VID	Config0 Vendor ID	0x01-0x00	n/a	0x09	0x00
PCI_CFG0_DID	Config0 Device ID	0x03-0x02	n/a	0x09	0x02

Table B-25. Function 0 Registers (Continued)

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
PCI_CFG0_CCODEL	Config0 Class Code[7:0],Rev ID	0x08	n/a	0x09	0x08
PCI_CFG0_CCODEH	Config0 Class Code[23:8]	0x0B-0x0A	n/a	0x09	0x0A
PCI_CFG0_SVID	Config0 Sub-system Vendor ID	0x2D-0x2C	n/a	0x09	0x2C
PCI_CFG0_SDID	Config0 Sub-system Device ID	0x2F-0x2E	n/a	0x09	0x2E
PCI_CFG0_PWRMT	Config0 Power Mgt Capabilities. Bit 15 set if Vaux is sensed valid.	0x45-0x44	n/a	0x09	0x44

PCI Configuration Register Space, Function 1

PCI Configuration Spaces should be accessed only by the DSP, and only during the boot process. After the PCI interface has been configured, bit 2 of the PCI_CFGCTL register (ConfRdy) should be set by the DSP. This allows the PCI interface access to these registers while at the same time denying the DSP access.

Access to these registers is controlled by the PCI RDY bit in the Chip Mode/Status Register (Page 0x00, Address 0x00). See "ADSP-2192 Chip Control Registers" on page B-13.

Table B-26. Function 1 Registers

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
PCI_CFG1_VID	Config1 Vendor ID	0x01-0x00	n/a	0x0A	0x00
PCI_CFG1_DID	Config1 Device ID	0x03-0x02	n/a	0x0A	0x02
PCI_CFG1_CCODEL	Config1 Class Code[7:0], Rev ID	0x08	n/a	0x0A	0x08
PCI_CFG1_CCODEH	Config1 Class Code[23:8]	0x0B-0x0A	n/a	0x0A	0x0A
PCI_CFG1_SVID	Config1 Sub-system Vendor ID	0x2D-0x2C	n/a	0x0A	0x2C
PCI_CFG1_SDID	Config1 Sub-system Device ID	0x2F-0x2E	n/a	0x0A	0x2E

Table B-26. Function 1 Registers (Continued)

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
PCI_CFG1_PWRMT	Config1 Power Mgt Capabilities Bit 15 set, if Vaux is sensed valid.	0x45-0x44	n/a	0x0A	0x44

PCI Configuration Register Space, Function 2

PCI Configuration Spaces should be accessed only by the DSP, and only during the boot process. After the PCI interface has been configured, bit 2 of the `PCI_CFGCTL` register (`ConfRdy`) should be set by the DSP. This allows the PCI interface access to these registers while at the same time denying the DSP access.

Access to these registers is controlled by the PCI RDY bit in the PCI Interrupt Control Register (Page 0x08, Address 0xA2). See "General-purpose I/O (GPIO) Control Registers" on page B-24.

Table B-27. Function 2 Registers

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
PCI_CFG2_VID	Config2 Vendor ID	0x01-0x00	n/a	0x0B	0x00
PCI_CFG2_DID	Config2 Device ID	0x03-0x02	n/a	0x0B	0x02
PCI_CFG2_CCODEL	Config2 Class Code[7:0],Rev ID	0x08	n/a	0x0B	0x08

Table B-27. Function 2 Registers (Continued)

Register Name	Description	PCI Address	USB Address	DSP I/O Page	DSP I/O Address
PCI_CFG2_CCODEH	Config2 Class Code[23:8]	0x0B-0x0A	n/a	0x0B	0x0A
PCI_CFG2_SVID	Config2 Sub-system Vendor ID	0x2D-0x2C	n/a	0x0B	0x2C
PCI_CFG2_SDID	Config2 Sub-system Device ID	0x2F-0x2E	n/a	0x0B	0x2E
PCI_CFG2_PWRMT	Config2 Power Mgt Capabili-ties. Bit 15 set if Vaux is sensed valid.	0x45-0x44	n/a	0x0B	0x44

PCI Configuration Space

Table B-28 on page B-60, Table B-29 on page B-63, and Table B-30 on page B-65 show the PCI Configuration Space definitions for functions 0, 1, and 2.

Table B-28. PCI CONFIG SPACE for Function 0

Address	Name	Reset	Comments
0x01-0x00	Vendor ID	0x11D4	Writable from the DSP during initial-ization
0x03-0x02	Device ID	0x2192	Writable from the DSP during initial-ization

Table B-28. PCI CONFIG SPACE for Function 0 (Continued)

Address	Name	Reset	Comments
0x05-0x04	Command Register	0x0	Bus Master, Memory Space Capable, I/O Space Capable
0x07-0x06	Status Register	0x0	Bits enabled: Capabilities List, Fast B2B, Medium Decode
0x08	Revision ID	0x0	Writable from the DSP during initialization
0x0B-0x09	Class Code	0x078000	Writable from the DSP during initialization
0x0C	Cache Line Size	0x0	Read-only
0x0D	Latency Timer	0x0	
0x0E	Header Type	0x80	Multifunction bit set
0x0F	BIST	0x0	Unimplemented
0x13-0x10	Base Address 1	0x08	Register Access for all ADSP-2192 Registers, Prefetchable Memory
0x17-0x14	Base Address 2	0x08	24-bit DSP Memory Access
0x1B-0x18	Base Address 3	0x08	16-bit DSP Memory Access
0x1F-0x1C	Base Address 4	0x01	I/O access for control registers and DSP memory
0x23-0x20	Base Address 5	0x0	Unimplemented
0x27-0x24	Base Address 6	0x0	Unimplemented
0x2B-0x28	Cardbus CIS Pointer	0x1FF03	CIS RAM Pointer - Function 0 (Read Only).

Table B-28. PCI CONFIG SPACE for Function 0 (Continued)

Address	Name	Reset	Comments
0x2D-0x2C	Subsystem Vendor ID	0x11D4	Writable from the DSP during initialization
0x2F-0x2E	Subsystem Device ID	0x2192	Writable from the DSP during initialization
0x33-0x30	Expansion ROM Base Address	0x0	Unimplemented
0x34	Capabilities Pointer	0x40	Read-only
0x3C	Interrupt Line	0x0	
0x3D	Interrupt Pin	0x1	Uses $\overline{\text{INTA}}$ Pin
0x3E	Min_Gnt	0x1	Read-only
0x3F	Max_Lat	0x4	Read-only
0x40	Capability ID	0x1	Power Management Capability Identifier
0x41	Next_Cap_Ptr	0x0	Read-only
0x43-0x42	Power Management Capabilities	0x6C22	Writable from the DSP during initialization
0x45-0x44	Power Management Control/Status	0x0	Bits 15 and 8 initialized only on Power-up

Table B-29. PCI Configuration Space for Function 1

Address	Name	Reset	Comments
0x01-0x00	Vendor ID	0x11D4	Writable from the DSP during initialization
0x03-0x02	Device ID	0x219A	Writable from the DSP during initialization
0x05-0x04	Command Register	0x0	Bus Master, Memory Space Capable, I/O Space Capable
0x07-0x06	Status Register	0x0	Bits enabled: Capabilities List, Fast B2B, Medium Decode
0x08	Revision ID	0x0	Writable from the DSP during initialization
0x0B-0x09	Class Code	0x078000	Writable from the DSP during initialization
0x0C	Cache Line Size	0x0	Read-only
0x0D	Latency Timer	0x0	Read-only
0x0E	Header Type	0x80	Multifunction bit set
0x0F	BIST	0x0	Unimplemented
0x13-0x10	Base Address 1	0x08	Register Access for all ADSP-2192 Registers, Prefetchable Memory
0x17-0x14	Base Address 2	0x08	24-bit DSP Memory Access
0x1B-0x18	Base Address 3	0x08	16-bit DSP Memory Access
0x1F-0x1C	Base Address 4	0x01	I/O access for control registers and DSP memory

Table B-29. PCI Configuration Space for Function 1 (Continued)

Address	Name	Reset	Comments
0x23-0x20	Base Address5	0x0	Unimplemented
0x27-0x24	Base Address 6	0x0	Unimplemented
0x2B-0x28	Cardbus CIS Pointer	0x1FE03	CIS RAM Pointer - Function 1 (Read Only).
0x2D-0x2C	Subsystem Vendor ID	0x11D4	Writable from the DSP during initialization
0x2F-0x2E	Subsystem Device ID	0x219A	Writable from the DSP during initialization
0x33-0x30	Expansion ROM Base Address	0x0	Unimplemented
0x34	Capabilities Pointer	0x40	Read-only
0x3C	Interrupt Line	0x0	
0x3D	Interrupt Pin	0x1	Uses $\overline{\text{INTA}}$ Pin
0x3E	Min_Gnt	0x1	Read-only
0x3F	Max_Lat	0x4	Read-only
0x40	Capability ID	0x1	Power Management Capability Identifier
0x41	Next_Cap_Ptr	0x0	Read-only
0x43-0x42	Power Management Capabilities	0x6C22	Writable from the DSP during initialization
0x45-0x44	Power Management Control/Status	0x0	Bits 15 and 8 initialized only on power-up.

Table B-30. PCI Configuration Space for Function 2

Address	Name	Reset	Comments
0x01-0x00	Vendor ID	0x11D4	Writable from the DSP during initialization
0x03-0x02	Device ID	0x219E	Writable from the DSP during initialization
0x05-0x04	Command Register	0x0	Bus Master, Memory Space Capable, I/O Space Capable
0x07-0x06	Status Register	0x0	Bits enabled: Capabilities List, Fast B2B, Medium Decode
0x08	Revision ID	0x0	Writable from the DSP during initialization
0x0B-0x09	Class Code	0x040100	Writable from the DSP during initialization
0x0C	Cache Line Size	0x0	Read-only
0x0D	Latency Timer	0x0	Read-only
0x0E	Header Type	0x80	Multifunction bit set
0x0F	BIST	0x0	Unimplemented
0x13-0x10	Base Address 1	0x08	Register Access for all ADSP-2192 Registers, Prefetchable Memory
0x17-0x14	Base Address 2	0x08	24-bit DSP Memory Access
0x1B-0x18	Base Address 3	0x08	16-bit DSP Memory Access
0x1F-0x1C	Base Address 4	0x01	I/O access for control registers and DSP memory

Table B-30. PCI Configuration Space for Function 2 (Continued)

Address	Name	Reset	Comments
0x23-0x20	Base Address 5	0x0	Unimplemented
0x27-0x24	Base Address 5	0x0	Unimplemented
0x2B-0x28	Cardbus CIS Pointer	0x1FD03	CIS RAM Pointer - Function 2 (Read Only).
0x2D-0x2C	Subsystem Vendor ID	0x11D4	Writable from the DSP during initialization
0x2F-0x2E	Subsystem Device ID	0x219E	Writable from the DSP during initialization
0x33-0x30	Expansion ROM Base Address	0x0	Unimplemented
0x34	Capabilities Pointer	0x40	Read-only
0x3C	Interrupt Line	0x0	
0x3D	Interrupt Pin	0x1	Uses $\overline{\text{INTA}}$ Pin
0x3E	Min_Gnt	0x1	Read-only
0x3F	Max_Lat	0x4	Read-only
0x40	Capability ID	0x1	Power Management Capability Identifier
0x41	Next_Cap_Ptr	0x0	Read-only
0x43-0x42	Power Management Capabilities	0x6C22	Writable from the DSP during initialization
0x45-0x44	Power Management Control/Status	0x0	Bits 15 and 8 initialized only on power-up.

Interaction Between Registers

Table B-31 on page B-67 and Table B-32 on page B-70 show the register interactions between functions.

Table B-31. Configuration Space Register Interactions Between Functions

Name			**Comments**
Vendor ID			Separate registers, no interaction
Device ID			Separate registers, no interaction
GROUP	Description	Bit	Bit Function
Command Register Bits	I/O Space Enable	0	Enables are separate in each function and go along with the function's base addresses
	Memory Space Enable	1	Enables are separate in each function and go along with the function's base addresses
	Bus Master Enable	2	Enables are separate in each function and go along with the function's base addresses
	Special Cycles	3	None of the functions support special cycles, read-only
	Memory Write and Invalidate	4	No function generates Memory Write and Invalidate commands, read-only
	VGA Palette Snoop	5	Not applicable, read-only
	Parity Error Response	6	If any function has the bit set, $\overline{\text{PERR}}$ may be asserted
	Stepping Control	7	No address stepping is done, read-only

Table B-31. Configuration Space Register Interactions Between Functions (Continued)

Name			Comments
	SERR# Enable	8	If any function enables $\overline{\text{SERR}}$ driver, then $\overline{\text{SERR}}$ may be asserted
	Fast Back-to-back Enable	9	No function generates fast back-to-back transactions
Status Register Bits	Capabilities List	4	Read-only
	66 MHz Capable	5	Read-only
	Reserved	6	Read-only
	Fast Back-to-back Capable	7	Read-only
	Master Data Parity Error	8	Separate for each function, no interaction
	DEVSEL Timing	10-9	Read-only
	Signaled Target Abort	11	Separate for each function, no interaction
	Received Target Abort	12	Separate for each function, no interaction
	Received Master Abort	13	Separate for each function, no interaction
	Signaled System Error	14	Separate for each function, set if $\overline{\text{SERR}}$ enabled and $\overline{\text{SERR}}$ asserted
	Detected Parity Error	15	Separate for each function, but set in all functions simultaneously

Table B-31. Configuration Space Register Interactions Between Functions (Continued)

Name	Comments
Revision ID	Read-only
Class Code	Separate registers, no interaction
Cache Line Size	Read-only
Latency Timer	Separate for each function, no interaction
Header Type	Read-only
Base Address 1	In range signal ORed between functions, any function can access memory
Base Address 2	In range signal ORed between functions, any function can access memory
Base Address 3	In range signal ORed between functions, any function can access memory
Base Address 4	In range signal ORed between functions, any function can access memory
Subsystem Vendor ID	Separate registers, no interaction
Subsystem Device ID	Separate registers, no interaction
Capabilities Pointer	Read-only
Interrupt Line	Separate registers, no interaction
Interrupt Pin	Read-only
Min_Gnt	Read-only
Max_Lat	Read-only

Table B-32. Power Management Register Interactions Between Functions

Name			Comments
Capability ID			Read-only
Next_Cap_Ptr			Read-only
Power Management Capabilities Bits	Version	2-0	Read-only
	PME Clock	3	Read-only
	Reserved	4	Read-only
	Device Specific Initialization	5	Read-only
	Aux Current	8-6	Read-only by PCI, writable by DSP
	D1 Support	9	Read-only
	D2 Support	10	Read-only
	PME Support	15-11	Read-only by PCI, writable by DSP
Power Management Control/ Status Bits	Power State	1-0	Part will be in highest power state of the three functions
	Reserved	7-2	Read-only, no interaction
	PME Enable	8	Separate for each function, no interaction
	Data Select	12-9	Read-only, no interaction

Table B-32. Power Management Register Interactions Between Functions (Continued)

Name			Comments
	Data Scale	14-13	Read-only, no interaction
	PME Status	15	Separate for each function, may be set in all functions by a wakeup

USB DSP Registers

Overview

The USB registers control the USB interface, specifically the operation and configuration of the USB Interface. Most of these registers are accessible only via the USB Bus, although a subset is accessible to the DSP.

The ADSP-2192 USB allows you to configure and attach a single device with multiple interfaces and various endpoint configurations. The advantages to this design are:

- Programmable descriptors and class specific command interpreter. An MCU is supported on board, which allows you to soft download different configurations and support any number of standard or class specific commands.
- Eight user defined endpoints are provided. Endpoints can be configured as either `BULK`, `ISO`, or `INT` and can be grouped and assigned to any interface.

DSP Register Definitions

For each endpoint, four registers are defined in order to provide a memory buffer in the DSP. These registers are defined for each endpoint shared by all interfaces that are defined for a total of 4x8 = 32 registers. These registers are read/write by the DSP only.

Table B-33. USB DSP Register Definitions

Page	Address	Name
0x0C	0x00-0x03	DSP Memory Buffer Base Addr EP4
0x0C	0x04-0x05	DSP Memory Buffer Size EP4
0x0C	0x06-0x07	DSP Memory Buffer RD Offset EP4
0x0C	0x08-0x09	DSP Memory Buffer WR Offset EP4
0x0C	0x10-0x13	DSP Memory Buffer Base Addr EP5
0x0C	0x14-0x15	DSP Memory Buffer Size EP5
0x0C	0x16-0x17	DSP Memory Buffer RD Offset EP5
0x0C	0x18-0x19	DSP Memory Buffer WR Offset EP5
0x0C	0x20-0x23	DSP Memory Buffer Base Addr EP6
0x0C	0x24-0x25	DSP Memory Buffer Size EP6
0x0C	0x26-0x27	DSP Memory Buffer RD Offset EP6
0x0C	0x28-0x29	DSP Memory Buffer WR Offset EP6
0x0C	0x30-0x33	DSP Memory Buffer Base Addr EP7
0x0C	0x34-0x35	DSP Memory Buffer Size EP7
0x0C	0x36-0x37	DSP Memory Buffer RD Offset EP7
0x0C	0x38-0x39	DSP Memory Buffer WR Offset EP7
0x0C	0x40-0x43	DSP Memory Buffer Base Addr EP8

Table B-33. USB DSP Register Definitions (Continued)

Page	Address	Name
0x0C	0x44-0x45	DSP Memory Buffer Size EP8
0x0C	0x46-0x47	DSP Memory Buffer RD Offset EP8
0x0C	0x48-0x49	DSP Memory Buffer WR Offset EP8
0x0C	0x50-0x53	DSP Memory Buffer Base Addr EP9
0x0C	0x54-0x55	DSP Memory Buffer Size EP9
0x0C	0x56-0x57	DSP Memory Buffer RD Offset EP9
0x0C	0x58-0x59	DSP Memory Buffer WR Offset EP9
0x0C	0x60-0x63	DSP Memory Buffer Base Addr EP10
0x0C	0x64-0x65	DSP Memory Buffer Size EP10
0x0C	0x66-0x67	DSP Memory Buffer RD Offset EP10
0x0C	0x68-0x69	DSP Memory Buffer WR Offset EP10
0x0C	0x70-0x73	DSP Memory Buffer Base Addr EP11
0x0C	0x74-0x75	DSP Memory Buffer Size EP11
0x0C	0x76-0x77	DSP Memory Buffer RD Offset EP11
0x0C	0x78-0x79	DSP Memory Buffer WR Offset EP11
0x0C	0x80-0x81	USB Descriptor Vendor ID
0x0C	0x84-0x85	USB Descriptor Product ID
0x0C	0x86-0x87	USB Descriptor Release Number
0x0C	0x88-0x89	USB Descriptor Device Attributes

DSP Memory Buffer Base Addr Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	DS	BA

most significant word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BA	BA	BA	BA	BA	BA	BA	BA	BA	BA	BA	BA	BA	BA	BA	BA

least significant word

Figure B-2. DSP Memory Buffer Base Addr Register

Points to the base address for the DSP memory buffer assigned to this Endpoint.

Table B-34. DSP Memory Buffer Base Addr Register

[DS, BA16:0]	Memory Buffer Base Address
DS	DSP Memory select bit. 0 = DSP1 memory space, 1 = DSP2 memory space
BA[16:0]	Lower 17 address bits

DSP Memory Buffer Size Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ	SZ

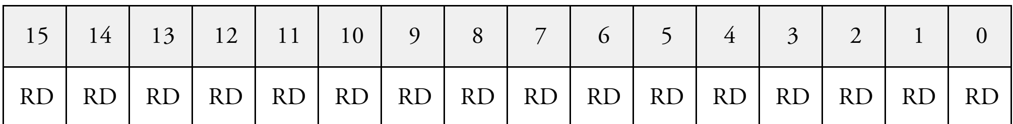
Figure B-3. DSP Memory Buffer Size Register

Indicates the size of the DSP memory buffer assigned to this Endpoint.

Table B-35. DSP Memory Buffer Size Register

SZ[15:0]	Memory Buffer Size

DSP Memory Buffer RD Pointer Offset Register



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RD	RD	RD	RD	RD	RD	RD	RD	RD	RD	RD	RD	RD	RD	RD	RD

Figure B-4. DSP Memory Buffer RD Pointer Offset Register

The offset from the base address for the read pointer of the memory buffer assigned to this Endpoint.

Table B-36. DSP Memory Buffer RD Pointer Offset Register

RD[15:0]	Memory Buffer RD Offset

DSP Memory Buffer WR Pointer Offset Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WR	WR	WR	WR	WR	WR	WR	WR	WR	WR	WR	WR	WR	WR	WR	WR

Figure B-5. DSP Memory Buffer WR Pointer Offset Register

The offset from the base address for the write pointer of the memory buffer assigned to this Endpoint.

Table B-37. DSP Memory Buffer WR Pointer Offset Register

WR[15:0]	Memory Buffer WR Offset

MCU Register Definitions

MCU registers are defined in four memory spaces that are grouped by the following address ranges:

0x0XXX	Defines general-purpose USB status and control registers
0x1XXX	Defines registers that are specific to Endpoint setup and control
0x2XXX	Defines the registers used for REGIO accesses to the DSP register space
0x3XXX	Defines the MCU program memory write address space

Table B-38. USB MCU Register Definitions

Address	Name	Comment
0x0000-0x0007	USB SETUP Token Cmd	8 bytes total
0x0008-0x000F	USB SETUP Token Data	8 bytes total

Table B-38. USB MCU Register Definitions (Continued)

Address	Name	Comment
0x0010-0x0011	USB SETUP Counter	16 bit counter
0x0012-0x0013	USB Control	Misc control including re-attach
0x0014-0x0015	USB Address/Endpoint	Address of device/active Endpoint
0x0016-0x0017	USB Frame Number	Current frame number
0x1000-0x1001	USB EP4 Description	Configures Endpoint
0x1002-0x1003	USB EP4 NAK Counter	
0x1004-0x1005	USB EP5 Description	Configures Endpoint
0x1006-0x1007	USB EP5 NAK Counter	
0x1008-0x1009	USB EP6 Description	Configures Endpoint
0x100A-0x100B	USB EP6 NAK Counter	
0x100C-0x100D	USB EP7 Description	Configures Endpoint
0x100E-0x100F	USB EP7 NAK Counter	
0x1010-0x1011	USB EP8 Description	Configures Endpoint
0x1012-0x1013	USB EP8 NAK Counter	
0x1014-0x1015	USB EP8 Description	Configures Endpoint
0x1016-0x1017	USB EP9 NAK Counter	
0x1018-0x1019	USB EP10 Description	Configures Endpoint
0x101A-0x101B	USB EP10 NAK Counter	

Table B-38. USB MCU Register Definitions (Continued)

Address	Name	Comment
0x101C-0x101D	USB EP11 Description	Configures Endpoint
0x101E-0x101F	USB EP11 NAK Counter	
0x1020-0x1021	USB EP STALL Policy	
0x1040-0x1043	USB EP1 Code Download Base Address	Starting address for code download on Endpoint 1
0x1044-0x1047	USB EP2 Code Download Base Address	Starting address for code download on Endpoint 2
0x1048-0x104B	USB EP3 Code Download Base Address	Starting address for code download on Endpoint 3
0x1060-0x1063	USB EP1 Code Current Write Pointer Offset	Current write pointer offset for code download on Endpoint 1
0x1064-0x1067	USB EP2 Code Current Write Pointer Offset	Current write pointer offset for code download on Endpoint 2
0x1068-0x106B	USB EP3 Code Current Write Pointer Offset	Current write pointer offset for code download on Endpoint 3
0x2000-0x2001	USB Register I/O Address	
0x2002-0x2003	USB Register I/O Data	
0x3000-0x3FFF	USB MCU Program Mem	

USB Endpoint Description Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TB	LT	LT	TY	TY	DR	PS	PS	PS	PS	PS	PS	PS	PS	PS	PS

Figure B-6. USB Endpoint Description Register

Provides the USB core with information about the Endpoint type, direction, and maximum packet size. This register is read/write by the MCU only. This register is defined for Endpoints[4:11].

Table B-39. USB Endpoint Description Register

PS[9:0]	Maximum packet size for Endpoint
LT[1:0]	Last transaction handshake indicator bits sent by the ADSP-2192: 00 = Clear 01 = ACK 10 = NAK 11 = ERR
TY[1:0]	Endpoint type bits: 00 = DISABLED 01 = ISO 10 = Bulk 11 = Interrupt
DR	Endpoint direction bit: 1 = IN 0 = OUT
TB	Toggle bit for Endpoint. Reflects the current state of the DATA toggle bit.

USB Endpoint NAK Counter Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	NE	ST	NC	NC	NC	NC

Figure B-7. USB Endpoint NAK Counter Register

Contains the individual NAK count, stall control, and NAK counter enable bits for Endpoints 4-11. This register is read/write by the MCU only.

Table B-40. USB Endpoint NAK Counter Register

NC[3:0]	NAK counter. Number of sequential NAKs that have occurred on a given Endpoint. When N[3:0] is equal to the base NAK counter NK[3:0] value in the Endpoint Stall Policy register, a zero-length packet or packet less than maxpacketsize will be issued.
ST	A value of 1 means: Endpoint is stalled
NE	1 = Enable NAK counter 0 = Disable NAK counter

USB Endpoint Stall Policy Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NK	NK	NK	NK	X	X	X	X	X	TB3	TB2	TB1	ST3	ST2	ST1	FE

Figure B-8. USB Endpoint Stall Policy Register

Contains the base NAK count and FIFO error policy bits for Endpoints 4-11. The STALL status and Data toggle bits for Endpoints 1-3 are included as well. This register is read/write by the MCU only.

Table B-41. USB Endpoint Stall Policy Register

ST[3:1]	A value of 1 means the Endpoint is stalled. ST[1] maps to Endpoint 1, ST[2] maps to Endpoint 2, etc.
TB[3:1]	Toggle bit for Endpoint. Reflects the current state of the DATA toggle bit. ST[1] maps to Endpoint 1, ST[2] maps to Endpoint 2, etc.
NK[3:0]	Base NAK counter. Determines how many sequential NAKs are issued before sending zero length packet, or a packet less than the maximum packet size, on any given Endpoint.
FE	FIFO error policy. A value of 1 means: Endpoint FIFO is overrun/underrun, STALL Endpoint

USB Endpoint 1 Code Download Base Address Register

LSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD

MSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	DS	AD

Figure B-9. USB Endpoint 1 Code Download Base Address Register

Contains an 18 bit address which corresponds to the starting location for DSP code download on Endpoint 1. This register is read/write by the MCU only. The most significant bit (`DS` bit) selects either DSP1 PM address space (`DS=0`) or DSP2 PM address space (`DS=1`).

USB Endpoint 2 Code Download Base Address Register

LSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD

MSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	DS	AD

Figure B-10. USB Endpoint 2 Code Download Base Address Register

Contains an 18 bit address which corresponds to the starting location for DSP code download on Endpoint 2. This register is read/write by the MCU only. The most significant bit (DS bit) selects either DSP1 PM address space (DS=0) or DSP2 PM address space (DS=1).

USB Endpoint 3 Code Download Base Address Register

LSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD

MSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	DS	AD

Figure B-11. USB Endpoint 3 Code Download Base Address Register

Contains an 18 bit address which corresponds to the starting location for DSP code download on Endpoint 3. This register is read/write by the MCU only. The most significant bit (DS bit) selects either DSP1 PM address space (DS=0) or DSP2 PM address space (DS=1).

USB Endpoint 1 Code Download Current Write Pointer Offset Register

LSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD

MSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	AD	AD

Figure B-12. USB Endpoint 1 Code Download Current Write Pointer Offset Register

Contains an 18 bit address which corresponds to the current write pointer offset from the base address register for DSP code download on Endpoint 1. The sum of this register and the EP1 code download base address register represents the last DSP PM location written.

This register is read by the MCU only and is cleared to 3FFFF (-1) when the Endpoint 1 Code Download Base Address Register is updated.

USB Endpoint 2 Code Download Current Write Pointer Offset Register

LSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD

MSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	AD	AD

Figure B-13. USB Endpoint 2 Code Download Current Write Pointer Offset Register

Contains an 18 bit address which corresponds to the current write pointer offset from the base address register for DSP code download on Endpoint 2. The sum of this register and the EP2 code download base address register represents the last DSP PM location written.

This register is read by the MCU only and is cleared to 3FFFF (-1) when the Endpoint 2 Code Download Base Address Register is updated.

USB Endpoint 3 Code Download Current Write Pointer Offset Register

LSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD	AD

MSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	AD	AD

Figure B-14. USB Endpoint 3 Code Download Current Write Pointer Offset Register

Contains an 18 bit address which corresponds to the current write pointer offset from the base address register for DSP code download on Endpoint 3. The sum of this register and the EP3 code download base address register represents the last DSP PM location written.

This register is read by the MCU only and is cleared to 3FFFF (-1) when the Endpoint 3 Code Download Base Address Register is updated.

USB SETUP Token Command Register

This register is defined as eight bytes long and contains the data sent on the USB from the most recent SETUP transaction. This register is read by the MCU only.

Table B-42. USB SETUP Token Command Register

Byte	7 ... 0
0	bmRequest
1	b Request
2	w Value (L)
3	w Value (H)
4	w Index (L)
5	w Index (H)
6	w Length (L)
7	w Length(H)

USB SETUP Token Data Register

If the most recent SETUP transaction involves a data OUT stage, the USB SETUP Token Data Register is defined as eight bytes long and contains the data sent on the USB during the data stage. This is also where the MCU will write data to be sent in response to a SETUP transaction involving a data IN stage. This register is read/write by the MCU only.

Table B-43. USB SETUP Token Data Register

Byte	7 ... 0
0	Data 0
1	Data 1
2	Data 2
3	Data 3
4	Data 4
5	Data 5
6	Data 6
7	Data 7

USB SETUP Counter Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	C3	C2	C1	C0

Figure B-15. USB SETUP Counter Register

Provides information as to the total size of the SETUP transaction data stage. This register is read/write by the MCU only.

C[3:0]	Counter bits.

The counter hardware is a modulo 4 bit down counter used for tallying data bytes in both the IN and OUT data stages of SETUP transactions. As such, the count value stored has different meanings.

IN Transfers:	The MCU loads the counter with the number of bytes to transfer (must be 8 or less since the USB Setup Token Data Register file is 8 bytes maximum). The USB interface then decrements the count value after each byte is transferred to the host.
OUT Transfers:	Starting from a cleared value of 0, the counter is decremented with each byte received from the host, including the two CRC bytes. For example, if 8 bytes are received, the count value progresses from 15, 14, 13, etc. to a value of 6 (inclusive is the 2 CRC bytes). The MCU reads the value and subtracts it from 14 to determine the actual number of data bytes in the USB Setup Token Register file (14 - 6 = 8 bytes).

USB Register I/O Address Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0

Figure B-16. USB Register I/O Address Register

Contains the address of the ADSP-2192 register that is to be read/written. This register is read/write by the MCU only.

Table B-44. USB Register I/O Address Register

A[15]	MCU sets to 1 to notify the PDC Register Interface block to start ADSP-2192 read/write cycle. PDC Register Interface block clears to 0 to notify MCU the read/write cycle has completed.
A[14]	1 = WRITE, 0 = READ
A[13:0]	ADSP-2192 address to read/write

USB Register I/O Data Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Figure B-17. USB Register I/O Data Register

Contains the data of the ADSP-2192 register which has been read or is to be written. This register is read/write by the MCU only.

Table B-45. USB Register I/O Data Register

D[15:0]	During READ this register contains the data read from the ADSP-2192, during WRITE this register is the data to be written to the ADSP-2192

USB Control Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INT	ISE	IIN	IOU	BY	X	X	ER	X	X	X	X	RW	MO	BB	DI

Figure B-18. USB Control Register

Controls various USB functions. This register is read/write by the MCU only.

Table B-46. USB Control Register

MO	A value of 1 means: MCU has completed boot sequence and is ready to respond to USB commands
DI	A value of 1 means: Disconnect CONFIG device and enumerate again using the downloaded MCU configuration
BB	A value of 1 means: After reset boot from MCU RAM, 0 = after reset boot from MCU ROM
RW	A value of 1 means: Enables remote wake-up capability, 0 = disables remote wake-up capability
INT	Active interrupt for the 8051 MCU
ISE	Current interrupt is for a SETUP token
IIN	Current interrupt is for an IN token sent with a non zero length data stage
IOU	Current interrupt is for an OUT token received with a non zero length data stage
BY	Busy bit. A value of 1 means: MCU is busy processing a command. USB interface responds with NAK to further IN/OUT requests from the host until MCU clears this bit.
ER	Error in the current SETUP transaction. Generate STALL condition on EP0.

USB Address/Endpoint Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	EP3	EP2	EP1	EP0	A6	A5	A4	A3	A2	A1	A0

Figure B-19. USB Address/Endpoint Register

Contains the USB address and active Endpoint. This register is read/write by the MCU only.

Table B-47. USB Address/Endpoint Register

A[6:0]	USB address assigned to device
EP[3:0]	USB last active Endpoint

USB Frame Number Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	FN1	FN9	FN8	FN7	FN6	FN5	FN4	FN3	FN2	FN1	FN1	FN0

Figure B-20. USB Frame Number Register

Contains the last USB frame number. This register is read by the MCU only.

Table B-48. USB Frame Number Register

FN[10:0]	USB frame number

Register and Bit #Defines File

The following example definitions file is for the ADSP-2192 DSP. For the most current definitions file, programs should use the version of this file that comes with the software development tools. The version of the file that appears here is included as a guide only.

```
/*
--------------------------------------------------------------
def2192-12.h - SYSTEM REGISTER BIT & ADDRESS DEFINITIONS FOR
ADSP-2192-12 DSP

Copyright (c) 2001 Analog Devices, Inc.,  All rights reserved

The def2192-12.h file defines ALL ADSP-2192-12 DSP symbolic
names.

--------------------------------------------------------------
*/
#ifndef __DEF2192_12_H_
#define __DEF2192_12_H_

// Begin with a 219x CORE
#include <def219x.h>

//------------------------------------------------------------
//           System Register bit definitions
//------------------------------------------------------------

//***********************************************
//   IRPTL and IMASK registers
//***********************************************

// Bit Positions
#define INT_MAILBXI_P 4 // Bit  4: Offset: 10: Mailbox
#define INT_TMZHI_P  5 // Bit  5: Offset: 14: Timer (High
Priority)
#define INT_INT6_P 6 // Bit  6: Offset: 18: Unused
#define INT_PCIBMI_P 7 // Bit  7: Offset: 1c: PCI
#define INT_DSPDSPI_P 8 // Bit  8: Offset: 20: DSP
#define INT_FIFO0TXI_P 9  // Bit  9: Offset: 24: FIFO 0 Transmit
Empty
```

```
#define INT_FIFO0RXI_P 10  // Bit 10: Offset: 28: FIFO 0 Receive
Full
#define INT_FIFO1TXI_P 11  // Bit 11: Offset: 2c: FIFO 1
Transmit Empty
#define INT_FIFO1RXI_P 12  // Bit 12: Offset: 30: FIFO 1 Receive
Full
#define INT_INT13_P 13 // Bit 13: Offset: 34: Unused
#define INT_INT14_P 14 // Bit 14: Offset: 38: Unused
#define INT_AC97FR_P 15 // Bit 15: Offset: 3c: AC97 serial port

// Bit Masks
#define INT_MAILBXI MK_BMSK_(INT_MAILBXI_P) // Offset: 10:
Mailbox
#define INT_TMZHI MK_BMSK_(INT_TMZHI_P) // Offset: 14: Timer
   // (High Priority)
#define INT_INT6 MK_BMSK_(INT_INT6_P) // Offset: 18: Unused
#define INT_PCIBMI MK_BMSK_(INT_PCIBMI_P) // Offset: 1c: PCI
#define INT_DSPDSPI MK_BMSK_(INT_DSPDSPI_P) // Offset: 20: DSP
#define INT_FIFO0TXI MK_BMSK_(INT_FIFO0TXI_P) // Offset: 24:
   // FIFO 0 Transmit Empty
#define INT_FIFO0RXI MK_BMSK_(INT_FIFO0RXI_P) // Offset: 28:
   // FIFO 0 Receive Full
#define INT_FIFO1TXI MK_BMSK_(INT_FIFO1TXI_P) // Offset: 2c:
   // FIFO 1 Transmit Empty
#define INT_FIFO1RXI MK_BMSK_(INT_FIFO1RXI_P) // Offset: 30:
   // FIFO 1 Receive Full
#define INT_INT13 MK_BMSK_(INT_INT13_P) // Offset: 34: Unused
#define INT_INT14  MK_BMSK_(INT_INT14_P) // Offset: 38: Unused
#define INT_AC97FR MK_BMSK_(INT_AC97FR_P) // Offset: 3c: AC97
serial port

//*************************************************
//   SRCTLx and STCTLx registers
//*************************************************

// Bit Positions
#define SCTL_SPEN_P      0                       // AC'97 FIFO
Connection Enable
#define SCTL_SSEL3_P    7                   // AC'97 Slot Select
#define SCTL_SSEL2_P    6                   // AC'97 Slot Select
#define SCTL_SSEL1_P    5                   // AC'97 Slot Select
#define SCTL_SSEL0_P    4                   // AC'97 Slot Select
#define SCTL_FIP2_P     10                      // AC'97 FIFO
Interrupt Position
```

```
#define SCTL_FIP1_P      9                        // AC'97 FIFO
Interrupt Position
#define SCTL_FIP0_P      8                        // AC'97 FIFO
Interrupt Position
#define SCTL_SDEN_P     11                        // AC'97 Port
DMA Enable
#define SCTL_FULL_P     13                        // FIFO Full,
(read-only)
#define SCTL_EMPTY_P    14                       // FIFO Empty,
(read-only)
#define SCTL_FLOW_P     15                        // FIFO
Over/Underflow, sticky, write-one-clear)

// Bit Masks
#define SCTL_SPEN MK_BMSK_(SCTL_SPEN_P) // AC'97 FIFO
Connection Enable
#define SCTL_SSEL3 MK_BMSK_(SCTL_SSEL3_P) // AC'97 Slot Select
#define SCTL_SSEL2 MK_BMSK_(SCTL_SSEL2_P) // AC'97 Slot Select
#define SCTL_SSEL1 MK_BMSK_(SCTL_SSEL1_P) // AC'97 Slot Select
#define SCTL_SSEL0 MK_BMSK_(SCTL_SSEL0_P) // AC'97 Slot Select
#define SCTL_FIP2 MK_BMSK_(SCTL_FIP2_P )  // AC'97 FIFO
Interrupt Position
#define SCTL_FIP1 MK_BMSK_(SCTL_FIP1_P )  // AC'97 FIFO
Interrupt Position
#define SCTL_FIP0 MK_BMSK_(SCTL_FIP0_P )  // AC'97 FIFO
Interrupt Position
#define SCTL_SDEN MK_BMSK_(SCTL_SDEN_P )  // AC'97 Port DMA
Enable
#define SCTL_FULL MK_BMSK_(SCTL_FULL_P )  // FIFO Full,
read-only
#define SCTL_EMPTY MK_BMSK_(SCTL_EMPTY_P)  // FIFO Empty,
read-only
#define SCTL_FLOW MK_BMSK_(SCTL_FLOW_P )  // FIFO
Over/Underflow, sticky,
   // write-one-clear

//-------------------------------------------------------------
-----------
//                    I/O Processor Register Map
//-------------------------------------------------------------
-----------

// Chip Control Registers (DSP IOPAGE=0x00)
```

```
#define   SYSCON    0x00   // Chip Mode/Status Register
#define   PWRCFG0   0x02   // Function 0 Power Management
#define   PWRCFG1   0x04   // Function 1 Power Management
#define   PWRCFG2   0x06   // Function 2 Power Management
#define   PWRP0     0x08   // DSP 0 Interrupt/Power down
#define   PWRP1     0x0A   // DSP 1 Interrupt/Power down
#define   PLLCTL    0x0C   // DSP PLL Control
#define   REVID    0x0E   // ADSP-2192 Revision ID (read only)

//**************************************************
//   SYSCON register
//**************************************************

// Bit Positions
#define SCON_PCIRST_P 15 // PCI Reset
#define SCON_VAUX_P 14 // Vaux Present
#define SCON_PCI_5V_P 13 // PCI 5V level
#define SCON_BUS1_P 11 // Bus Mode
#define SCON_BUS0_P 10 // Bus Mode
#define SCON_CRST1_P 9 // Chip Reset Source
#define SCON_CRST0_P 8 // Chip Reset Source
#define SCON_REGD_P  7  // 2.5V Regulator Control Disable
#define SCON_VXPD_P 6  // Vaux Policy for AC'97 Pad Drivers
#define SCON_VXPW_P 5  // Vaux Policy for AC'97 Pad Well Bias
#define SCON_ACVX_P 4 // AC'97 External Devices Vaux Powered
#define SCON_XON_P 3 // XTAL Force On
#define SCON_RDIS_P 2 // Reset Disable
#define SCON_RST_P 0 // Soft Chip Reset

// Bit Masks
#define SCON_PCIRST    MK_BMSK_(SCON_PCIRST_P) // PCI Reset
#define SCON_VAUX     MK_BMSK_(SCON_VAUX_P  ) // Vaux Present
#define SCON_PCI_5V    MK_BMSK_(SCON_PCI_5V_P) // PCI 5V level
#define SCON_BUS1      MK_BMSK_(SCON_BUS1_P  ) // Bus Mode
#define SCON_BUS0      MK_BMSK_(SCON_BUS0_P  ) // Bus Mode
#define SCON_CRST1     MK_BMSK_(SCON_CRST1_P ) // Chip Reset
Source
#define SCON_CRST0     MK_BMSK_(SCON_CRST0_P ) // Chip Reset
Source
#define SCON_REGD    MK_BMSK_(SCON_REGD_P  ) // 2.5V Regulator
   // Control Disable
#define SCON_VXPD MK_BMSK_(SCON_VXPD_P) // Vaux Policy for
AC'97
   // Pad Drivers
```

```
#define SCON_VXPW     MK_BMSK_(SCON_VXPW_P) // Vaux Policy for
AC'97
   // Pad Well Bias
#define SCON_ACVX     MK_BMSK_(SCON_ACVX_P) // AC'97 External
Devices
   // Vaux Powered
#define SCON_XON       MK_BMSK_(SCON_XON_P) // XTAL Force On
#define SCON_RDIS      MK_BMSK_(SCON_RDIS_P) // Reset Disable
#define SCON_RST      MK_BMSK_(SCON_RST_P) // Soft Chip Reset

//**************************************************
//   PWRPx register
//**************************************************

// Bit Positions
#define PWRP_AINT_P 13  // DSP Interrupt Pending from AC'97
Input
#define PWRP_PMINT_P 12  // Power Management Interrupt Pending
#define PWRP_GIEN_P 10  // DSP Interrupt Enable for GPIO Input
#define PWRP_GWE_P 6 // DSP Wake up on GPIO Input Enable
#define PWRP_PMWE_P 4  // Power Management Wake up Enable
#define PWRP_RSTD_P 2 // DSP Soft Reset
#define PWRP_PU_P 1 // DSP Power Up
#define PWRP_PD_P 0 // DSP Power Down

// Bit Masks
#define PWRP_AINT    MK_BMSK_(PWRP_AINT_P)   // DSP Interrupt
Pending
   // from AC'97 Input
#define PWRP_PMINT    MK_BMSK_(PWRP_PMINT_P)   // Power
Management
   // Interrupt Pending
#define PWRP_GIEN    MK_BMSK_(PWRP_GIEN_P)   // DSP Interrupt
Enable
   // for GPIO Input
#define PWRP_GWE      MK_BMSK_(PWRP_GWE_P  )   // DSP Wake up
on GPIO
   // Input Enable
#define PWRP_PMWE     MK_BMSK_(PWRP_PMWE_P )   // Power
Management
   // Wake up Enable
#define PWRP_RSTD    MK_BMSK_(PWRP_RSTD_P )   // DSP Soft Reset
#define PWRP_PU       MK_BMSK_(PWRP_PU_P   )   // DSP Power Up
#define PWRP_PD      MK_BMSK_(PWRP_PD_P   )   // DSP Power Down
```

```
//************************************************
//   PLLCTL register
//************************************************

// Bit Positions
#define PLLC_DPLLN1_P  11                     // DSP PLL N
Divisor Selects
#define PLLC_DPLLN0_P  10                     // DSP PLL N
Divisor Selects
#define PLLC_DPLLK1_P   9                     // DSP PLL K
Divisor Selects
#define PLLC_DPLLK0_P   8                     // DSP PLL K
Divisor Selects
#define PLLC_DPLLM3_P   7                     // DSP PLL M
Divisor Selects
#define PLLC_DPLLM2_P   6                     // DSP PLL M
Divisor Selects
#define PLLC_DPLLM1_P   5                     // DSP PLL M
Divisor Selects
#define PLLC_DPLLM0_P   4                     // DSP PLL M
Divisor Selects
#define PLLC_DADJ_P     0                   // DSP PLL Adjust

// Bit Masks
#define PLLC_DPLLN1    MK_BMSK_(PLLC_DPLLN1_P) // DSP PLL N
Divisor Selects
#define PLLC_DPLLN0    MK_BMSK_(PLLC_DPLLN0_P) // DSP PLL N
Divisor Selects
#define PLLC_DPLLK1    MK_BMSK_(PLLC_DPLLK1_P) // DSP PLL K
Divisor Selects
#define PLLC_DPLLK0    MK_BMSK_(PLLC_DPLLK0_P) // DSP PLL K
Divisor Selects
#define PLLC_DPLLM3    MK_BMSK_(PLLC_DPLLM3_P) // DSP PLL M
Divisor Selects
#define PLLC_DPLLM2    MK_BMSK_(PLLC_DPLLM2_P) // DSP PLL M
Divisor Selects
#define PLLC_DPLLM1    MK_BMSK_(PLLC_DPLLM1_P) // DSP PLL M
Divisor Selects
#define PLLC_DPLLM0    MK_BMSK_(PLLC_DPLLM0_P) // DSP PLL M
Divisor Selects
#define PLLC_DADJ     MK_BMSK_(PLLC_DADJ_P)   // DSP PLL Adjust

//************************************************
```

```
//   PWRCFGx register
//**************************************************

// Bit Positions
#define PWRC_SPME_P 14 // Power Management Event Set
#define PWRC_GPME_P 6  // GPIO Power Management Event Enable
#define PWRC_PWRST1_P 1 // PCI Function Power State
#define PWRC_PWRST0_P 0 // PCI Function Power State

// Bit Masks
#define PWRC_SPME MK_BMSK_(PWRC_SPME_P) // DSP PLL N Divisor
Selects
#define PWRC_GPME MK_BMSK_(PWRC_GPME_P) // DSP PLL N Divisor
Selects
#define PWRC_PWRST1 MK_BMSK_(PWRC_PWRST1_P) // DSP PLL K
Divisor Selects
#define PWRC_PWRST0 MK_BMSK_(PWRC_PWRST0_P) // DSP PLL K
Divisor Selects

//------------------------------------------------------------
-----------
//          System Register address definitions
//------------------------------------------------------------
-----------

#define DMAPAGE      0x0C    // DMA Page Register

#define STCTL0      0x10    // FIFO0 Transmit Control Register
#define SRCTL0      0x11    // FIFO0 Receive Control Register
#define TX0        0x12   // FIFO0 Transmit Data (TX) register
#define RX0         0x13    // FIFO0 Receive Data (RX) register

#define STCTL1      0x20    // FIFO1 Transmit Control Register
#define SRCTL1      0x21    // FIFO1 Receive Control Register
#define TX1        0x22   // FIFO1 Transmit Data (TX) register
#define RX1        0x23    // FIFO1 Receive Data (RX) register

#define TPERIOD      0x30    // Timer Period Register
#define TCOUNT       0x31    // Timer Counter Register
#define TSCALE       0x32    // Timer Scaling Register
#define TSCALECNT   0x33    // Timer Scale Count Register

#define FLAGS       0x34    // Flags Register
```

```
#define MASTADDR    0x44   // DMA Address,      DSP Master DMA
#define MASTNXTADDR 0x45   // DMA Next Address,  DSP Master DMA
#define MASTCNT     0x46   // DMA Count,        DSP Master DMA
#define MASTCURCNT  0x47   // DMA Current Count, DSP Master DMA

#define TX0ADDR     0x48   // DMA Address,      Fifo0 Transmit
#define TX0NXTADDR  0x49   // DMA Next Address,  Fifo0 Transmit
#define TX0CNT      0x4A   // DMA Count,        Fifo0 Transmit
#define TX0CURCNT   0x4B   // DMA Current Count, Fifo0 Transmit

#define RX0ADDR     0x4C   // DMA Address,      Fifo0 Receive
#define RX0NXTADDR  0x4D   // DMA Next Address,  Fifo0 Receive
#define RX0CNT      0x4E   // DMA Count,        Fifo0 Receive
#define RX0CURCNT   0x4F   // DMA Current Count, Fifo0 Receive

#define TX1ADDR     0x50   // DMA Address,      Fifo1 Transmit
#define TX1NXTADDR  0x51   // DMA Next Address,  Fifo1 Transmit
#define TX1CNT      0x52   // DMA Count,        Fifo1 Transmit
#define TX1CURCNT   0x53   // DMA Current Count, Fifo1 Transmit

#define RX1ADDR     0x54   // DMA Address,      Fifo1 Receive
#define RX1NXTADDR  0x55   // DMA Next Address,  Fifo1 Receive
#define RX1CNT      0x56   // DMA Count,        Fifo1 Receive
#define RX1CURCNT   0x57   // DMA Current Count, Fifo1 Receive

// GPIO Control Registers (DSP IOPAGE=0x00)
#define GPIOCFG 0x10 // PIO Config Direction Control (1 = in,
0 = out)
#define GPIOPOL 0x12 // GPIO Polarity
   //(Inputs: 0 = active hi, 1 = active lo;
   //Outputs: 0 = CMOS, 1 = Open Drain)
#define GPIOSTKY 0x14 // GPIO Sticky: 1 = sticky, 0 = not sticky
#define GPIOWAKECTL 0x16 // GPIO Wake Control
   // 1 = wake-up enabled (requires sticky set)
#define GPIOSTAT 0x18 //GPIO Status (Read = Pin state;
   // Write: 0 = clear sticky status, 1 = no effect)
#define GPIOCTL 0x1A // GPIO Control(w), Init(r)
   //(Read = Power-on state; Write : Set state of output pins)
#define GPIOPUP  0x1C // GPIO Pull-up Pull-up enable (if
input):
   // 1 = enable, 0 = hi-Z
#define GPIOPDN 0x1E // GPIO Pull-down Pull-down enable (if
input):
   //1 = enable, 0 = hiZ
```

```
// PCI/USB Mailbox Registers (DSP IOPAGE=0x00)
#define MBXSTAT       0x20   //  Mailbox Status Mailbox Status
#define MBXCTL         0x22    //  Mailbox Control Mailbox
Interrupt Control
#define MBX_IN0        0x24    //  Incoming Mailbox 0 PCI/USB
to DSP mailbox
#define MBX_IN1        0x26    //  Incoming Mailbox 1 PCI/USB
to DSP mailbox
#define MBX_OUT0       0x28    //  Outgoing Mailbox 0 DSP to
PCI/USB mailbox
#define MBX_OUT1       0x2A    //  Outgoing Mailbox 0 DSP to
PCI/USB mailbox

// SERIAL EEPROM Control Register (DSP IOPAGE=0x00)
#define SPROMCTL 0x30 // Serial EEPROM I/O Control/Status
   // Direction and status  for SEN, SCK, SDA pins.

// JTAG ID Registers(DSP IOPAGE=0x00)
#define JTAGIDL 0xA0 // IDCODE[15:0]  JTAG ID0 :
   // Value = 0xA1CB ( Read Only ).
#define JTAGIDH 0xA2 // IDCODE[31:16] JTAG ID1 :
   // Value = 0x0278 ( Read Only ).

//  AC'97 Control Registers (DSP IOPAGE=0x00)
#define AC97LCTL       0xC0    //  AC'97 Link Control
#define AC97LSTAT      0xC2    //  AC'97 Link Status
#define AC97SEN        0xC4    //  AC'97 Slot Enable
#define AC97SVAL       0xC6    //  AC'97 Input Slot Valid
#define AC97SREQ       0xC8    //  AC'97 Slot Request
#define AC97GPIO      0xCA   //  AC'97 External GPIO Register

//  AC'97 External Codec IO Register Spaces

#define AC97CODEC0     0x04    //  External Primary   Codec 0
   // IOPAGE space registers (0x00 - 0x7F)
#define AC97CODEC1     0x05    //  External Secondary Codec 1
   // IOPAGE space registers (0x00 - 0x7F)
#define AC97CODEC2     0x06    //  External Secondary Codec 2
   // IOPAGE space registers (0x00 - 0x7F)

// CardBus Function Event Registers (DSP IOPAGE=0x01)

#define CB_EVENT0       0x00    //  Function 0 Event
```

```
#define CB_EVENTMASK0   0x04    //  Function 0 Event Mask
#define CB_PSTATE0      0x08    //  Function 0 Present State
#define CB_EVENTFORCE0  0x0C    //  Function 0 Event Force

#define CB_EVENT1       0x10    //  Function 1 Event
#define CB_EVENTMASK1   0x14    //  Function 1 Event Mask
#define CB_PSTATE1      0x18    //  Function 1 Present State
#define CB_EVENTFORCE1  0x1C    //  Function 1 Event Force

#define CB_EVENT2       0x20    //  Function 2 Event
#define CB_EVENTMASK2   0x24    //  Function 2 Event Mask
#define CB_PSTATE2      0x28    //  Function 2 Present State
#define CB_EVENTFORCE2  0x2C    //  Function 2 Event Force

// PCI DMA Address/Count Registers (DSP IOPAGE=0x08)

#define PCI_Rx0BADDRL   0x00   //  Rx0 DMA Base Address
Bits 15:0
#define PCI_Rx0BADDRH   0x02   //  Rx0 DMA Base Address
Bits 31:16
#define PCI_Rx0CURADDRL 0x04   //  Rx0 DMA Current Address
Bits 15:0
#define PCI_Rx0CURADDRH 0x06   //  Rx0 DMA Current Address
Bits 31:16
#define PCI_Rx0BCNTL    0x08   //  Rx0 DMA Base Count
Bits 15:0
#define PCI_Rx0BCNTH    0x0A   //  Rx0 DMA Base Count
Bits 31:16
#define PCI_Rx0CURCNTL  0x0C   //  Rx0 DMA Current Count
Bits 15:0
#define PCI_Rx0CURCNTH  0x0E   //  Rx0 DMA Current Count
Bits 31:16

#define PCI_Tx0BADDRL   0x10   //  Tx0 DMA Base Address
Bits 15:0
#define PCI_Tx0BADDRH   0x12   //  Tx0 DMA Base Address
Bits 31:16
#define PCI_Tx0CURADDRL 0x14   //  Tx0 DMA Current Address
Bits 15:0
#define PCI_Tx0CURADDRH 0x16   //  Tx0 DMA Current Address
Bits 31:16
#define PCI_Tx0BCNTL    0x18   //  Tx0 DMA Base Count
Bits 15:0
```

```
#define PCI_Tx0BCNTH    0x1A   //  Tx0 DMA Base Count
Bits 31:16
#define PCI_Tx0CURCNTL  0x1C   //  Tx0 DMA Current Count
Bits 15:0
#define PCI_Tx0CURCNTH  0x1E   //  Tx0 DMA Current Count
Bits 31:16

#define PCI_Rx1BADDRL   0x20   //  Rx1 DMA Base Address
Bits 15:0
#define PCI_Rx1BADDRH   0x22   //  Rx1 DMA Base Address
Bits 31:16
#define PCI_Rx1CURADDRL 0x24   //  Rx1 DMA Current Address
Bits 15:0
#define PCI_Rx1CURADDRH 0x26   //  Rx1 DMA Current Address
Bits 31:16
#define PCI_Rx1BCNTL    0x28   //  Rx1 DMA Base Count
Bits 15:0
#define PCI_Rx1BCNTH    0x2A   //  Rx1 DMA Base Count
Bits 31:16
#define PCI_Rx1CURCNTL  0x2C   //  Rx1 DMA Current Count
Bits 15:0
#define PCI_Rx1CURCNTH  0x2E   //  Rx1 DMA Current Count
Bits 31:16

#define PCI_Tx1BADDRL   0x30   //  Tx1 DMA Base Address
Bits 15:0
#define PCI_Tx1BADDRH   0x32   //  Tx1 DMA Base Address
Bits 31:16
#define PCI_Tx1CURADDRL 0x34   //  Tx1 DMA Current Address
Bits 15:0
#define PCI_Tx1CURADDRh 0x36   //  Tx1 DMA Current Address
Bits 31:16
#define PCI_Tx1BCNTL    0x38   //  Tx1 DMA Base Count
Bits 15:0
#define PCI_Tx1BCNTH    0x3A   //  Tx1 DMA Base Count
Bits 31:16
#define PCI_Tx1CURCNTL  0x3C   //  Tx1 DMA Current Count
Bits 15:0
#define PCI_Tx1CURCNTH  0x3E   //  Tx1 DMA Current Count
Bits 31:16

#define PCI_Rx0IRQCNTL  0x40   //  Rx0 DMA Interrupt Count
Bits 15:0
```

```
#define PCI_Rx0IRQCNTH  0x42   //  Rx0 DMA Interrupt Count
Bits 23:16
#define PCI_Rx0IRQBCNTL 0x44   //  Rx0 DMA Interrupt Base Count
Bits 15:0
#define PCI_Rx0IRQBCNTH 0x46   //  Rx0 DMA Interrupt Base Count
Bits 23:16

#define PCI_Tx0IRQCNTL  0x48   //  Tx0 DMA Interrupt Count
Bits 15:0
#define PCI_Tx0IRQCNTH  0x4A   //  Tx0 DMA Interrupt Count
Bits 23:16
#define PCI_Tx0IRQBCNTL 0x4C   //  Tx0 DMA Interrupt Base Count
Bits 15:0
#define PCI_Tx0IRQBCNTH 0x4E   //  Tx0 DMA Interrupt Base Count
Bits 23:16

#define PCI_Rx1IRQCNTL  0x50   //  Rx1 DMA Interrupt Count
Bits 15:0
#define PCI_Rx1IRQCNTH  0x52   //  Rx1 DMA Interrupt Count
Bits 23:16
#define PCI_Rx1IRQBCNTL 0x54   //  Rx1 DMA Interrupt Base Count
Bits 15:0
#define PCI_Rx1IRQBCNTH 0x56   //  Rx1 DMA Interrupt Base Count
Bits 23:16

#define PCI_Tx1IRQCNTL  0x58   //  Tx1 DMA Interrupt Count
Bits 15:0
#define PCI_Tx1IRQCNTH  0x5A   //  Tx1 DMA Interrupt Count
Bits 23:16
#define PCI_Tx1IRQBCNTL 0x5C   //  Tx1 DMA Interrupt Base Count
Bits 15:0
#define PCI_Tx1IRQBCNTH 0x5E   //  Tx1 DMA Interrupt Base Count
Bits 23:16

#define PCI_Rx0CTL      0x60   //  Rx0 DMA PCI Control/Status
#define PCI_Tx0CTL      0x68   //  Tx0 DMA PCI Control/Status
#define PCI_Rx1CTL      0x70   //  Rx1 DMA PCI Control/Status
#define PCI_Tx1CTL      0x78   //  Tx1 DMA PCI Control/Status

#define PCI_MSTRCNT0    0x80   //  DMA Transfer Count0 Bus
master
   // sample transfer count 0.
#define PCI_MSTRCNT1    0x82   //  DMA Transfer Count1 Bus
master
```

```
   // sample transfer count 1.
#define PCI_DMAC0       0x84   //  DMA Control0 Bus master
control
   // and status 0.
#define PCI_DMAC1       0x86   //  DMA Control1 Bus master
control
   // and status 1.
#define PCI_IRQSTAT    0x88  //  PCI Interrupt Register Status
bits
   for all PCI interrupt sources.
#define PCI_CFGCTL      0x8A   //  PCI Control Includes config
   // register read/write control.

//  PCI FUNCTION 0 Configuration Space Registers (DSP
IOPAGE=0x09)

// Note: Access to these registers is controlled by the PCI RDY
bit in the
// Chip Mode/Status Register (Page 0x00, Address 0x00).

#define PCI_VendorID0   0x00   // Configuration 0    Vendor ID
#define PCI_DeviceID0   0x02   // Configuration 0    Device ID
#define PCI_ClassCODE0L 0x08   // Configuration 0    Class
Code[7:0],Rev ID
#define PCI_ClassCODE0H 0x0A   // Configuration 0    Class
Code[23:8]
#define PCI_SVendorID0  0x2C   // Configuration 0    Subsystem
Vendor ID
#define PCI_SDeviceID0  0x2E   // Configuration 0    Subsystem
Device ID
#define PCI_PWRMT0      0x44   // Configuration 0
   // Power Mgt Capabilities Bit 15 set if Vaux is sensed valid.

//  PCI FUNCTION 1 Configuration Space Registers (DSP
IOPAGE=0x0A)

// Note: Access to these registers is controlled by the PCI RDY
bit in the
// Chip Mode/Status Register (Page 0x00, Address 0x00).

#define PCI_VendorID1   0x00   // Configuration 1    Vendor ID
#define PCI_DeviceID1   0x02   // Configuration 1    Device ID
#define PCI_ClassCODE1L 0x08   // Configuration 1    Class
Code[7:0], Rev ID
```

```
#define PCI_ClassCODE1H 0x0A   // Configuration 1    Class
Code[23:8]
#define PCI_SvendorID1  0x2C   // Configuration 1    Subsystem
Vendor ID
#define PCI_SdeviceID1  0x2E   // Configuration 1    Subsystem
Device ID
#define PCI_PWRMT1      0x44   // Configuration 1
   // Power Mgt Capabilities Bit 15 set if Vaux is sensed valid.

//  PCI FUNCTION 2 Configuration Space Registers  (DSP
IOPAGE=0x0B)

// Note: Access to these registers is controlled by the PCI RDY
bit in the
// PCI Interrupt Control Register ( Page 0x08, Address 0xA2 ).

#define PCI_VendorID2   0x00   // Configuration 2   Vendor ID
#define PCI_DeviceID2   0x02   // Configuration 2   Device ID
#define PCI_ClassCODE2L 0x08   // Configuration 2   Class
Code[7:0],Rev ID
#define PCI_ClassCODE2H 0x0A   // Configuration 2   Class
Code[23:8]
#define PCI_SvendorID2  0x2C   // Configuration 2   Subsystem
Vendor ID
#define PCI_SdeviceID2  0x2E   // Configuration 2   Subsystem
Device ID
#define PCI_PWRMT2      0x44   // Configuration 2   Power Mgt
    // Capabilities Bit 15 set if Vaux is sensed valid.

// USB Endpoint DMA Control Registers (DSP IOPAGE=0x0C)

#define USB_EP4_ADDR   0x00    // Memory Buffer Base Addr. EP4
#define USB_EP4_SIZE   0x04    // Memory Buffer Size EP4
#define USB_EP4_RD     0x06    // Memory Buffer RD Offset EP4
#define USB_EP4_WR     0x08    // Memory Buffer WR Offset EP4

#define USB_EP5_ADDR   0x10    // Memory Buffer Base Addr. EP5
#define USB_EP5_SIZE   0x14    // Memory Buffer Size EP5
#define USB_EP5_RD     0x16    // Memory Buffer RD Offset EP5
#define USB_EP5_WR     0x18    // Memory Buffer WR Offset EP5

#define USB_EP6_ADDR   0x20    // Memory Buffer Base Addr. EP6
#define USB_EP6_SIZE   0x24    // Memory Buffer Size EP6
#define USB_EP6_RD     0x26    // Memory Buffer RD Offset EP6
```

```
#define USB_EP6_WR      0x28    // Memory Buffer WR Offset EP6

#define USB_EP7_ADDR    0x30    // Memory Buffer Base Addr. EP7
#define USB_EP7_SIZE    0x34    // Memory Buffer Size EP7
#define USB_EP7_RD      0x36    // Memory Buffer RD Offset EP7
#define USB_EP7_WR      0x38    // Memory Buffer WR Offset EP7

#define USB_EP8_ADDR    0x40    // Memory Buffer Base Addr. EP8
#define USB_EP8_SIZE    0x44    // Memory Buffer Size EP8
#define USB_EP8_RD      0x46    // Memory Buffer RD Offset EP8
#define USB_EP8_WR      0x48    // Memory Buffer WR Offset EP8

#define USB_EP9_ADDR    0x50    // Memory Buffer Base Addr. EP9
#define USB_EP9_SIZE    0x54    // Memory Buffer Size EP9
#define USB_EP9_RD      0x56    // Memory Buffer RD Offset EP9
#define USB_EP9_WR      0x58    // Memory Buffer WR Offset EP9

#define USB_EP10_ADDR   0x60    // Memory Buffer Base Addr. EP10
#define USB_EP10_SIZE   0x64    // Memory Buffer Size EP10
#define USB_EP10_RD     0x66    // Memory Buffer RD Offset EP10
#define USB_EP10_WR     0x68    // Memory Buffer WR Offset EP10

#define USB_EP11_ADDR   0x70    // Memory Buffer Base Addr. EP11
#define USB_EP11_SIZE   0x74    // Memory Buffer Size EP11
#define USB_EP11_RD     0x76    // Memory Buffer RD Offset EP11
#define USB_EP11_WR     0x78    // Memory Buffer WR Offset EP11

#endif
```

C NUMERIC FORMATS

Overview

ADSP-219x family processors support 16-bit fixed-point data in hardware. Special features in the computation units allow you to support other formats in software. This appendix describes various aspects of the 16-bit data format. It also describes how to implement a block floating-point format in software.

Un/Signed: Twos-Complement Format

Unsigned binary numbers may be thought of as positive, having nearly twice the magnitude of a signed number of the same length. The least significant words of multiple precision numbers are treated as unsigned numbers.

Signed numbers supported by the ADSP-219x family are in twos-complement format. Signed-magnitude, ones-complement, BCD or excess-n formats are not supported.

Integer or Fractional

The ADSP-219x family supports both fractional and integer data formats. In an integer, the radix point is assumed to lie to the right of the LSB, so that all magnitude bits have a weight of 1 or greater. This format is shown in Figure C-1, which can be found on the following page. Note that in twos-complement format, the sign bit has a negative weight.

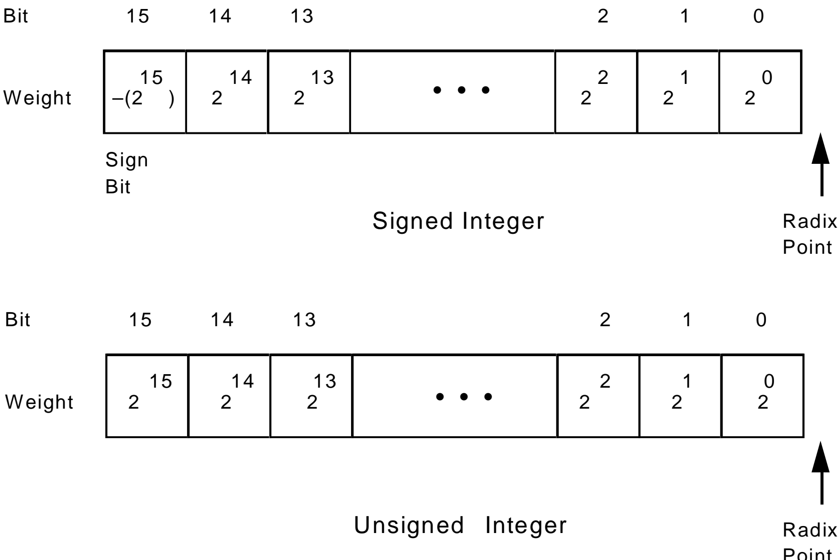


Figure C-1. Integer Format

In a fractional format, the assumed radix point lies within the number, so that some or all of the magnitude bits have a weight of less than 1. In the format shown in Figure C-2 on page C-3, the assumed radix point lies to the left of the 3 LSBs, and the bits have the weights indicated.

The notation used to describe a format consists of two numbers separated by a period (.); the first number is the number of bits to the left of radix point, the second is the number of bits to the right of the radix point. For example, 16.0 format is an integer format; all bits lie to the left of the radix point. The format in Figure C-2 is 13.3.

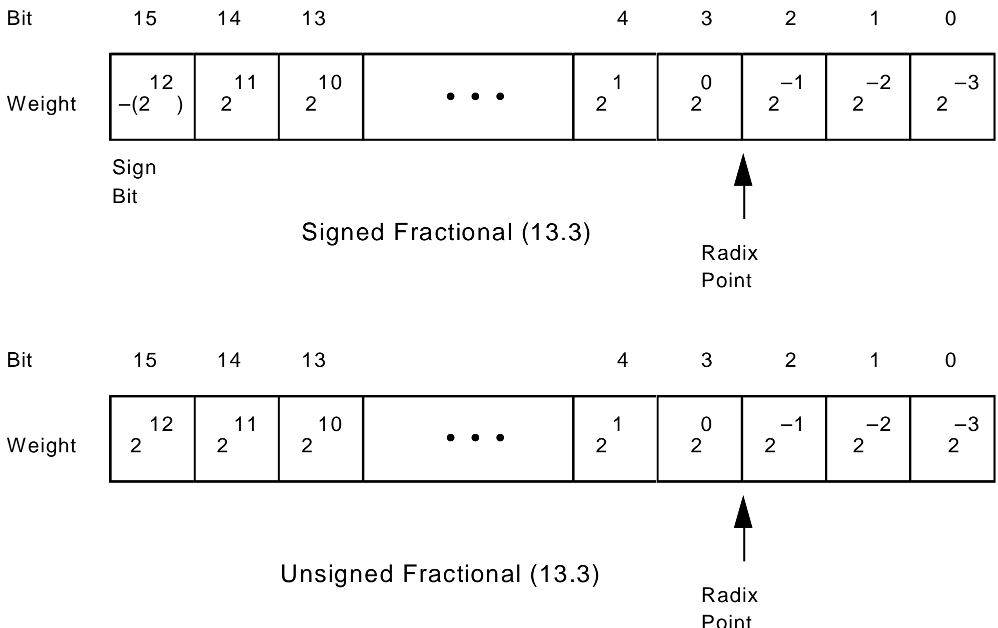


Figure C-2. Example Of Fractional Format

Table C-1 shows the ranges of numbers representable in the fractional formats that are possible with 16 bits.

Table C-1. Fractional Formats And Their Ranges

Format	# of Integer Bits	# of Fractional Bits	Max Positive Value (0x7FFF) In Decimal	Max Negative Value (0x8000) In Decimal	Value of 1 LSB (0x0001) In Decimal
1.15	1	15	0.999969482421875	–1.0	0.000030517578125
2.14	2	14	1.999938964843750	–2.0	0.000061035156250
3.13	3	13	3.999877929687500	–4.0	0.000122070312500
4.12	4	12	7.999755859375000	–8.0	0.000244140625000
5.11	5	11	15.999511718750000	–16.0	0.000488281250000
6.10	6	10	31.999023437500000	–32.0	0.000976562500000
7.9	7	9	63.998046875000000	–64.0	0.001953125000000
8.8	8	8	127.996093750000000	–128.0	0.003906250000000
9.7	9	7	255.992187500000000	–256.0	0.007812500000000
10.6	10	6	511.984375000000000	–512.0	0.015625000000000
11.5	11	5	1023.968750000000000	–1024.0	0.031250000000000
12.4	12	4	2047.937500000000000	–2048.0	0.062500000000000
13.3	13	3	4095.875000000000000	–4096.0	0.125000000000000
14.2	14	2	8191.750000000000000	–8192.0	0.250000000000000
15.1	15	1	16383.500000000000000	–16384.0	0.500000000000000
16.0	16	0	32767.000000000000000	–32768.0	1.000000000000000

Binary Multiplication

In addition and subtraction, both operands must be in the same format (signed or unsigned, radix point in the same location) and the result format must be the same as the input format. Addition and subtraction are performed the same way whether the inputs are signed or unsigned.

In multiplication, however, the inputs can have different formats, and the result depends on their formats. The ADSP-219x family assembly language allows you to specify whether the inputs are both signed, both unsigned, or one of each (mixed-mode). The location of the radix point in the result can be derived from its location in each of the inputs. Figure C-3 illustrates this point. The product of two 16-bit numbers is a 32-bit number. If the inputs' formats are M.N and P.Q, the product has the format (M+P).(N+Q). For example, the product of two 13.3 numbers is a 26.6 number. The product of two 1.15 numbers is a 2.30 number.

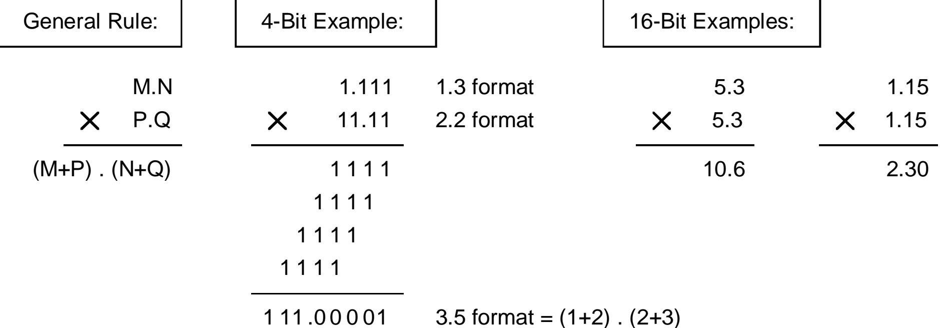


Figure C-3. Format of Multiplier Result

Fractional Mode And Integer Mode

A product of 2 twos-complement numbers has two sign bits. Since one of these bits is redundant, you can shift the entire result left one bit. Additionally, if one of the inputs was a 1.15 number, the left shift causes the result to have the same format as the other input (with 16 bits of additional precision). For example, multiplying a 1.15 number by a 5.11 number yields a 6.26 number. When shifted left one bit, the result is a 5.27 number, or a 5.11 number plus 16 LSBs.

The ADSP-219x family provides a mode (called the fractional mode) in which the multiplier result is always shifted left one bit before being written to the result register. This left shift eliminates the extra sign bit when both operands are signed, yielding a correctly formatted result.

When both operands are in 1.15 format, the result is 2.30 (30 fractional bits). A left shift causes the multiplier result to be 1.31, which can be rounded to 1.15. Thus, if you use a fractional data format, it is most convenient to use the 1.15 format.

In the integer mode, the left shift does not occur. This is the mode to use if both operands are integers (in the 16.0 format). The 32-bit multiplier result is in 32.0 format, which is also an integer.

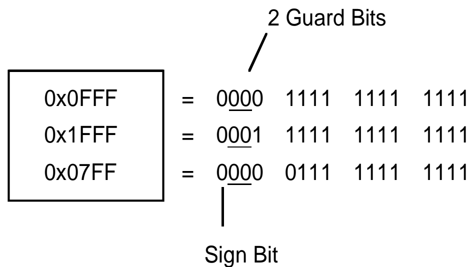
In all ADSP-219x DSPs, fractional and integer modes are controlled by a bit in the `MSTAT` register. At reset, these processors default to the fractional mode.

Block Floating-Point Format

A block floating-point format enables a fixed-point processor to gain some of the increased dynamic range of a floating-point format without the overhead needed to do floating-point arithmetic. Some additional programming is required to maintain a block floating-point format, however.

A floating-point number has an exponent that indicates the position of the radix point in the actual value. In block floating-point format, a set (block) of data values share a common exponent. To convert a block of fixed-point values to block floating-point format, you would shift each value left by the same amount and store the shift value as the block exponent. Typically, block floating-point format allows you to shift out non-significant MSBs, increasing the precision available in each value.

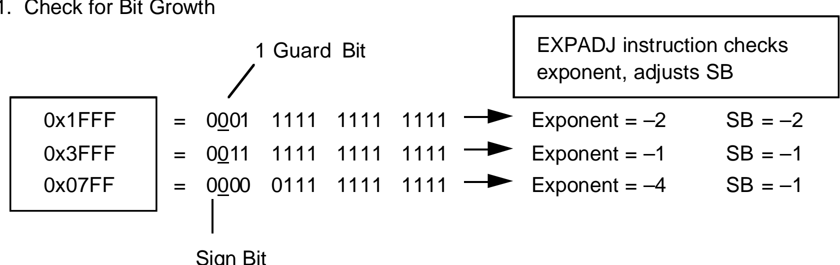
You can also use block floating-point format to eliminate the possibility of a data value overflowing. Figure C-4 shows an example. The three data samples each have at least two non-significant, redundant sign bits. Each data value can grow by these two bits (two orders of magnitude) before overflowing; thus, these bits are called guard bits. If it is known that a process will not cause any value to grow by more than these two bits, then the process can be run without loss of data. Afterward, however, the block must be adjusted to replace the guard bits before the next process.



To detect bit growth into 2 guard bits, set SB=–2

Figure C-4. Data With Guard Bits

Figure C-5 on page C-8 shows the data after processing but before adjustment. The block floating-point adjustment is performed as follows. Initially, the value of SB is –2, corresponding to the two guard bits. During processing, each resulting data value is inspected by the EXPADJ instruction, which counts the number of redundant sign bits and adjusts SB as if the number of redundant sign bits is less than 2. In this example, SB=–1 after processing, which indicates that the block of data must be shifted right one bit to maintain the two guard bits. If SB were 0 after processing, the block would have to be shifted two bits right. In either case, the block exponent is updated to reflect the shift.



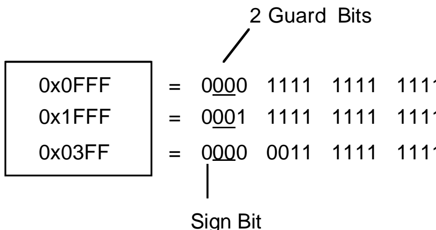


Figure C-5. Block Floating-Point Adjustment

D ADSP-2192 TIMER

Overview

The programmable interval timer can generate periodic interrupts based on multiples of the processor's cycle time. When enabled, a 16-bit count register is decremented every *n* cycles, where *n-1* is a scaling value stored in a 16-bit register. When the value of the count register reaches zero, an interrupt is generated and the count register is reloaded from a 16-bit period register (TPERIOD).

The scaling feature of the timer allows the 16-bit counter to generate periodic interrupts over a wide range of periods. Given a processor cycle time of 6.25 ns, the timer can generate interrupts with periods of 6.25 ns up to 0.4 ms with a zero scale value. When scaling is used, time periods can range up to 26.875 seconds.

Timer interrupts can be masked, cleared and forced in software if desired. For additional information, refer to "Interrupts and Sequencing" on page 3-24.

Timer Architecture

The timer includes four 16-bit registers: TCOUNT, TPERIOD, TSCALE, and TSCLCNT. The extended Mode Control instruction enables and disables the timer by setting and clearing bit 5 in the Mode Status register, MSTAT. For a description of the Mode Control instructions, refer to the *ADSP-219x DSP Instruction Set Reference*. The timer registers, which reside in the core register space of each core, are shown in Figure D-1.

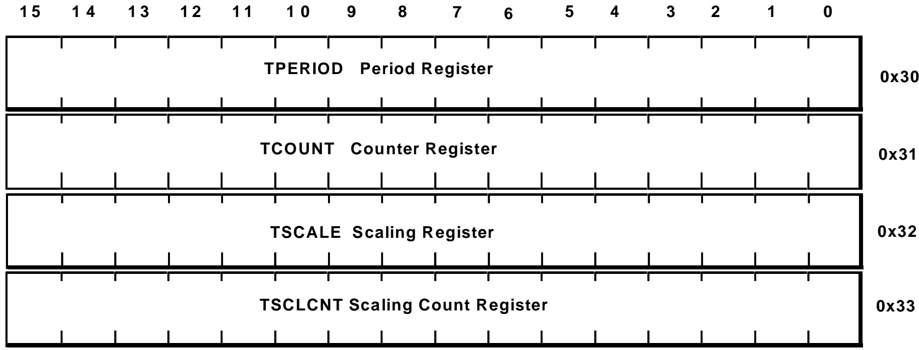


Figure D-1. Timer Registers

TCOUNT is the count register. When the timer is enabled, it is decremented as often as once every instruction cycle. When the counter reaches zero, an interrupt is generated. TCOUNT is then reloaded from the TPERIOD register and the count begins again. TSCALE stores a scaling value that is one less than the number of cycles between decrements of TCOUNT. For example, if the value in TSCALE register is 0, the counter register decrements once every instruction cycle. If the value in TSCALE is 1, the counter decrements once every two cycles. TSCLCNT holds the current scale count. An interrupt request is generated when both (TCOUNT-1) and (TSCLCNT-1) require borrows (that is, they would have a value of 0000-1).

Figure D-2 shows the timer block diagram.

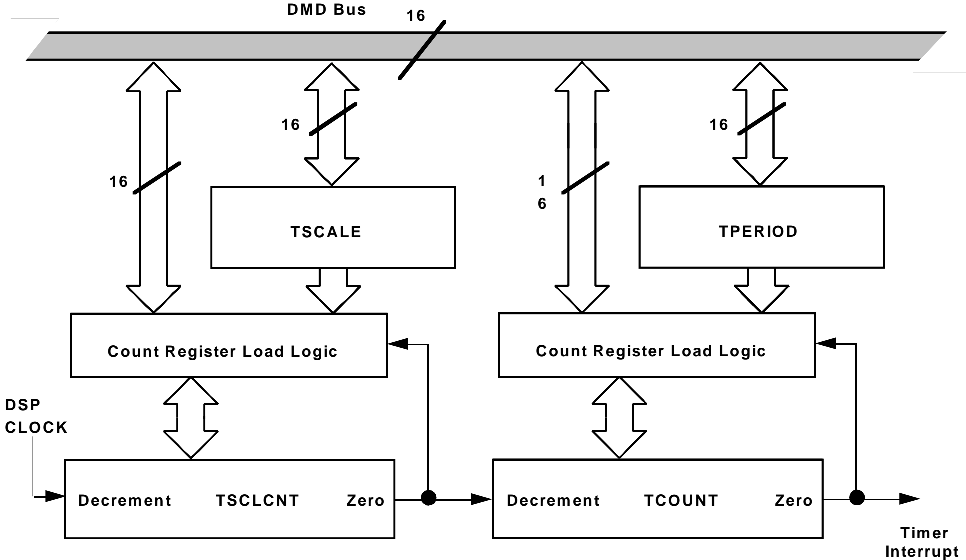


Figure D-2. Timer Block Diagram

Resolution

TSCALE provides the capability to program longer time intervals between interrupts, extending the range of the 16-bit TCOUNT register. Table D-1 shows the range and the relationship between period length and resolution for TPERIOD = maximum (65536).

Table D-1. Timer Range and Resolution

Cycle Time = 6.25 nsec		
TSCALE	**Interrupt Every…**	**Resolution**
0x0000	0.41 msec	6.25 nsec
0xFFFF	26.87 sec	0.41 µsec

Timer Operation

Table D-2 shows the effect of operating the timer with TPERIOD=5, TSCALE=1 and TCOUNT=5. After the timer is enabled (cycle n–1) the counter begins. Because TSCALE is 1, TCOUNT is decremented on every other cycle. The reloading of TCOUNT and continuation of the counting occurs, as shown, during the interrupt service routine.

Table D-2. Example Of Timer Operation

Cycle	TCOUNT	Action
n–4		TPERIOD loaded with 5
n–3		TSCALE loaded with 1
n–2		TCOUNT loaded with 5
n–1	5	ENA TIMER executed

Table D-2. Example Of Timer Operation (Cont'd)

Cycle	TCOUNT	Action
n	5	Since TSCALE = 1, no decrement
n+1	5	Decrement TCOUNT
n+2	4	No decrement
n+3	4	Decrement TCOUNT
n+4	3	No decrement
n+5	3	Decrement TCOUNT
n+6	2	No decrement
n+7	2	Decrement TCOUNT
n+8	1	No decrement
n+9	1	Decrement TCOUNT
n+10	0	No decrement
n+11	0	Zero reached, interrupt occurs load TCOUNT from TPERIOD
n+12	5	No decrement
n+13	5	Decrement TCOUNT
n+14	4	No decrement
n+15	4	Decrement TCOUNT, etc.

One interrupt occurs every (TSCALE+1) * (TPERIOD+1) clock cycles. To set the first interrupt at a different time interval from subsequent interrupts, load TCOUNT with a different value from TPERIOD. The formula for the first interrupt is (TSCLCNT+1) * (TSCALE+1) * (TCOUNT).

If you write a new value to TSCALE or TCOUNT, the change is effective immediately. If you write a new value to TPERIOD, the change does not take effect until after TCOUNT is reloaded.

Enabling the Timer

This section tells you how to enable the timer and generate interrupts. It lists the steps you need to use and provides sample code (see Listing D-1).

To enable the timer:

1. Set values for TCOUNT, TPERIOD, TSCALE, and TSCLCNT.
2. Set bit 0 in IMASK to enable interrupt.
3. Execute ENA TIMER instruction to start counting down (bit 5 in MSTAT register).

Listing D-1. Code for Enabling the Timer and Generating Interrupts

```
// init timer
ay0  = 0xf000;
ay1  = 0x0200;
reg(0x30) = ay0;     // set tperiod
reg(0x32) = ay1;     // set tscale

// enable global interrupts and kernel, mailbox, and
// timer interrupts specifically, and enable nesting
ar = icntl;
ar = setbit 4 of ar;
icntl = ar;
ena int;
imask = 0x0034;
ena ti;
```

E ADSP-2192 INTERRUPTS

Overview

The DSP has two core-to-core flags to control interrupts between the two DSPs. The interrupt controller lets the DSP respond to thirteen interrupts with minimum overhead. The controller implements an interrupt priority scheme as shown in Table E-1 on page E-1. Applications can use the unassigned slots for software and peripheral interrupts. The DSP's Interrupt Control (ICNTL) register (shown in Table E-2 on page E-3) provides controls for global interrupt enable, stack interrupt configuration, and interrupt nesting.

Peripheral Interrupts

Table E-1 shows the interrupt vector and DSP-DSP semaphores of each of the peripheral interrupts at reset. (For information about DSP-DSP semaphores, see "Using Dual-DSP Interrupts and Flags" on page 6-13.) The peripheral interrupt's position in the IMASK and IRPTL register and its vector address depend on its priority level, as shown in Table E-1.

Table E-1. Interrupt Vector Table

Bit	Priority	Interrupt	Vector Address Offset[1]
0	1	Reset (non-maskable)	0x00
1	2	Powerdown (non-maskable)	0x04

Table E-1. Interrupt Vector Table (Continued)

Bit	Priority	Interrupt	Vector Address Offset[1]
2	3	Kernel interrupt (single step)	0x08
3	4	Stack Status	0x0C
4	5	Mailbox	0x10
5	6	Timer	0x14
6	7	System Interrupt	0x18
7	8	PCI Bus Master	0x1C
8	9	DSP-DSP	0x20
9	10	FIFO0 Transmit	0x24
10	11	FIFO0 Receive	0x28
11	12	FIFO1 Transmit	0x2C
12	13	FIFO1 Receive	0x30
13	14	Reserved	0x34
14	15	Reserved	0x38
15	16	AC'97 Frame	0x3C

1 The interrupt vector address values are represented as offsets from address 0x01 0000. This address corresponds to the start of Program Memory in DSP P0 and P1.

Interrupt routines can either be nested with higher priority interrupts taking precedence or they can be processed sequentially. Interrupts can be masked or unmasked with the `IMASK` register. Individual interrupt requests are logically ANDed with the bits in `IMASK`; the highest priority unmasked interrupt is then selected. The emulation, power down, and reset interrupts are nonmaskable with the `IMASK` register, but software can use the `DIS INT` instruction to mask the power down interrupt.

Table E-2. Interrupt Control (ICNTL) Register Bits

Bit	Description
0–3	Reserved
4	Interrupt nesting enable
5	Global interrupt enable
6	Reserved
7	MAC biased rounding enable
8–9	Reserved
10	PC stack interrupt enable
11	Loop stack interrupt enable
12	Low power idle enable
13–15	Reserved

The IRPTL register is used to force and clear interrupts. On-chip stacks preserve the processor status and are automatically maintained during interrupt handling. To support interrupt, loop, and subroutine nesting, the PC stack is 33 levels deep, the loop stack is eight-levels deep, and the status stack is 16 levels deep. To prevent stack overflow, the PC stack can generate a stack level interrupt if the PC stack falls below 3 locations full or rises above 28 locations full.

The following instructions globally enable or disable interrupt servicing, regardless of the state of IMASK.

```
Ena Int;
Dis Int;
```

At reset, interrupt servicing is disabled.

For quick servicing of interrupts, a secondary set of DAG and computational registers exist. Switching between the primary and secondary registers lets programs quickly service interrupts while preserving the DSP's state.

For more information about interrupts, refer to "Using Dual-DSP Interrupts and Flags" on page 6-13 and "ADSP-2192 Interrupts" on page E-1.

Other Interrupt Types

The programmable interval timer generates periodic interrupts. See "ADSP-2192 Timer" on page D-1 for more information about the programmable interval timer and its four registers (TCOUNT, TSCALE, TPERIOD, and TSCLCNT).

Bits in the PCI Control/Status register control whether an interrupt occurs when the EOL is reached or when the FLAG bit is set. See "ADSP-2192 DSP Peripheral Registers" on page B-1 for more information about these bits.

Interrupts to the DSP can also be generated by DMA (in regular or scatter-gather modes), by PCI, when some words have been received in the input FIFOs, or when the Transmit FIFOs are empty. Internal interrupts, including serial EEPROM port, PCI, USB, AC'97, Sub-ISA, timer, and DMA interrupts, are discussed elsewhere in this book. ("Host (PCI/USB) Port" on page 8-1 discusses USB, PCI, Sub-ISA, serial EEPROM, and DMA; "AC'97 Codec Port" on page 9-1 discusses the AC'97 interface.) Additional information about interrupt masking, set up, and operation can be found in "Interrupts and Sequencing" on page 3-24.

G GLOSSARY

Terms

Arithmetic Logic Unit (ALU). This part of a processing element performs arithmetic and logic operations on fixed-point data.

Asynchronous transfers. Asynchronous host accesses of the DSP. After acquiring control of the DSP's external bus, the host must assert the $\overline{\text{CS}}$ pin of the DSP it wants to access.

Base address. The starting address of a circular buffer to which the DAG wraps around. This address is stored in a DAG `Bx` register.

Base registers. A base (`Bx`) register is a Data Address Generator (DAG) register that sets up the starting address for a circular buffer.

Bit-reverse addressing. The Data Address Generator (DAG) provides a bit-reversed address during a data move without reversing the stored address.

Circular buffer addressing. The DAG uses the `Ix`, `Mx`, `Lx`, and `Bx` register settings to constrain addressing to a range of addresses. This range contains data that the DAG steps through repeatedly, "wrapping around" to repeat stepping through the range of addresses in a circular pattern.

Companding (compressing/expanding). This is the process of logarithmically encoding and decoding data to minimize the number of bits that must be sent.

Conditional branches. These are Jump or Call/return instructions whose execution is based on testing an If condition.

Data Address Generators. The data address generators (DAGs) provide memory addresses when data is transferred between memory and registers.

Data register file. This is the set of data registers that transfer data between the data buses and the computation units. These registers also provide local storage for operands.

Data registers (Dregs). These are registers in the computational units. These registers are hold operands for multiplier, ALU, or shifter operations.

Delayed branches. These are Jumps and Call/return instructions with the delayed branches `(DB)` modifier. In delayed branches, two (instead of four) instruction cycles are lost in the pipeline, because the DSP executes the two instructions after the branch while the pipeline fills with instructions from the new branch.

Direct branches. These are Jump or Call/return instructions that use an absolute—not changing at runtime—address (such as a program label) or use a PC-relative address.

DMA (Direct Memory Accessing). The DSP's I/O processor supports DMA of data between DSP memory and peripherals through the host port or AC'97 serial port. Each DMA operation transfers an entire block of data.

DMA chaining. The DSP supports chaining together multiple DMA sequences. In chained DMA, the I/O processor loads the next Transfer Control Block (DMA parameters) into the DMA parameter registers when the current DMA finishes and auto-initializes the next DMA sequence.

DMA Parameter Registers. These registers function similarly to data address generator registers, setting up a memory access process.

DMA TCB chain loading. This is the process that the I/O processor uses for loading the TCB of the next DMA sequence into the parameter registers during chained DMA.

Edge-sensitive interrupt. The DSP detects this type of interrupt if the input signal is high (inactive) on one cycle and low (active) on the next cycle when sampled on the rising edge of XTALI.

Endian Format, Little Versus Big. The DSP uses big-endian format—moves data starting with the most-significant-bit and finishing with the least-significant-bit—in almost all instances. The two exceptions are bit order for data transfer through the serial port and word order for packing through the external port. For compatibility with little-endian (least-significant-first) peripherals, the DSP supports both big- and little-endian bit order data transfers. Also for compatibile little endian hosts, the DSP supports both big- and little-endian word order data transfers.

Flag update. The DSP's update to status flags occurs at the end of the cycle in which the status is generated and is available on the next cycle.

Harvard architecture. DSPs use memory architectures that have separate buses for program and data storage. The two buses let the DSP get a data word and an instruction simultaneously.

I/O processor register. One of the control, status, or data buffer registers of the DSP's on-chip I/O processor.

Idle. An instruction that causes the processor to cease operations, holding its current state until an interrupt occurs. Then, the processor services the interrupt and continues normal execution.

Index registers. An index register is a Data Address Generator (DAG) register that holds an address and acts as a pointer to memory.

Indirect branches. These are Jump or Call/return instructions that use a dynamic—changes at runtime—address that comes from the PM data address generator.

Interrupts. Subroutines in which a runtime event (not an instruction) triggers the execution of the routine.

JTAG port. This port supports the IEEE standard Joint Test Action Group (JTAG) standard for system test. This standard defines a method for serially scanning the I/O status of each component in a system. Note that the ADSP-2192 does not support boundary scan.

Jumps. Program flow transfers permanently to another part of program memory.

Length registers. A length register is a Data Address Generator (DAG) register that sets up the range of addresses a circular buffer.

Level-sensitive interrupts. The DSP detects this type of interrupt if the signal input is low (active) when sampled on the rising edge of XTALI.

Loops. One sequence of instructions executes several times with zero overhead.

Memory blocks. The DSP's internal memory is divided into **blocks** that are each associated with different data address generators.

Modified addressing. The DAG generates an address that is incremented by a value or a register.

Modify address. The Data Address Generator (DAG) increments the stored address without performing a data move.

Modify registers. A modify register is a Data Address Generator (DAG) register that provides the increment or step size by which an index register is pre- or post-modified during a register move.

Multifunction computations. Using the many parallel data paths within its computational units, the DSP supports parallel execution of multiple computational instructions. These instructions complete in a single cycle, and they combine parallel operation of the computational units and memory accesses. The multiple operations perform the same as if they were in corresponding single-function computations.

Multiplier. This computational unit does fixed-point multiplication and executes fixed-point multiply/add and multiply/subtract operations.

Peripherals. This refers to everything outside the processor core. The ADSP-2192's peripherals include internal memory, I/O processor, JTAG port, and external devices that connect to the DSP.

Precision. The precision of a fixed-point number is the number of digits to the right of the decimal point. The ADSP-219x family arithmetic is optimized for numerical values in a fractional binary format denoted by 1.15 ("one dot fifteen"). The 1.15 format uses one sign bit (MSB) and fifteen fractional bits, representing values from -1 up to one LSB less than +1.

Post-modify addressing. The Data Address Generator (DAG) provides an address during a data move and auto-increments the stored address for the next move.

Pre-modify addressing. The Data Address Generator (DAG) provides a modified address during a data move without incrementing the stored address.

Saturation (ALU saturation mode). In this mode, all positive fixed-point overflows return the maximum positive fixed-point number, and all negative overflows return the maximum negative number.

Shifter. This computational unit completes logical shifts and arithmetic shifts on 16-bit operands.

Subroutines. The processor temporarily interrupts sequential flow to execute instructions from another part of program memory.

TCB chain loading. The process in which the DSP's DMA controller downloads a Transfer Control Block (TCB) from memory and autoinitializes the DMA parameter registers.

Transfer control block (TCB). A set of DMA parameter register values stored in memory that are downloaded by the DSP's DMA controller for chained DMA operations.

Tristate Versus Three-state. Analog Devices documentation uses the term “three-state” instead of “tristate” because Tristate™ is a trademarked term, which is owned by National Semiconductor.

Von Neumann architecture. This is the architecture used by most (non-DSP) microprocessors. This architecture uses a single address and data bus for memory access.

I INDEX

Numerics

Bits
  CE 9-6
CE 9-6
1/2X clock 11-10
16-bit receive data register 9-9
16-bit transmit data register 9-9
1X clock 11-10
Bits
  ACR 9-20
  FIP 9-4, 9-7
FIP 9-4
Pins
  SDI 9-19, 9-26
SDI 9-19, 9-26
Bits
  SLOT 9-4
  VGS 9-19
SLOT 9-4

A

Aborting a powerdown 11-24, 11-31
Aborting a powerup 11-24, 11-31
Abs function 2-17
Absolute address 3-15, G-2
AC '97 digital interface
  Defined 9-1
AC '97 frame
  Structure 9-27
AC '97 GPIO control / status register bit definitions 9-24
AC '97 GPIO status register (AC97SIF) 9-24
AC '97 input slot valid (AC97SVAL) register 9-22
AC '97 interface 9-1…9-37
AC '97 Interface Initiated (AC97) bit B-52
AC '97 Interface Initiated Interrupt Enabled (AC97 IEN) bit B-54
AC '97 link 9-13
  Stopping and restarting 9-13
AC '97 link control/status register (AC97LCTL) 9-15
AC '97 link status register (AC97STAT) 9-19
AC '97 link status register bit definitions 9-19
AC '97 pin listing 9-26
AC '97 protocol summary 9-27
AC '97 slot enable register (ACSE) 9-21
AC '97 slot enable register bit definitions 9-22

AC '97 slot request register (ACRQ) 9-24
AC codec 9-1
AC'97 codec control/status (CSR) register 9-29
AC'97 codec control/status register (CSR) write 9-30
AC'97 Codec Register Space, Secondary Codec 1 register (AC97EXT1) B-45
AC'97 Codec Register Space, Secondary Codec 2 register (AC97EXT2) B-46
AC'97 Codec Register Space-Primary Codec 0 register (AC97EXT0) B-45
AC'97 cold power down state 11-38
AC'97 Controller Registers B-41
AC'97 data slots 9-36
AC'97 digital controller 9-13
AC'97 frame structure 9-28
AC'97 GPIO Status Register (AC97SIF) B-44
AC'97 Input Slot Valid register (AC97SVAL) B-43
AC'97 Link Control/Status register (AC97LCTL) B-42
AC'97 link control/status register (AC97LCTL) B-42
AC'97 link powerdown states 9-30
AC'97 Link Status register (AC97STAT) B-42
AC'97 low power mode 11-38
AC'97 power management states 11-28
AC'97 Slot Enable register (AC97SEN) B-43
AC'97 Slot Request register (AC97SREQ) B-44
AC'97 standard B-3
AC'97 warm power down state 11-38
AC'97, powering down 11-33
AC97 bit B-52
AC97 IEN bit B-54
AC97LCTL
  AFR bit 9-33
  LKEN=1 bit 9-33
AC97LCTL register 9-15, B-42
AC97LCTL register bit definitions 9-15
AC97SEN register 9-21, B-43
  Writing to 9-29
AC97SEN register bit definitions 9-21
AC97SIF register 9-24, B-44
AC97SIF register bit definitions
  AC '97 GPIO control / status 9-24
AC97SREQ register B-44
AC97SREQ register bit definitions 9-24
AC97STAT
  LKOK=1 bit 9-33
AC97STAT register 9-12, 9-19, B-42
AC97SVAL register 9-22, B-43

Access to AC'97 codec control/status registers 9-28
Accumulator, dual 1-18
Accumulators, dual 2-2
AC-link digital serial interface protocol 9-10
  Characteristics 9-10
ACR bit 9-12, 9-22
ACR bits 9-20
ACRQ bit 9-24
ACRQ register 9-24
ACRST#
  De-asserting 9-30
ACRST# bit 9-33
ACRST# pin 9-26
ACSE bit 9-21
ACSE register 9-36
ACSV bit 9-22
ACTL register 9-27
ACVX bit B-16
ACWE bit 9-17, 9-33
ADC 9-10
Add instruction 2-20
Add with carry 2-17
Add/multiply G-5
Address decode (AD) stage 3-8
Address register 9-8, 9-9
Addressing
  (*See also DAGs and broadcast load*)
  (*See post-modify, pre-modify, modify, bit-reverse, or circular buffer*)
ADI chaining mode 9-30
Adjust bit B-23
ADSP-2192 AC '97 control registers 9-13
ADSP-2192 AC97 audio interface 9-25
ADSP-2192 Chip Control Registers B-13
ADSP-2192 DSP Peripheral Registers B-1
ADSP-2192 Dual-Core DSP Block Diagram B-3
ADSP-2192 Peripheral Device Control Register B-6, B-11
ADSP-2192 register bits 9-27
ADSP-2192 USB Data Pipe Operations 8-87
ADSP-21xx Family DSPs
  (*See Differences from previous DSPs and Porting from previous DSPs*)
AFD bit 9-16
AFR bit 9-15, 9-34
AFS bit 9-16
AGI bit 9-19
AGIx bit 9-19
AGPE bit 9-18
AGS bit 9-24
AIEN bit B-22
AINT bit B-22
Alternate registers (*See Secondary registers*)
ALU 2-1
  Arithmetic 2-6
  Arithmetic formats 2-8
  Data Flow Details 2-21
  Data registers A-2

Data Types 2-5
Division Support Features 2-23
Instruction Summary 2-19
Instructions 2-17, 2-19
Operations 2-17
Saturated results 2-11
Saturation mode 2-23
Status 2-16, 2-18
Status latching 2-10
ALU carry (AC) bit 2-6, 2-11, 2-18, 2-20, 2-21, 2-23, 2-45, 2-55, A-9, A-12
ALU carry (AC) condition 1-19, 3-40
ALU Feedback (AF) register 2-22
ALU feedback (AF) register
Saturated results 2-11
ALU input (AX/AY) registers 1-17
ALU negative (AN) bit 2-5, 2-18, 2-20, 2-21, A-9
ALU negative result (Neg) condition 1-20
ALU overflow (AV) bit 2-5, 2-6, 2-10, 2-11, 2-18, 2-20, 2-21, 2-38, 2-45, 2-51, 2-55, A-9, A-12
ALU overflow (AV) condition 1-19, 3-40
ALU overflow latch enable (AV_LATCH) bit 2-10, A-6, A-12
ALU overflow latch mode (OL) enable/disable 3-43
ALU positive result (Pos) condition 1-20
ALU quotient (AQ) bit 2-18, 2-20, 2-21, 2-24, A-10
ALU result (AR) register 2-11, 2-45, 2-53, A-2, A-4, A-6, A-12, A-16, A-17
ALU saturation mode enable (AR_SAT) bit 2-10, 2-11, A-6, A-12
ALU sign (AS) bit 2-18, 2-20, 2-21, A-9
ALU signed (AS) condition 1-19
ALU x- and y-input (AX AY) registers A-2, A-4, A-16
ALU zero (AZ) bit 2-18, 2-20, 2-21, A-9
AMC codec 9-1
Analog audio codec (AC) 9-1
Analog Devices' products 1-24
Analog/digital converter (ADC) 9-10
AND operator 2-17, 2-20
APME bit B-18
AR saturation mode (AS) enable/disable 3-43
Arithmetic
Formats Summary 2-8
Operations 2-17
Shifts 2-1
Arithmetic Logic Unit (*See ALU*)
Arithmetic operations 1-5
Arithmetic shift (Ashift) instruction 2-7, 2-37, 2-38, 2-41, 2-51

Arithmetic shifts G-5
Arithmetic status (ASTAT) register 1-19, 2-10, 2-11, 2-24, 2-31, 2-45, 2-55, 2-57, A-2, A-4, A-5, A-7, A-9, A-12
  Effect latency 3-6
  Latency 2-19
Arithmetic status bits A-9, A-10
ARPD bit 9-18
Assembler 1-16
Assembly language 2-1
Assigning DSP FIFOs 9-36
  AC'97 data slots 9-36
Asynchronous transfers G-1
Attributes of the powerdown states 9-32
Audience (intended) 1-1
Audio/modem (AMC) codec 9-1
Autobuffering 11-17
AWE bit B-21

B

B5V bit B-17
Background registers (*See Secondary registers*)
Barrel-shifter (*See Shifter*)
Base (Bx) registers 4-2, 4-15, A-3, A-25, G-1
Base Address Registers 8-8
Battery-powered operation 11-29
BCEN bit 9-16, 9-35
BCOE bit 9-16, 9-27, 9-33
BCOK bit 9-19
Begin loop address 3-21
Bias rounding enable (BIASRND) bit 2-10, A-20
Biased rounding 2-15
Binary coded decimal (BCD) format 2-4
Binary string 2-4
Bit
  AFR 9-15
Bit descriptions for FPM0, FPM1, and FPM2 registers B-18
Bit manipulation 2-1, G-5
BIT Organization of PCI I/O Space Registers 8-24
BIT_CLK bit 9-11
BITCLK
  Disabling 9-29
BITCLK bit 9-34
BITCLK pin 9-26
BITCLK signal 9-27
Bit-reverse addressing G-1
Bit-reverse addressing (BIT_REV) bit A-6, A-11
Bit-reversed addressing 4-1, 4-4, 4-6, 4-15
Bit-reversed addressing (BIT_REV) bit 4-4, 4-6, 4-15
Bit-reversed addressing mode (BR) enable/disable 3-43
Bits
  AC97 B-52
  AC97 IEN B-54
  ACR 9-12, 9-22
  ACRQ 9-24
  ACRST# 9-33

ACSE 9-21
ACSV 9-22
ACVX B-16
ACWE 9-17, 9-33
Adjust B-23
ADSP-2192 register 9-27
AFD 9-16
AFR 9-33, 9-34
AFS 9-16
AGI 9-19
AGIx 9-19
AGPE 9-18
AGS 9-24
AIEN B-22
AINT B-22
APME B-18
ARPD 9-18
AWE B-21
B5V B-17
BCEN 9-16, 9-35
BCOE 9-16, 9-27, 9-33
BCOK 9-19
BIT_CLK 9-11
BITCLK 9-34
BUS B-17
CE 9-4
Conf Rdy B-53
CRST B-16
D2PM1 IEN B-53
DMA EN B-49
DME 9-4, 9-7
DPLLK 11-7
DPLLM 11-7
DPLLN 11-7
DSP1 IN0 ENA 6-24
DSP1 IN0 PEND 6-21
DSP1 IN1 ENA 6-25
DSP1 IN1 PEND 6-22
DSP1 OUT0 ENA 6-25
DSP1 OUT0 PEND 6-22
DSP1 OUT1 ENA 6-25
DSP1 OUT1 PEND 6-22
DSP2 IN0 ENA 6-25
DSP2 IN0 PEND 6-22
DSP2 IN1 ENA 6-25
DSP2 IN1 PEND 6-22
DSP2 OUT0 PEND 6-22
DSP2 OUT1 PEND 6-22
DSP2/$\overline{\text{DSP1}}$ B-49
Empty B-50
FIEN B-21
Flush FIfo B-49
FUNCTION B-49
GIEN B-22
GINT B-22
GPIO B-52
GPIO IEN B-54
GPIO_INT 9-19
GPME B-18
GWAKE B-38
GWE B-21
GWKE B-36
GWKF B-40
GWKM B-37
HALT B-50
IN0 valid 6-22
IN1 valid 6-23
INTE B-36

INTF B-40
INTM B-38
INTR B-39
LKEN 9-17, 9-33, 9-34
LKOK 9-19, 9-33, 9-34
LOOP B-50
MAbort IEN B-54
Master Abort B-52
MBox 0 IN B-52
MBox 0 OUT B-52
MBox 1 IN B-52
MBox 1 OUT B-52
MLNK 9-27, 9-34, 9-35
OUT0 valid 6-23
OUT1 valid 6-23
P2DM0 IEN B-53
P2DM1 IEN B-53
PACK DIS B-49
PCI IN0 ENA 6-24
PCI IN0 PEND 6-21
PCI IN1 ENA 6-24
PCI IN1 PEND 6-21
PCI OUT0 ENA 6-24
PCI OUT0 PEND 6-21
PCI OUT1 ENA 6-24
PCI OUT1 PEND 6-21
PCI RDY B-56, B-58, B-59
PCI5V B-17
PCIF B-53
PCIRST B-17
PD 11-24, 11-25, 11-31, 11-34, B-20
PME B-19
PME_EN B-18
PMIEN B-21
PMINT B-22
PMWE B-21
PR4 9-27, 9-34, 9-35
PRn 9-34
PU 11-24, 11-25, 11-31, 11-34, 11-35, B-20
PWRST B-18
RDIS B-15
REG 9-20
REGD B-16
RFE 9-7
RO 9-8
RST B-14
RSTD B-21
RWE B-21
RX0 DMA B-51
RX1 DMA B-51
SCK B-29
SCKI B-29
SDA B-29
SDAI B-29
SEN B-29
SLOT 9-7
SMSEL 9-4, 9-6
SPME B-19
SYEN 9-15, 9-33, 9-34
SYNC 9-20
TAbort IEN B-54
Target Abort B-52
TFE 9-5
TFF 9-5
TU 9-5
TX0 DMA B-51

TX1 DMA B-51
Vaux B-17
WKUP B-37
WR/$\overline{RD}$ B-49
XON B-15
bits 9-4, 9-6
BitsDSP2 OUT0 ENA 6-25
BitsDSP2 OUT1 ENA 6-25
Block exponent 2-57
Blocks of memory 5-1, 5-2, 5-4, G-4
Books to read 11-41
Branching execution 3-14
Delayed branch 3-14
Direct and indirect branches 3-15
Immediate branches 3-17
Indirect branches 3-15
Buffer overflow 4-14
Buffer overflow, circular 4-11
BUS bit B-17
Bus exchange (*See Program memory bus Exchange (PX) register*)
Bus exchange, program memory (PX) register A-3
Buses
Arbitration 5-3
BUSMODE Configuration 8-1

C

Cable for information 1-25
Cache control (CACTL) register 3-11, A-6, A-23
Effect latency 3-6
Cache DM access enable (CDE) bit 3-11, A-6
Cache efficiency 3-12
Cache Freeze (CFZ) bit A-6
Cache freeze (CFZ) bit 3-11
Cache hit/miss (*See Cache efficiency*)
Cache PM access enable (CPE) bit 3-11, A-6
Cache usage, optimizing 3-12
Call instruction 1-22
Call instructions 3-14, 3-43
Conditional branch 3-14
Delayed branch 3-14
Restrictions 3-19
CardBus function event (CB_FE0) register B-36
CardBus Function Event Force (CB_FEF0) Register B-40
CardBus Function Event Mask (FEM) Register B-37
CardBus Function Event Present State (CB_FPS0) Register B-38
CardBus Function Event Registers B-32
CardBus mode
external clock 11-36
Carry (*See ALU carry (AC) bit*)
Carry output 2-18
CB_FEF0 register B-40
CB_FPS0 register B-38
CE bit 9-4
CE bits 9-6
Chain Pointer (CPx) registers 7-9, 7-22

Chained DMA sequences 7-22
Chaining mode
ADI 9-30
Chip Control (SYSCON) Registers B-14
Chipset
Digital controller 9-1
Circuit, digital, design of 11-41
Circular buffer addressing 4-11, A-25, G-1
Registers 4-14
Restrictions 4-12
Setup 4-12
Wrap around 4-14
CIS Tuple Requirements B-35
Clear bit (CLRBIT) instruction 2-20
Clear interrupt (Clrint) instruction 3-43
Clearing results 2-30
CLKSEL pin 11-7
Clock
1/2X 11-10
1X 11-10
Clock distribution 11-41
Clock domains 11-8, 11-9
Clock generator 11-25, 11-32
Clock generator, DSP 11-25, 11-32
Clock oscillator 11-35
Clock rate, internal, changing default multiplier 11-7
Clock signals 11-7
Clock, external 11-7, 11-36
CMOS level 11-37
CMOS standby state 11-32
Codec ID
Reading unpopulated 9-29
Codecs
External 9-25
Codec-to-ADSP-2192 communication 9-11
Cold or warm states 9-30
Cold power down state, AC'97 11-38
Commands executed after boot 11-15
Commonalities Between the Three Functions B-54
Companding (compressing/expanding) G-1
Compiler 1-16
Computational
Instructions 2-1
Mode, setting 2-10
Status, using 2-16
Computational resources 9-26
Computational units 2-1
Condition Code (CCODE) condition 3-41
Condition code (CCODE) register 1-19, A-3, A-4, A-5, A-7, A-23
Effect latency 3-6
Conditional
Branches 3-14, 3-16, G-1
Instructions 2-16, 3-5, 3-39
Test in loops 3-21
Conditional instructions 1-22
Register usage 2-62

Conf Rdy bit B-53
CONFIG DEVICE DEFINITION 8-85
Configuration Ready (Conf Rdy) bit B-53
Configuration Space Interactions between Functions 8-5
Configuration Spaces 8-2
Configuring AC'97 sample data streams 9-36
Connections, external crystal 11-8
Connectors 11-41
Contact information 1-25
Context switching 2-23, 2-36, 2-54, 2-59
Control 8-21
Control registers
  STCTLx 9-2
Control status (CSR) register 9-29
Conventions 1-27
Core registers A-2
Core-to-core flags E-1
Count (Cx) registers 7-9
Count register 9-8
Counter (CNTR) register 3-20, 3-38, A-4, A-5, A-22
  Effect latency 3-6
Counter expired (CE) condition 1-22, 3-20, 3-41
CRST bit B-16
Crystal connections, external 11-8
Crystal oscillator 11-7
Crystal type 11-7
CSR writes 9-30, 9-37
Current Interrupt State (INTR) bit B-39
Current wakeup state (GWAKE) bit B-38
Customer support 1-25

D

D0 power management state 11-28
D1 power management state 11-28
D2 power management state 11-28
D2PM1 IEN bit B-53
D3cold power management state 11-28
D3hot power management state 11-28
DAG
  Addressing Modes 1-21
  Features 1-5
  Instructions 4-21
DAG secondary registers mode (BSR) enable/disable 3-43
DAGs 4-3
  Data move restrictions 4-20
  Data moves 4-20
  Instructions 4-21
  Operations 4-9
  Setting Modes 4-4
  Status 4-8
  Support for branches 3-3, 3-15
Data (Dreg) registers G-2
Data access
  (*See also Data moves*)
  Conflicts 5-3
  Dual-data access restrictions 5-2

Dual-data accesses 5-2
Data Address Generators (DAGs) G-2
Instructions 5-14
Data Address Generators (*See DAGs*)
Data alignment 5-5
Data Buffers 7-10
Data flow 2-1
Data formats 2-2
Data Memory (DM) bus 2-53
Data memory page (DMPGx) registers 1-11, 1-20, 4-2, 4-6, 4-7, A-3, A-4, A-25
Effect latency 3-6
Data move
Instructions 5-14
Data register file 2-1, 2-57
Data registers 2-57, A-16, G-2
Data slots 9-36
De-asserting ACRST# 9-30
Delay, synchronization 11-9
Delayed branch (DB) Jump or Call 3-14, 3-16, 3-18, G-2
Restrictions 3-19
Delayed branch (DB) operator 3-16
Delayed branch slots 3-18
Denormalize operation 2-42, 2-53
Derive block exponent 2-37, 2-39, 2-57
Destination registers 9-9
Development tools 1-15
Differences from previous DSPs 1-17…1-24
Shifting data into SR2 2-41
Digital circuit design 11-41
Digital controller (DC) chipset 9-1
Diodes, protection 11-25, 11-26
Direct branch 3-15, G-2
Disable mode (Dis) instruction 3-43
Disabling an AC’97 sample stream 9-36
Disabling the link 9-29
Disabling the locally generated BITCLK 9-29
Divide primitive (DIVS/DIVQ) instructions 2-5, 2-20, 2-23, 2-24, 2-26
Division
Signed 2-24
Unsigned 2-24
DMA
External port 7-22
Serial port 7-24
DMA (Direct Memory Accessing) G-2
DMA chaining G-2
DMA channel
Channels, parameters, and buffers 7-10
Current count 9-8
Priority 7-21
DMA Channel Halt Status (HALT) bit B-50
DMA Channel Loop Status (LOOP) bit B-50
DMA channels 7-10

DMA Control (DMACx) registers 7-12
DMA Control 0 - Bus master control and status B-49
DMA Control Registers B-46
DMA controller 1-4
  Operation 7-20
DMA EN bit B-49
DMA Enable (DMA EN) bit B-49
DMA enable (DME) bit 9-7
DMA Enable, external port (DEN) bit 7-11, 7-13
DMA FIFO Empty Status (Empty) bit B-50
DMA Packing Disable (PACK DIS) bit B-49
DMA parameter registers G-2
DMA sequences
  Chaining sequences 7-22
  Sequence end 7-20
  Sequence start 7-20
  TCB loading G-2
DMA Transfer Count 0 - Bus master Sample transfer count B-48
DMA Transfer Count 1 - Bus master Sample transfer count B-48
DMA transfers
  Number specified 9-9
DMA Write / Read (WR/$\overline{RD}$) bit B-49
DME bit 9-4, 9-7
Do Until instruction A-6
Do/Until instruction 1-22, 3-22, 3-23, 3-43
  (*See also Loop*)
  Restrictions 3-19
Dormant state 11-29
DPLLK bit 11-7
DPLLM bit 11-7
DPLLN bit 11-7
DSP
  Background information 1-17
  Core architecture 1-7
  Defined 1-1
  Peripherals architecture 1-9
  Serial ports (SPORTs) 1-13
DSP clock generator 11-25, 11-32
DSP Code Download 8-65
DSP core
  Register map 9-9
DSP DMA FIFOs 9-26
DSP Mailbox Registers 8-18, B-30
DSP peripherals architecture B-3
DSP PLL Control Register B-23
DSP power management states 11-28
DSP Powerdown Registers B-19
DSP Register Definitions 8-58
DSP to PCI Mailbox 1 Transfer Interrupt Enabled (D2PM1 IEN) bit B-53
DSP1 IN0 ENA bit 6-24
DSP1 IN0 PEND bit 6-21
DSP1 IN1 ENA bit 6-25
DSP1 IN1 PEND bit 6-22
DSP1 OUT0 ENA bit 6-25

DSP1 OUT0 PEND bit 6-22
DSP1 OUT1 ENA bit 6-25
DSP1 OUT1 PEND bit 6-22
DSP2 IN0 ENA bit 6-25
DSP2 IN0 PEND bit 6-22
DSP2 IN1 ENA bit 6-25
DSP2 IN1 PEND bit 6-22
DSP2 OUT0 ENA bit 6-25
DSP2 OUT0 PEND bit 6-22
DSP2 OUT1 ENA bit 6-25
DSP2 OUT1 PEND bit 6-22
DSP2/$\overline{\text{DSP1}}$ bit B-49
D-state 11-30
Dual accumulator 1-18
Dual accumulators 2-2
Dual-voltage processor 11-24, 11-25

E

Edge-sensitive interrupts 11-10, G-3
EEPROMI/O control/status (SPROMCTL) register B-28
Effect latency (*See Latency*)
E-mail for information 1-25
Empty bit B-50
Emulation 11-39
Emulation, JTAG port 1-4
Emulator cycle counter interrupt enable (EMUCNTE) bit A-21
Emulator interrupt mask (EMU) bit A-19
Enable mode (Ena) instruction 2-60, 3-43
Enable/Disable mode (Ena/Dis) instruction 4-6
Enabling an AC'97 sample stream 9-36
Endian Format, Little Versus Big G-3
End-of-loop 3-24
Equal zero (EQ) condition 3-40
Equals zero (EQ) condition 1-19
ESD protection 11-25, 11-26
Example Initialization Process 8-81
Excess-n formats 2-4
Execute (PC) stage 3-8
Explicit stack operations 3-39
Exponent adjust (EXPADJ) instruction 2-38, 2-39, 2-51
Exponent compare logic 2-52
Exponent derivation 2-1
  Double-precision number 2-38
Exponent derive (EXP) instruction 2-38, 2-44, 2-51
Exponent detector 2-53, 2-55, 2-57
Exponent, shifter (SE) register A-2
Exponent, shifter block (SB) register A-2
External audio codec (AC '97) subsystem 9-25
External clock 11-7
External codec 9-27
External codecs 9-25
External port DMA
  DMA setup 7-22
EZ-KIT Lite 11-40

F

FAX for information 1-24
Feedback, input 2-1
Fetch address 3-2, 3-3
Fetch address (FA) stage 3-8
FFT calculations 4-15
Field deposition/extraction G-5
FIEN bit B-21
FIFO
  Size for AC'97 interface 9-2
FIFO control and status register 9-3
FIFO DMA address registers 9-8
FIFO DMA count registers 9-9
FIFO DMA current count registers 9-8
FIFO DMA next address registers 9-9
FIFO interrupt position (FIP) bit 9-7
FIFO receive control and status register 9-5
FIFO receive control and status registers 9-2
FIFO registers 9-1
FIFO transmit control and status register 9-3
FIFOs E-5
File Transfer Protocol (FTP) site 1-24
FIP bit 9-7
FIP bits 9-4
Fixed-point DSP (why?) 1-2
Flag In (FI) 11-1
Flag Input 1-22
Flag input pins during low power state 11-37
Flag Out (FO) 11-1
Flag pins 11-22
Flag update 2-18, 2-31, 2-50, G-3
Flags 11-37
Flags, core-to-core E-1
Flush Cache instruction 3-11, 3-43
Flush DMA buffers/status (FLSH) bit 7-12
Flush FIfo bit B-49
Flush Master FIFO (Flush FIfo) bit B-49
Forced rebooting, by software 11-16
Forever condition 3-41
Fractional mode 2-5, 2-6, 2-10, A-13
  Results format 2-12
Fractional mode (*See also Multiplier results mode*)
Fractional Representation (1.15) 2-5
Fractional/integer mode (MM) enable/disable 3-43
FUNCTION bit B-49
Function Select (FUNCTION) bit B-49

G

Gates, logic 11-41
General Purpose I/O Pin Initiated (GPIO) bit B-52

General Purpose I/O Pin Initiated Interrupt Enabled (GPIO IEN) bit B-54
General Purpose IO (GPIO) Control Registers B-24
General USB Device Definitions and Descriptor Tables 8-52
General Wakeup Event Pending (GWKE) bit B-36
General Wakeup Mask (GWKM) bit B-37
GIEN bit B-22
GINT bit B-22
Global interrupt enable (GIE) bit 3-37, A-20
Global interrupt mode (INT) enable/disable 3-43
GND
  Used with SDI pins 9-26
GPIO bit B-52
GPIO configuration (CPIOCFG) register B-25
GPIO control (GPIOCTL) register B-27
GPIO IEN bit B-54
GPIO polarity (GPIOPOL) register B-25
GPIO pulldown (GPIOPDN) register B-27
GPIO pullup (GPIOPUP) register B-27
GPIO status (GPIOSTAT) register B-26
GPIO sticky (GPIOSTKY) register B-26
GPIO wakeup control (GPIOWCTL) register B-26
GPIO_INT bit 9-19
GPIOCFG register B-24
GPIOCTL register B-24
GPIOPDN register B-25
GPIOPOL register B-24
GPIOPUP register B-25
GPIOSTAT register B-24
GPIOSTKY register B-24
GPIOWCTL register B-24
GPME bit B-18
Greater than or equal to zero (GE) condition 1-19
Greater than or equal zero (GE) condition 3-40
Greater than zero (GT) condition 1-19, 3-40
Ground planes 11-41
GSM speech compression routines 2-15
GWAKE bit B-38
GWE bit B-21
GWKE bit B-36
GWKF bit B-40
GWKM bit B-37

H

HALT bit B-50
Harvard architecture 5-1, G-3
High shift (HI) option 2-37, 2-38, 2-53, 2-54

High shift, except overflow (HIX) option 2-37, 2-38, 2-45, 2-55
High watermark, stack 3-36, 3-37
Host Mailbox Registers B-30
Host port 1-12
Host Port Selection 8-1
Hypertext links 1-28

I

I/O memory page (IOPG) register A-3, A-4, A-6, A-26
  Effect latency 3-6
I/O memory space 5-9, 5-13
I/O pins B-3
I/O processor 1-4, 7-1, 7-6, 7-18
  DMA channel priority 7-21
  External port modes 7-12
  Registers G-3
  Serial port modes 7-18
I/O Space Indirect Access Registers 8-23
IDLE instruction 11-34, 11-39
Idle instruction 3-34, 3-43, G-3
  Defined 3-1
  Restrictions 3-19
IEEE 1149.1 JTAG specification G-4
IF conditional operator 3-43
If conditional operator 2-20
IMASK register E-1
Immediate addressing
  Memory page selection 4-7
Immediate branch 3-17
Immediate shifts 2-40
Implicit stack operations 3-38
IN Transactions (Host 8-92
IN0 valid bit 6-22
IN1 valid bit 6-23
InBox 8-18
InBox 0 - PCI/USB to DSP Mailbox 0 (MBX_IN0) register 6-26
InBox 1 - PCI/USB to DSP Mailbox 1 (MBX_IN1) register 6-26
InBox0 Data Valid (IN0 valid) bit 6-22
InBox0 DSP #1Interrupt Enable (DSP1 IN0 ENA) bit 6-24
InBox0 DSP 1 Interrupt Pending (DSP1 IN0 PEND) bit 6-21
InBox0 DSP 2 Interrupt Pending (DSP2 IN0 PEND) bit 6-22
InBox0 PCI Interrupt Enable (PCI IN0 ENA) bit 6-24
InBox0 PCI Interrupt Enable (PCI OUT0 ENA) bit 6-24
InBox0 PCI Interrupt Pending (PCI IN0 PEND) bit 6-21
InBox1 Data Valid (IN1 valid) bit 6-23
InBox1 DSP #1 Interrupt Enable (DSP1 IN1 ENA) bit 6-25
InBox1 DSP #2 Interrupt Enable (DSP2 IN0) bit 6-25
InBox1 DSP #2 Interrupt Enable (DSP2 IN1 ENA) bit 6-25

InBox1 DSP 1 Interrupt Pending (DSP1 IN1 PEND) 6-22
InBox1 DSP 2 Interrupt Pending (DSP2 IN1 PEND) bit 6-22
InBox1 PCI Interrupt Enable (PCI IN1 ENA) bit 6-24
InBox1 PCI Interrupt Enable (PCI OUT1 ENA) bit 6-24
InBox1 PCI Interrupt Pending (PCI IN1 PEND) bit 6-21
Incoming Mailbox 0 PCI Interrupt (MBox 0 IN) bit B-52
Incoming Mailbox 1 PCI Interrupt (MBox 1 IN) bit B-52
Index (Ix) registers 4-2, 4-7, 4-14, 4-15, A-3, A-4, A-24, G-3
Indirect Access to I/O Space 8-23
Indirect branch 3-15, G-3
Indirect jump memory page (IJPG) register A-3, A-4, A-6, A-21
Indirect Jump Page (IJPG) Register 3-15
Indirect jump page (IJPG) register 1-11, 3-15
  Effect latency 3-6
Infinite loops 3-20
Infinite loops (Forever) condition 1-22
Instruction cache 3-9, 3-10, 5-2
Instruction decode (ID) stage 3-8
Instruction pipeline 3-3, 3-7, 3-16
Instruction set 1-1, 1-27
  ALU instructions 2-20
  DAG instructions 4-22
  Enhancements 1-24
  Multifunction instructions 2-63, 4-22
  Multiplier instructions 2-33
  Program Sequencer instructions 3-43
  Shifter instructions 2-51
Instructions
  Registers for conditional/multifunction 2-62
INTE bit B-36
Integer mode 2-7, 2-10, A-13
  Results format 2-13
Integer mode (*See also Multiplier results mode*)
Integer/fractional mode (MM) enable/disable 3-43
Interactions Between Functions 8-5
Interactions Between the Three Functions B-55
Interleaved data 5-13
Internal clock rate, changing default multiplier 11-7
Internal I/O Bus Arbitration (Request and Grant) 7-21
Internal memory 5-1, 5-9, 5-10, 5-11
Interrupt control (ICNTL) register A-3, A-4, A-5, A-7, A-20
  Effect latency 3-6
Interrupt controller 3-3
Interrupt Event Pending (INTE) bit B-36
Interrupt Force (INTF) bit B-40

Interrupt latch (IRPTL) register
3-33, A-3, A-4, A-19
Effect latency 3-6
Interrupt latency 3-26
Cache miss 3-28
Delayed branch 3-29
Single-cycle instruction 3-27
Writes to IRPTL 3-26
Interrupt mask (IMASK) register
3-31, A-3, A-4, A-5, A-19
Effect latency 3-6
Interrupt mask pointer (IMASKP)
register 3-33
Interrupt masking E-5
Interrupt mode (INT)
enable/disable 3-43
Interrupt nesting enable (INE) bit
3-32, A-20
Interrupt request 11-1
Interrupt requests, edge-sensitive
11-10
Interrupt vector, defined 3-24
Interrupt, non-maskable 11-39
Interrupt, peripheral E-1
Interrupt/ Wakeup Mask (INTM)
bit B-38
Interrupt/Powerdown Registers
11-24, 11-31
Interrupting Idle 3-34
Interrupts 1-12, 3-24, 8-14, 11-22,
11-37, E-4, E-5, G-3
Delayed branch 3-20
Hold off 3-30
Idle instructions 3-34
Inputs 3-25
Interrupt nesting 3-32
Interrupt sensitivity G-4
IRPTL write timing 3-26
Latency (*See Interrupt latency*)
Masking and latching 3-31
Nesting and processing 3-30
Processing delays 3-30
Response 3-25
Software 3-26
Interrupts, defined 3-1
Interrupts, global enable (GIE) bit
3-31, 3-37, A-20
INTF bit B-40
INTM bit B-38
INTR bit B-39
IO pin ESD protection 11-26
IOPG register
Selecting a codec 9-23
IRPTL register E-1

J

JTAG ID registers B-32
JTAG port 1-4, 1-15, 10-1, G-4
Jump instruction 1-22, 3-14, 3-43
Conditional 3-14
Delayed branch 3-14
Restrictions 3-19
Jump instructions G-4
Jumps, defined 3-1

L

Latching Interrupts 3-31
Latchup events 11-25

Latency 2-19, 3-11, 3-26, 7-12, 7-18
  Enabling modes 2-60
  Jump instructions 1-23
  Program sequencer registers 3-5
  Registers A-5
Layer stacking 11-41
Length (Lx) registers 4-2, 4-15, A-3, A-4, A-25, G-4
  Initialization requirements 4-2
Less than or equal zero (LE) condition 3-40
Less than zero (LE) condition 1-19
Less than zero (LT) condition 1-19, 3-40
Level, stack interrupt 3-37
Level-sensitive interrupts G-4
Link
  Disabling 9-29
  Restarting 9-35
Link buffer DMA Enable (LxDEN) bit 7-11
Link powerdown states
  Illustrated 9-31
Link powerdown states for AC'97 9-30
Link powerdown states, by function 9-31
Link powerdown states, by signal 9-32
Linker 1-17
LKEN bit 9-17, 9-33, 9-34
LKOK bit 9-19, 9-33, 9-34
Loader 1-17
Logic gates 11-41
Logical (AND, OR, XOR, NOT) operators 2-20
Logical shift (Lshift) instruction 2-7, 2-37, 2-38, 2-41, 2-51
Logical shifts G-5
Long call (Lcall) instruction 3-7, 3-15, 3-43
Long jump (Ljump) instruction 3-7, 3-15, 3-43
Long word
  Data access 5-5
Look ahead address (LA) stage 3-8
Loop G-4
  Address stack 3-5
  Begin address 3-21
  Conditional loops 3-20
  Conditional test 3-21
  Defined 3-1
  Do/Until example 3-20
  End restrictions 3-24
  Infinite 3-20, 3-41
  Nesting restrictions 3-24
  Stack management 3-24
  Termination 3-5, 3-23
LOOP bit B-50
Loop counter expired (CE) condition 3-20
Loop stack address (LPSTACKA) register A-3, A-4, A-22
Loop stack empty (LPSTKEMPTY) condition 3-43

Loop stack empty status (LPSTKEMPTY) bit 3-36, A-14
Loop stack full (LPSTKFULL) condition 3-43
Loop stack full status (LPSTKFULL) bit 3-36, A-14
Loop stack page (LPSTACKP) register A-3, A-4, A-22
Low latency
  Recommendation 9-23
Low power 11-29
Low power configuration 11-36
Low power consumption 11-32
Low power mode
  AC'97 11-38
Low power operation 1-13
Low shift (LO) option 2-37, 2-38, 2-53, 2-54
Low watermark, stack 3-36, 3-37
Lowest power consumption 11-30, 11-36, 11-37
Lowest power dissipation 11-37
LQFP 11-3

M

MAbort IEN bit B-54
MAC overflow (MV) condition 3-40
Mailbox Control Register 8-21
Mailbox interrupt control (MBXCTL) register 6-24
Mailbox status (MBXSTAT) register 6-21
Mailbox Status Register 8-19
Mailing address for information 1-25
Masking interrupts 3-31, E-5
Master Abort bit B-52
MBox 0 IN bit B-52
MBox 0 OUT bit B-52
MBox 1 IN bit B-52
MBox 1 OUT bit B-52
MBXCTL register 6-24
MBXCTL register bit descriptions 6-24
MC codec 9-1
MCU Register Definitions 8-33
Measurement techniques 11-41
Memory 1-4, 5-1, 5-4, 5-9, 5-10, 5-11
  Access priority 5-2, 5-3
  Architecture 1-9
  Blocks of memory 5-1, 5-2, 5-4, G-4
  External memory (off-chip) 1-12
  Internal memory (on-chip) 1-11
  Memory map 5-8
  MIPS and DSP 9-26
Memory page selection 4-7
Memory, unified 1-20
MIPS and DSP memory 9-26
MLNK bit 9-27, 9-34, 9-35
Mnemonics (*See Instructions*)
Mode
  Biased rounding 2-10

Mode status (MSTAT) register 2-13, 4-5, 4-15, A-2, A-4, A-6, A-7, A-11
  Effect latency 3-6
Mode Strap Pin Connections 8-2
Modem codec (MC) 9-1
MODEM/AUDIO DEVICE DEFINITION 8-85
Modes
  ALU overflow latch 2-10
  ALU saturation 2-10
  Chaining 9-30
Modified addressing 4-9, G-4
Modify (Mx) registers 4-2, 4-14, A-3, A-4, A-24, G-4
Modify address 4-1, G-4
Modify instruction 4-19, 4-22
Multifunction computations G-4
Multifunction instruction
  Register usage 2-62
  Registers 11-12
Multifunction instructions 2-60, 11-11
  DAG restrictions 4-11
  Definition 11-11
  Delimiting and terminating 2-61, 2-63
Multiplier 2-1, 2-36, G-5
  Arithmetic formats 2-8
  Clear operation 2-30
  Data types 2-6
  Input operators 2-29, 2-33
  Instructions 2-33, 2-63
  Operations 2-28, 2-31
  Result (MR) register 2-28
  Result mode 2-10
  Results 2-30
  Rounding 2-30
  Saturation 2-30
  Status 2-16, 2-31
Multiplier data registers A-2
Multiplier dual accumulator 1-18
Multiplier feedback (MF) register 1-18, 2-36
Multiplier input (MX/MY) registers 1-17
Multiplier integer mode (MM) enable/disable 3-43
Multiplier mode (M_MODE) bit 2-10
Multiplier of internal clock rate 11-7
Multiplier overflow (MV) bit 2-31, 2-32, 2-34, A-10
Multiplier overflow (MV) condition 1-19
Multiplier result (MR) register 2-13, 2-14, 2-29, 2-32, 2-34, 2-53
Multiplier result (MR0-2) registers A-2, A-4, A-16, A-17
Multiplier results (MR) register 1-18, 2-2
Multiplier results mode selection (M_MODE) bit A-6, A-13
Multiplier results overflow (MV) bit 2-33

Multiplier status bits 2-10, 2-31, A-10, A-20
Multiplier x- and y-input (MX MY) registers A-4, A-16, A-17
Multiply instruction 2-33
Multiply—accumulator (*See Multiplier*)
Multiprecision operations 2-23

N

Negative, ALU (AN) bit A-9
Nested interrupts 3-32
NextAddress register 9-8
No operation (Nop) instruction 3-43
Non-maskable interrupt 11-39
Normalize
  ALU result overflow 2-45
  Double precision input 2-47
  Operations 2-37, 2-49
  Single precision input 2-44
Normalize (NORM) instruction 2-38, 2-51
  Execution difference 2-46
Normalize operation 2-53
Not equal to zero (NE) condition 1-19
Not equal zero (NE) condition 3-40
NOT operator 2-20
NOVRAM Changeable Fields for USB Descriptors 8-86
NOVRAM Interface 8-86
NxtAddress register 9-9

O

One's complement 2-4
Operands 2-17, 2-28, 2-57, G-2
OR function 9-26
OR operator 2-20
Oscillator stabilization 11-35
Oscillator, clock 11-35
Oscillator, crystal 11-7
OUT Transactions (Host -> Device) 8-91
OUT0 valid bit 6-23
OUT1 valid bit 6-23
OutBox 8-19
OutBox 0 - DSP to PCI/USB Mailbox 0 (MBX_OUT0) register 6-26
OutBox 1 - DSP to PCI/USB Mailbox 1 6-26
OutBox0 Data Valid (OUT0 valid) bit 6-23
OutBox0 DSP #1 Interrupt Enable (DSP1 OUT0 ENA) bit 6-25
OutBox0 DSP #2 Interrupt Enable (DSP2 OUT0 ENA) bit 6-25
OutBox0 DSP 1 Interrupt Pending (DSP1 OUT0 PEND) bit 6-22
OutBox0 DSP 2 Interrupt Pending (DSP2 OUT0 PEND) bit 6-22
OutBox0 PCI Interrupt Pending (PCI OUT0 PEND) bit 6-21
OutBox1 Data Valid (OUT1 valid) bit 6-23
OutBox1 DSP #1 Interrupt Enable (DSP1 OUT1 ENA) bit 6-25

OutBox1 DSP #2 Interrupt Enable (DSP2 OUT1 ENA) bit 6-25
OutBox1 DSP 1 Interrupt Pending (DSP1 OUT1 PEND) 6-22
OutBox1 DSP 2 Interrupt Pending (DSP2 OUT1 PEND) bit 6-22
OutBox1 PCI Interrupt Pending (PCI OUT1 PEND) bit 6-21
Outgoing Mailbox 0 PCI Interrupt (MBox 0 Out) bit B-52
Outgoing Mailbox 1 PCI Interrupt (MBox 1 Out) bit B-52
Overflow A-9, A-10
  ALU (*see ALU overflow (AV) bit*)
  Multiplier (*see Multiplier overflow (MV) bit*)
  Shifter (*see Shifter overflow (MV) bit*)
Overflow latch mode (OL) enable/disable 3-43
Overflow, ALU 2-11
Overflow, ALU latch mode 2-10
Overflow, stack 3-36, 3-37
Overrun, timer 11-17
Overview 8-1, 8-25, B-1
Overview—Why Fixed-Point DSP? 1-2

P

P2DM0 IEN bit B-53
P2DM1 IEN bit B-53
PACK DIS bit B-49
Package configuration 11-3
Page
  DAG page (DMPGx) registers A-3
  I/O memory page (IOPG) register A-3
  Indirect jump page (IJPG) register A-3
Parallel assembly code (*See Multifunction computation*)
Parallel operation to improve performance 11-11
Parallel operations 2-60, G-4
Pass instruction 2-20
PC stack address (STACKA) register 3-5, A-3
PC stack empty (PCSTKEMPTY) condition 3-43
PC stack empty status (PCSTKEMPTY) bit 3-36, A-14
PC stack full (PCSTKFULL) condition 3-43
PC stack full status (PCSTKFULL) bit 3-36, A-14
PC stack interrupt enable (PCSTKE) bit 3-37, A-20
PC stack level (PCSTKLVL) condition 3-43
PC stack level status (PCSTKLVL) bit 3-36, 3-37, A-14
PC stack page (STACKP) register 3-5, A-3
PCI Clock Domain 8-11
PCI Configuration Register Space B-54

PCI Configuration Register Space, Function 0 B-56
PCI Configuration Register Space, Function 1 B-58
PCI Configuration Register Space, Function 2 B-59
PCI Configuration Space 8-3
PCI Control Register 8-16, B-53
PCI DMA address, count registers B-46
PCI DMA Control Registers B-46
PCI functions B-18
PCI Functions Configured (PCIF) bit B-53
PCI IN0 ENA bit 6-24
PCI IN0 PEND bit 6-21
PCI IN1 ENA bit 6-24
PCI IN1 PEND bit 6-21
PCI Interface Master Abort Detect Interrupt Enabled (MAbort IEN) bit B-54
PCI Interface Master Abort Detected (Master Abort) bit B-52
PCI Interface Target Abort Detect Interrupt Enabled (TAbort IEN) bit B-54
PCI Interface Target Abort Detected (Target Abort) bit B-52
PCI Interrupt Register 8-15, B-50
PCI Interrupt, Control Registers B-47
PCI mode
external clock 11-36
PCI OUT0 ENA bit 6-24
PCI OUT0 PEND bit 6-21
PCI OUT1 ENA bit 6-24
PCI OUT1 PEND bit 6-21
PCI Parallel Interface 8-2
PCI Port Priority on PDC Bus 8-18
PCI power management control/status (SYSCON) register B-33
PCI power management states 11-28
PCI RDY bit B-56, B-58, B-59
PCI reset 11-14
PCI to DSP Mailbox 0 Transfer Interrupt Enabled P2DM0 IEN) bit B-53
PCI to DSP Mailbox 1 Transfer Interrupt Enabled (P2DM1 IEN) bit B-53
PCI, powering down 11-32
PCI5V bit B-17
PCIF bit B-53
PCIRST bit B-17
PCM digital stream 9-10
PD (power down) bit 11-24, 11-25, 11-31
PD bit B-20
PD bit, in PWRP1 and PWRP2 registers 11-34
PD bit, using to power down DSP 11-24, 11-31
PDC registers B-4
PDC waitstates 9-29

Performance, maximizing, of DSP algorithms 11-11, 11-12
Peripheral device control (PDC) registers B-4
Peripheral Device Control Register Access 8-12
Peripheral Device Control Registers 8-9
Peripheral Device Register Groups B-4
Peripheral interrupt E-1
Peripheral Registers B-2
Peripherals G-5
Peripherals supported by the serial interface 9-10
Phase locked loop (PLL) 11-10
Phase locked loop (PLL), internal 11-7
Phase locked loop clock generators B-23
Pin descriptions 11-3
  AC'97 11-5
  crystal/configuration 11-4
  emulator 11-5
  I/O 11-6
  PCI/USB 11-3
  power supply 11-6
  Serial EEPROM 11-5
Pin listing for AC '97 9-26
Pin loading 11-30
Pin names 1-27
Pins
  AC'97, descriptions 11-5
  ACRST# 9-26
  BITCLK 9-26
  CLKSEL 11-7
  Crystal, descriptions 11-4
  Definitions 11-3
  Emulator, descriptions 11-5
  Flag 11-22
  I/O, descriptions 11-6
  low power state 11-37
  PCI/USB, descriptions 11-3
  Power Supply, descriptions 11-6
  Processor 11-3
  Processor control 11-1
  Programmable flag B-3
  SDI 9-25
  SDO 9-26
  Serial EEPROM, descriptions 11-5
  SYNC 9-26
pins 9-19, 9-26
Pipeline (*See Instruction pipeline*)
Platform States 11-30
PLL 11-10
PLL (Phase Locked Loop) clock generators B-23
PLL, internal 11-7
PME bit B-19
PME_EN bit B-18
PME_Enable state 11-30
PMIEN bit B-21
PMINT bit B-22
PMWE bit B-21
Pop/Push instruction 3-14
Pop/Push instructions 3-43
PORST 11-1, 11-13, 11-35

PORST signal 11-13
Porting from previous DSPs
ALU sign (AS) status 2-18
Circular buffer addressing 4-15
DAG instruction syntax 4-11
DAG registers 4-3
Data register file 2-35
Multiplier dual accumulators 2-28
Multiplier feedback support 2-36
Normalize operation 2-46
Secondary DAG registers 4-6
Shifter results (SR) register 2-50
Post-modify addressing 4-1, 4-22, G-5
Instruction syntax 4-9
Power conservation 11-29
Power consumption limits 11-26
Power consumption, lowest 11-30, 11-36, 11-37
Power down modes 11-29
Power management 11-28
Power Management Functions B-18
Power Management Interactions 8-9
Power Management Register Interactions between Functions 8-10
Power management states 11-28
AC'97 11-28
PCI 11-28
software-controlled 11-28
USB 11-28
Power On Reset 11-13
Power regulators 11-26, 11-27
Power states 11-23
AC'97 11-23
Clock crystal 11-23
PCI 11-23
USB 11-23
Power supplies 11-25
Power systems 11-41
Power, AC'97 low power mode 11-38
Power, low 11-29
Powerdown 11-29, 11-30, 11-31, 11-35, 11-36, 11-39
Powerdown interrupt 11-39
Powerdown interrupt instructions 11-39
Powerdown interrupt mask (PWDN) bit A-19
Powerdown mode 11-35, 11-36
Powerdown states
Attributes 9-32
Power-down transitions 9-34
Powerdown, aborting 11-24, 11-31
Powerdown, exiting 11-35, 11-36
Powered down, both DSPs 11-25, 11-32
Powering down DSP 11-24, 11-31
Powering down the AC'97 11-33
Powering down the link
Cautions 9-27
Powering down the PCI 11-32
Powering down the USB 11-32
Powering up DSP 11-24, 11-31

Powerup 11-24
Power-up transitions 9-33
Powerup, aborting 11-24, 11-31
PR4 bit 9-27, 9-34, 9-35
Precision 1-5, G-5
Prefetch address (PA) stage 3-8
Pre-modify addressing 4-1, 4-22, G-5
  Instruction syntax 4-9
Primary codec 9-27
Primary registers 2-23, 2-36, 2-57
PRn bits 9-34
Processor control pins 11-1
Processor, resetting 11-13
Program Control Interrupt (PCI) bit 7-22
Program counter (PC) register 1-11, 3-3
Program Counter (PC) relative address G-2
Program counter (PC) relative address 3-15
Program counter (PC) stack 3-5
Program flow 3-2, 3-7, 11-35
Program memory bus Exchange (PX) register 5-5, 5-6
Program memory bus exchange (PX) register A-3, A-4, A-26
Program sequencer 1-3, 1-5, 3-1
  Instructions 3-42
Programmable Flag Data register 11-22
Programmable flag pins B-3
Programming information 1-1
Protection diodes 11-25, 11-26
Protocol summary for AC '97 9-27
PS0 platform state 11-30
PS1 platform state 11-31
PU bit 11-24, 11-25, 11-31, B-20
PU bit, in PWRP1 and PWRP2 registers 11-34, 11-35
PU bit, using to power up DSP 11-24, 11-31
Purpose (of text) 1-1
Push/Pop instructions
  Restrictions 3-19
PWRP1 register 11-24, 11-31
  PD bit 11-34
  PU bit 11-34, 11-35
PWRP2 register 11-24, 11-31
  PD bit 11-34
  PU bit 11-34, 11-35
PWRST bit B-18

Q

Quotient bit 2-18
Quotient, ALU (AQ) bit A-10

R

RDIS bit B-15
Reading unpopulated codec IDs 9-29
Reading, recommended 11-41
Rebooting, software-forced 11-16
Receive FIFO empty (RFE) bit 9-7
Receive overflow (RO) bit 9-8
Recommended reading 11-41
REG bit 9-20

REGD bit B-16
Register
  SRCTL 9-37
Register and Bit #Defines File 8-94
Register and Bit #Defines File
    (def2192.h) 8-94, B-95
Register bit definitions
  AC '97 link status 9-19
  AC '97 slot enable 9-22
Register files G-2
Register files (*See Data register files*)
Register Group Descriptions B-11
Register map for DSP core 9-9
Register names 1-27
Registers
  AC'97 codec control/status 9-29
  AC'97 control 9-13
  AC97LCTL 9-15, 9-27, B-42
  AC97SEN B-43
  AC97SIF 9-24, B-44
  AC97SREQ B-44
  AC97STAT 9-12, 9-19, B-42
  AC97SVAL 9-22, B-43
  AC98TLCTL B-42
  Access to AC'97 codec
    control/status 9-28
  ACRQ 9-24
  ACSE 9-21, 9-36
  Address 9-8, 9-9
  CB_FE0 B-36
  CB_FPS0 B-38
  Count 9-8
  CPIOCFG B-24
  Destination 9-9
  DSP core A-2
  FEM B-37
  FIFO 9-1
  FIFO control and status 9-3
  FIFO DMA address 9-8
  FIFO DMA count 9-9
  FIFO DMA current count 9-8
  FIFO DMA next address 9-9
  FIFO receive control and status
    9-5
  FIFO transmit control and status
    9-3
  GPIOCTL B-24
  GPIOPDN B-25
  GPIOPOL B-24
  GPIOPUP B-25
  GPIOSTAT B-24
  GPIOSTKY B-24
  GPIOWCTL B-24
  IMASK E-1
  Interrupt/Powerdown 11-24,
    11-31
  IRPTL E-1
  Load latencies A-5
  MBXCTL 6-24
  multifunction instructions 11-12
  NextAddress 9-8
  NxtAddress 9-9
  Programmable Flag Data 11-22
  PWRP1 11-24, 11-31, 11-34,
    11-35
  PWRP2 11-24, 11-31, 11-34,
    11-35
  Source 9-9

SPROMCTL B-28
STCTL 9-37
STCTLx 9-2
SYSCON B-14, B-33
TCOUNT D-2, D-3, D-6
timer
period D-1
TCOUNT D-2…D-6
TPERIOD D-2…D-6
TSCALE D-2…D-6
TPERIOD D-2, D-3, D-6
TSCALE D-2, D-3, D-6
TSCLCNT D-2, D-3, D-6
RegistersPWRP2 11-34
Regulators, power 11-26, 11-27
Related documents 1-25
Relative address (*See Indirect address*)
Reset
commands executed after boot 11-15
PORST 11-13
Power On 11-13
Software 11-14
user code execution 11-15, 11-16
Reset Handler code 11-14
Reset progression 11-14, 11-15, 11-16
Resets 8-14, 11-16
Resetting AC'97 9-12
Resetting the processor 11-13
Resource allocation 9-25
Restart the link
Recommended method 9-35
Restrictions
Loop endings 3-24
Result registers A-2
Result, multiplier mode 2-10
Results
Placement 2-29
Results, clearing, rounding, and saturating 2-30
Return (Rti/Rts) instruction
Restrictions 3-19
Return (Rti/Rts) instructions 3-14, 3-26, 3-43
RFE bit 9-7
Ribbon cables 11-41
RO bit 9-8
Round (RND) operator 2-33
Rounding mode 2-2
Rounding results 2-30
RST bit B-14
RSTD bit B-21
RTI instruction 11-39
RWE bit B-21
RX0 9-2
RX0 DMA bit B-51
Rx0 DMA Channel Interrupt (RX0 DMA) bit B-51
RX1 9-2
RX1 DMA bit B-51
Rx1 DMA Channel Interrupt (RX1 DMA) bit B-51

S

Sample streams
Disabling 9-36

Enabling 9-36
Saturate (SAT) instruction 2-34
Saturating results 2-30
Saturation (ALU saturation mode) G-5
Saturation, ALU 2-11
Saturation, ALU mode 2-10
SCK bit B-29
SCK pin input enable (SCKI) bit B-29
SCK pin status (SCK) bit B-29
SCKI bit B-29
SDA bit B-29
SDA pin input enable (SDAI) bit B-29
SDA pin status (SDA) bit B-29
SDAI bit B-29
SDI (serial data in) pins 9-25
SDI pins 9-26
Used to define the wakeup protocol
Wakeup protocol
SDI pins 9-30
Using with GND 9-26
SDO pin 9-26
Secondary DAG registers enable (SEC_DAG) bit 4-4, 4-5, A-6, A-13
Secondary registers 2-23, 2-36, 2-54, 2-59, 4-4, 4-5, A-11
Swapping to 2-60
Secondary registers enable (SEC_REG) bit 2-59, A-6
Secondary registers for DAGs mode (BSR) enable/disable 3-43
Secondary registers mode (SR) enable/disable 3-43
Select DSP2 / $\overline{\text{DSP1}}$ bit B-49
SEN bit B-29
SEN pin input enable (SENI) bit B-29
SEN pin status (SEN) bit B-29
SENI bit
Bits
SENI B-29
Sequencer (*See Program sequencer*)
Serial bit clock (BIT_CLK) 9-11
Serial interface
Support for peripherals 9-10
Serial port (SPORT)
DMA 7-24
Serial port DMA Enable (SDEN) bit 7-11, 7-19
Serial port Receive Control (SRCTLx) registers 7-18
Serial port Transmit Control (STCTLx) registers 7-18
Serial pulse code modulated (PCM) digital stream 9-10
Set bit (Setbit) instruction 2-20
Set interrupt (Setint) instruction 3-43
Shared memory 5-8
Shared memory space 5-9
Shift, immediate 2-40
Shifter 2-1, 2-37, G-5
Arithmetic formats 2-9

Data types 2-7
Input 2-53
Instructions 2-50
Operations 2-50
Options 2-37, 2-38
Status flags 2-50
Shifter bitwise OR (Or) option 2-55
Shifter block exponent (SB) register 2-38, 2-39, 2-53, A-2, A-4, A-18
Shifter data register A-2
Shifter exponent (SE) register 1-18, 2-38, 2-40, 2-42, 2-44, 2-47, 2-53, 2-54, 2-57, A-2, A-4, A-18
Shifter input (SI) register 1-17, 2-53, 2-54, A-4, A-16, A-17
Shifter overflow (SV) bit 2-31, 2-32, 2-50, 2-51, A-10
Shifter result (SR) register 2-29, 2-32, 2-53, 2-54
SR2 usage 2-41
Shifter result (SR0-2) registers A-2, A-4, A-16
Shifter results (SR) register 1-18, 2-2
Shifter results overflow (SV) bit 2-33
Shifter sign (SS) bit 2-51, 2-55, A-10
Shifter status bits 2-31, A-10
SI 9-14
Sign bit 2-18
Loss through overflow 2-32
Sign extension 2-2, 2-6, 2-41, 2-53, 2-54
Signals
BITCLK 9-27
SYNC 9-11
Signals, clock 11-7
Signed inputs (SS) operator 2-33
Signed magnitude 2-4
Signed multiplier inputs (SS) operator 2-29
Signed Numbers
Two's Complement 2-5
Signed numbers 2-4
Signed, ALU (AS) bit A-9
Signed, shifter (SS) bit A-10
Signed/unsigned inputs (SU) operator 2-33
Signed/unsigned multiplier inputs (SU) operator 2-29
Single cycle operation 2-22, 2-29, 2-36, 2-54, 2-59
Single-step interrupt mask (SSTEP) bit A-19
SLOT bit 9-7
SMSEL bit 9-4
SMSel bit 9-6
Software condition (SWCOND) condition 1-19, 3-41
Software condition (SWCOND) operator 1-19
Software reset 11-14
Software-forced rebooting 11-16
Source registers 9-9
Spill-fill mode 3-37

SPME bit B-19
SPROMCTL register B-28
SRAM (memory) 1-4
SRCTL register 9-37
Stabilization, of oscillator 11-35
Stack
  Explicit operations 3-39
  Implicit operations 3-38
  PC high/low-watermark 3-36
  Registers 3-35
Stack address, PC (STACKA) register A-3, A-4, A-21
Stack interrupt 3-37
Stack interrupt mask (STACK) bit A-19
Stack management
  Implicit/explicit operations 3-37
Stack over/underflow status 3-36
Stack overflow status (STKOVERFLOW) bit 3-36, 3-37, A-15
Stack page, PC (STACKP) register A-3, A-4, A-21
Stack, PC interrupt enable (PCSTKE) bit 3-37, A-20
Stacking, layer 11-41
Start-up delay 11-35
State transitions 9-33
States
  Cold or warm 9-30
Status 8-19
Status stack empty (STSSTKEMPTY) condition 3-43
Status stack empty status (STSSTKEMPTY) bit 3-36, A-15
Status stack overflow (STKOVERFLOW) condition 3-43
Status, arithmetic (ASTAT) register A-2
Status, conditional 3-40
Status, mode (MSTAT) register A-2
STCTL register 9-37
STCTLx register 9-2
Stereo/monaural select (SMSel) bit 9-6
Stopping and restarting the AC '97 link 9-13
Sub-ISA mode
  external clock 11-36
Subroutines G-5
Subroutines, defined 3-1
Subtract instruction 2-17, 2-20, 2-23
Subtract/multiply G-5
Summary B-4
Support (technical or customer) 1-25
SYEN bit 9-15, 9-33, 9-34
SYNC bit 9-20
SYNC pin 9-26
SYNC pulse generator
  Starting 9-33
SYNC signal 9-11
Synchronization delay 11-9
SYSCON register B-33

SYSCON registers B-14
System control registers 5-9, 5-13
System interface 11-1
System status (SSTAT) register A-2, A-4, A-6, A-14
  Effect latency 3-6
  Latency A-6

T

TAbort IEN bit B-54
Target Abort bit B-52
TCB chain loading G-5
TCOUNT register D-2, D-3, D-6
TDM scheme 9-10
Technical support 1-25
Telex for information 1-25
Terminating a loop 3-21
Terminations 11-41
Test bit (Tstbit) instruction 2-20, 3-40
Test clock (TCK) pin 10-1
Test data input (TDI) pin 10-1
Test logic reset ($\overline{\text{TRST}}$) pin 10-1
Test mode select (TMS) pin 10-1
TFE bit 9-5
TFF bit 9-5
The CSTSCHG Signal B-33
The INTA# Signal B-34
Time division-multiplexing (TDM) scheme 9-10
Time slot
  Support of 9-11
Timer 11-17
  architecture D-2
  enabling D-6
  operation D-4
  period register D-1
  registers
    TCOUNT D-2…D-6
    TPERIOD D-2…D-6
    TSCALE D-2…D-6
  resolution D-4
Timer (TI) enable/disable 3-43
Timer block diagram D-3
Timer enable (TIMER) bit A-6, A-13
Timer overrun 11-17
Timer registers D-2, D-3, D-6
Timing specifications 11-1
Toggle bit (Tglbit) instruction 2-20
Top-of-loop address 3-21
TPERIOD register D-2, D-3, D-6
Transfer Control Block (TCB) G-6
Transmission lines 11-41
Tristate versus three-state G-6
True (Forever) condition 1-22
True condition 3-41
TSCALE register D-2, D-3, D-6
TSCLCNT register D-2, D-3, D-6
TTL/CMOS clock, external 11-36
TU bit 9-5
Two's complement 2-5, 2-44
TX0 9-2
TX0 DMA bit B-51
Tx0 DMA Channel Interrupt (TX0 DMA) bit B-51
TX1 9-2
TX1 DMA bit B-51

TX1 DMA Channel Interrupt (TX1 DMA) bit B-51

U

Unbiased rounding 2-14
Underflow, ALU 2-11
Underflow, stack status 3-36
Unsigned 2-4
Unsigned inputs (UU) operator 2-33
Unsigned multiplier inputs (UU) operator 2-29
Unsigned/signed inputs (US) operator 2-33
Unsigned/signed multipler inputs (US) operator 2-29
USB Data Pipe Operations 8-89
USB DSP Register Definitions 8-58
USB DSP Registers B-71
USB Interface 8-25
USB mode
  external clock 11-36
USB power management states 11-28
USB reset 11-14
USB, powering down 11-32
Using the logical OR function 9-26

V

Vaux bit B-17
VGS bits 9-19
Vias 11-41
VisualDSP 1-15
Von Neumann architecture 5-1, G-6

W

Waitstates
  PDC 9-29
Wakeup Enable (WKUP) bit B-37
Wakeup Force (GWKF) bit B-40
Warm power down state, AC'97 11-38
Web site 1-24
WKUP bit B-37
WR/$\overline{RD}$ bit B-49
Wrap around, buffer 4-11, 4-14

X

X-input operand (XOP) A-2
XON bit B-15
XOR operator 2-20
XTALI 11-10

Y

Y operand (YOP) A-2

Z

Zero, ALU (AZ) bit A-9