

ADSP-21368 SHARC® Processor

Hardware Reference

*Includes ADSP-21367, ADSP-21369,
ADSP-21371, ADSP-21375*

Revision 1.0, September 2006

Part Number
82-000100-01

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



Copyright Information

© 2006 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 icon bar, Blackfin, EZ-KIT Lite, SHARC, the SHARC logo, TigerSHARC, and VisualDSP++ are registered trademarks of Analog Devices, Inc.

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

CONTENTS

PREFACE

Purpose of This Manual	xxxi
Intended Audience	xxxi
Manual Contents	xxxii
What's New in This Manual	xxxiv
Technical or Customer Support	xxxv
Supported Processors	xxxvi
Product Information	xxxvi
MyAnalog.com	xxxvii
Processor Product Information	xxxvii
Related Documents	xxxviii
Online Technical Documentation	xxxix
Printed Manuals	xli
Conventions	xlili

INTRODUCTION

Design Advantages	1-1
Architectural Overview	1-6
Processor Core	1-7

Contents

Processor Peripherals	1-7
I/O Processor	1-7
Digital Audio Interface (DAI)	1-9
Digital Peripheral Interface (DPI)	1-10
Development Tools	1-10
Differences From Previous Processors	1-11
I/O Architecture Enhancements	1-11
Instruction Set Enhancements	1-12

I/O PROCESSOR

General Procedure for Configuring DMA	2-2
Core Access to IOP Registers	2-3
Configuring IOP/Core Interaction	2-6
Interrupt-Driven I/O	2-6
Interrupt Latency in Interrupt-Driven Transfers	2-11
Polling/Status-Driven I/O	2-12
DMA Controller Operation	2-13
Chaining DMA Processes	2-14
Transfer Control Block Chain Loading (TCB)	2-16
Setting Up DMA Channel Allocation and Priorities	2-18
Managing DMA Channel Priority	2-19
DMA Bus Arbitration	2-20
Setting Up DMA Parameter Registers	2-24
DMA Transfer Direction	2-24
Data Buffer Registers	2-25

Port, Buffer, and DMA Control Registers	2-26
Addressing	2-29
External Port DMA	2-35
Setting Up and Starting Chained DMA	2-36
Delay Line DMA	2-38
Serial Port DMA	2-40
Setting Up and Starting Chained DMA	2-40
Inserting a TCB in an Active Chain	2-41
Serial Peripheral Interface DMA	2-42
Setting Up and Starting Chained DMA over the SPI	2-42
UART DMA	2-44
Notes On Using DMA With the UART	2-47
Memory-to-Memory DMA	2-48
Summary	2-48
Programming Example	2-49

EXTERNAL PORT

External Memory Interface	3-2
External Memory Interface on the ADSP-2137x Processors	3-3
Direct Execution of Instructions From External Memory	3-3
Throughput and Instruction Execution Rate	3-3
Location of Interrupt Vector Table (IVT)	3-4
Instruction Cache	3-5
Instruction Storage and Packing	3-9
Register Configurations for External Memory Execution	3-15

Contents

EMI Registers and Signals	3-16
External Port Arbitration Logic	3-18
Channel Freezing	3-18
Managing Data Paths	3-18
External Memory Interface Pins	3-19
Asynchronous Memory Interface	3-20
AMI Timing Control	3-21
Wait States	3-21
Bus Idle Cycles	3-22
Bus Hold Cycles	3-23
Setting AMI Modes	3-24
External Memory Reads	3-25
Data Packing	3-25
External Memory Writes	3-26
Data Packing	3-27
Read/Write Throughput	3-28
External Access Addressing	3-28
External Port DMA	3-30
Bootting Through the AMI	3-30
SDRAM Controller	3-30
Definition of Terms	3-31
Timing External Memory Accesses	3-36
Parallel Connection of SDRAMs	3-39
SDRAM Control Register (SDCTL)	3-39

SDRAM Control Status Register (SDSTAT)	3-49
SDRAM Refresh Rate Control Register (SDRRC)	3-49
SDRAM Initialization	3-51
SDRAM Address Mapping	3-51
SDRAM Controller Address Mapping	3-58
SDC Operation	3-58
Single Bank Operation	3-60
Multibank Operation (ADSP-2137x Processors)	3-60
Data Mask (DQM)	3-61
SDC Configuration	3-61
SDC Commands	3-63
Load Mode Register	3-64
Single Bank Activation	3-65
Multibank Activation (ADSP-2137x Processors)	3-66
Single Precharge (ADSP-2137x Processors)	3-66
Precharge All	3-66
Read/Write	3-67
Read/Write (ADSP-2137x Processors)	3-69
Burst Stop (ADSP-2137x Processors)	3-69
Auto-Refresh	3-70
Self-Refresh Mode	3-70
No Operation/Command Inhibit	3-71
Changing System Clock During Runtime	3-73

Contents

SDRAM Timing	3-74
SDRAM Read Optimization	3-75
External Memory Access Restrictions	3-78
Shared Memory Interface	3-79
Shared Memory Bus Arbitration	3-79
Bus Arbitration Protocol	3-82
Bus Arbitration Priority (RPBA)	3-86
Bus Mastership Time-out	3-87
Bus Synchronization After Reset	3-88
Bus Synchronization Notes	3-91
Bus Lock and Semaphores	3-92
Shared Memory Interface Status	3-93
Shared Memory and the SDRAM Controller	3-94
Shared Memory Booting	3-94

DIGITAL AUDIO/DIGITAL PERIPHERAL INTERFACES

Structure of the Interfaces	4-2
DAI/DPI System Design	4-3
Signal Routing Units	4-8
Connecting Peripherals	4-8
Pin Interface	4-10
Pin Buffers as Signal Output Pins	4-11
Pin Buffers as Signal Input Pins	4-12
Bidirectional Pin Buffers	4-13

Making Connections in the SRUs	4-15
DAI/SRU1 Connection Groups	4-18
Group A Connections—Clock Signals	4-19
Group B Connections—Data Signals	4-25
Group C Connections—Frame Sync Signals	4-31
Group D Connections—Pin Signal Assignments	4-36
Group E Connections—Interrupts and Miscellaneous Signals	4-43
Group F—Pin Enable Signals	4-47
DPI/SRU2 Connection Groups	4-51
Group A Connections—Input Routing Signals	4-52
Group B Connections—Pin Assignment Signals	4-56
Group C Connections—Pin Enable Signals	4-60
General-Purpose I/O (GPIO) and Flags	4-64
DAI GPIO and Flags	4-64
DPI GPIO and Flags	4-65
Miscellaneous Signals	4-65
DAI/DPI Interrupt Controller	4-65
Relationship to the Core	4-65
DAI Interrupts	4-66
DPI Interrupts	4-67
High and Low Priority Latches	4-69
Rising and Falling Edge Masks	4-70

Contents

Configuring Peripherals Using SRU1	4-71
Configuring the SPORTs	4-71
Configuring the PCGs	4-72
Configuring Peripherals Using SRU2	4-72
Configuring the SPI	4-72
Choosing the Pin Enable for the SPI Clock	4-72
Configuring the Two Wire Interface	4-73
Using the SRU() Macro to Configure the DAI	4-76

SERIAL PORTS

Features	5-2
Operation Modes	5-3
Serial Port Signals	5-5
Serial Port Signal Sensitivity	5-9
SPORT Operation Modes	5-10
Standard DSP Serial Mode	5-12
Standard DSP Serial Mode Control Bits	5-13
Clocking Options	5-13
Frame Sync Options	5-13
Data Formatting	5-14
Data Transfers	5-15
Status Information	5-15
Left-Justified Sample Pair Mode	5-16
Setting the Internal Serial Clock and Frame Sync Rates	5-17

Left-Justified Sample Pair Mode Control Bits	5-17
Setting Word Length (SLEN)	5-17
Enabling SPORT Master Mode (MSTR)	5-18
Selecting Transmit and Receive Channel Order (FRFS)	5-18
Selecting Frame Sync Options (DIFS)	5-18
Enabling SPORT DMA (SDEN)	5-19
I2S Mode	5-20
Setting the Internal Serial Clock and Frame Sync Rates	5-21
I2S Mode Control Bits	5-21
Setting Word Length (SLEN)	5-22
Enabling SPORT Master Mode (MSTR)	5-23
Selecting Transmit and Receive Channel Order (FRFS)	5-23
Selecting Frame Sync Options (DIFS)	5-23
Enabling SPORT DMA (SDEN)	5-24
Multichannel Operation	5-25
Frame Syncs in Multichannel Mode	5-28
Multichannel Mode Control Bits	5-29
Packed I2S Mode	5-33
Programming Packed I2S Mode	5-34
SPORT Loopback	5-35
Clock Signal Options	5-36
Frame Sync Options	5-37
Framed Versus Unframed Frame Syncs	5-37
Internal Versus External Frame Syncs	5-38

Contents

Active Low Versus Active High Frame Syncs	5-39
Sampling Edge for Data and Frame Syncs	5-39
Early Versus Late Frame Syncs	5-40
Data-Independent Frame Syncs	5-41
Frame Sync Error Detection	5-42
Data Word Formats	5-43
Word Length	5-43
Endian Format	5-45
Data Packing and Unpacking	5-45
Data Type	5-46
Companding	5-47
SPORT Control Registers and Data Buffers	5-49
Register Writes and Effect Latency	5-58
Serial Port Control Registers (SPCTLx)	5-59
Transmit and Receive Data Buffers (TXSPxA/B, RXSPxA/B)	5-67
Clock and Frame Sync Frequency Registers (DIVx)	5-69
SPORT Reset	5-71
SPORT Interrupts	5-72
Moving Data Between SPORTs and Internal Memory	5-73
DMA Block Transfers	5-73
Setting Up DMA on SPORT Channels	5-75
SPORT DMA Parameter Registers	5-76
SPORT DMA Chaining	5-81
Single Word Transfers	5-81

SPORT Programming Examples	5-82
----------------------------------	------

SERIAL PERIPHERAL INTERFACE PORTS

Functional Description	6-2
SPI Interface Signals	6-4
SPI Clock Signal (SPICLK)	6-4
SPICLK Timing	6-5
SPI Slave Select Input (SPIDS)	6-6
SPI Flag Signals (SPIFLG3-0)	6-6
Master Out Slave In (MOSI)	6-7
Master In Slave Out (MISO)	6-7
SPI General Operations	6-8
SPI Enable	6-9
Open Drain Mode (OPD)	6-9
Master Mode Operation	6-10
Slave Mode Operation	6-11
Multimaster Operation	6-12
SPI Data Transfer Operations	6-13
SPI Operation Using the Core	6-13
SPI Operation Using DMA	6-14
Master Mode DMA Operation	6-15
Slave Mode DMA Operation	6-19
Changing SPI Configuration	6-21
Switching From Transmit To Receive DMA	6-23
Switching From Receive to Transmit DMA	6-24

Contents

DMA Error Interrupts	6-25
DMA Chaining	6-27
SPI Transfer Formats	6-27
Beginning and Ending an SPI Transfer	6-29
SPI Word Lengths	6-31
8-Bit Word Lengths	6-31
16-Bit Word Lengths	6-32
32-Bit Word Lengths	6-32
Packing	6-32
SPI Interrupts	6-33
Error Signals and Flags	6-35
Mode Fault Error (MME)	6-35
Transmission Error Bit (TUNF)	6-37
Reception Error Bit (ROVF)	6-37
Transmit Collision Error Bit (TXCOL)	6-37
Programming Notes	6-38
Routing SPI Signals Using The DPI	6-38
Programming Examples	6-38

INPUT DATA PORT

Serial Inputs	7-3
Parallel Data Acquisition Port (PDAP)	7-8
Masking	7-9
Packing Unit	7-9
Packing Mode 11	7-9

Packing Mode 10	7-10
Packing Mode 01	7-11
Packing Mode 00	7-11
Clocking Edge Selection	7-12
Hold Input	7-12
PDAP Strobe	7-14
FIFO Control and Status	7-15
FIFO to Memory Data Transfer	7-16
IDP Transfers Using the Core	7-17
Starting an Interrupt-Driven Transfer	7-18
Core Transfer Notes	7-19
IDP Transfers Using DMA	7-20
Simple DMA	7-20
Ping-Pong DMA	7-22
DMA Transfer Notes	7-25
DMA Channel Parameter Registers	7-27
IDP (DAI) Interrupt Service Routines for DMAs	7-28
FIFO Overflow	7-30
Input Data Port Programming Example	7-31
 PULSE WIDTH MODULATION	
PWM Implementation	8-1
PWM Waveforms	8-1
Edge-Aligned Mode	8-2
Center-Aligned Mode	8-3

Contents

Switching Frequencies	8-5
Dead Time	8-6
Duty Cycles	8-7
Duty Cycles and Dead Time	8-8
Over Modulation	8-12
Update Modes	8-15
Single Update	8-15
Double Update	8-15
Configurable Polarity	8-15
PWM Pins and Signals	8-16
Crossover	8-16
PWM Accuracy	8-17
PWM Registers	8-18
Duty Cycles	8-19
Output Enable	8-20
Programming Example	8-21

S/PDIF TRANSMITTER/RECEIVER

AES3/SPDIF Stream Format	9-2
Subframe Format	9-3
Channel Coding	9-5
Preambles	9-6
S/PDIF Transmitter	9-7
Channel Status	9-9
SRU1 Signals for the S/PDIF Transmitter	9-10

S/PDIF Transmitter Registers	9-12
Modes of Operation	9-12
Standalone Mode	9-13
Structure of the Serial Input Data	9-14
S/PDIF Receiver	9-16
S/PDIF Receiver Registers	9-17
SRU1 Receiver Signals	9-18
Phase-Locked Loop	9-19
Channel Status Decoding	9-19
Compressed or Non-Linear Audio Data	9-20
Emphasized Audio Data	9-21
Single-Channel, Double-Sampling Frequency Mode	9-21
Error Handling	9-22
Interrupts	9-24
DAI Programming Examples	9-24
S/PDIF Transmitter Programming Guidelines	9-24
Control Register	9-24
SRU1 Programming for Input and Output Streams	9-25
Control Register Programming and Enable	9-25
S/PDIF Receiver Programming Guidelines	9-25
Control Register	9-25
SRU1 Programming	9-26
Control Register Programming	9-26
Receiver Locking	9-26

Contents

Status Bits	9-26
Interrupted Data Streams on the Receiver	9-27

ASYNCHRONOUS SAMPLE RATE CONVERTER

Theory of Operation	10-2
Conceptual Model	10-4
Hardware Model	10-7
Sample Rate Converter Architecture	10-8
Group Delay	10-12
SRC Operation	10-12
Enabling the SRC	10-13
Serial Data Ports	10-13
Data Format	10-13
Time-Division Multiplex (TDM) Output Mode	10-15
TDM Input Mode	10-16
Matched-Phase Mode	10-16
Bypass Mode	10-18
De-Emphasis Filter	10-18
Mute Control	10-19
Soft Mute	10-20
Hard Mute	10-20
Auto Mute	10-20
SRC Registers	10-21
Programming the SRC Module	10-22
SRC Control Register Programming	10-22

SRU Programming	10-22
SRC Mute-Out Interrupt	10-23
Sample Rate Ratio	10-23
Programming Summary	10-23

UART PORT CONTROLLER

Serial Communications	11-2
UART Control and Status Registers	11-3
UARTxLCR Registers	11-3
UARTxLSR Register	11-4
UARTxTHR Register	11-4
UARTxRBR Register	11-5
UARTxIER Register	11-7
UARTxIIR Register	11-9
UARTxDLL and UARTxDLH Registers	11-11
UARTxSCR Register	11-12
UARTxMODE Register	11-13
I/O Mode	11-13
Packing Mode	11-15

TWO WIRE INTERFACE CONTROLLER

Overview	12-1
Architecture	12-2
Register Descriptions	12-4
TWI Master Internal Time Register	12-4

Contents

TWIDIV Register	12-5
Slave Mode Control Register	12-5
Slave Mode Address Register	12-6
Slave Mode Status Register	12-6
Master Mode Control Register	12-6
Master Mode Address Register	12-6
Master Mode Status Register	12-7
FIFO Control Register	12-7
FIFO Status Register	12-7
Interrupt Source Register	12-7
Interrupt Enable Register	12-8
8-Bit Transmit FIFO Register	12-8
16-Bit Transmit FIFO Register	12-8
8-Bit Receive FIFO Register	12-9
16-Bit Receive FIFO Register	12-10
Data Transfer Mechanics	12-10
Clock Generation and Synchronization	12-11
Bus Arbitration	12-12
Start and Stop Conditions	12-12
General Call Support	12-14
Fast Mode	12-14
Programming Examples	12-15
General Setup	12-15
Slave Mode	12-15

Master Mode Clock Setup	12-17
Master Mode Transmit	12-17
Master Mode Receive	12-18
Repeated Start Condition	12-19
Transmit/Receive Repeated Start Sequence	12-19
Receive/Transmit Repeated Start Sequence	12-21
Electrical Specifications	12-22

PRECISION CLOCK GENERATORS

Clock Outputs	13-3
Frame Sync Outputs	13-4
Normal Mode	13-5
Bypass Mode	13-6
Frame Sync Output Synchronization With an External Clock	13-7
Frame Sync	13-8
Phase Shift	13-9
Phase Shift Settings	13-10
Pulse Width	13-10
Bypass Mode	13-12
Bypass as a Pass Through	13-12
Bypass as a One-Shot	13-13
Programming Examples	13-14
PCG Setup for I2S or Left-Justified DAI	13-15
Clock and Frame Sync Divisors PCG Channel B	13-20
PCG Channel A and B Output Example	13-23

SYSTEM DESIGN

Processor Pin Descriptions	14-2
Pin Multiplexing	14-2
Choosing EP Data Mode	14-6
Interrupt and Timer Pins	14-8
Core-Based Flag Pins	14-8
Programming Flags	14-9
RESETOUT/CLKOUT/RUNRSTIN	14-12
JTAG Interface Pins	14-12
Clock Derivation	14-13
Power Management Control Register	14-14
PLL Programming Examples	14-16
Phase-Locked Loop Startup	14-19
RESET and CLKIN	14-20
Running Reset (ADSP-2137x)	14-22
System Design Considerations	14-23
Running Reset Control Register (RUNRSTCTL)	14-25
Programming The RUNRSTCTL Register	14-26
Reset Generators	14-27
Timing Specifications	14-28
Input Synchronization Delay	14-32
Conditioning Input Signals	14-32
RESET Input Hysteresis	14-33

Designing for High Frequency Operation	14-33
Clock Specifications and Jitter	14-33
Other Recommendations and Suggestions	14-34
Decoupling Capacitors and Ground Planes	14-35
Oscilloscope Probes	14-35
Recommended Reading	14-36
Booting	14-37
External Port Booting	14-39
Booting Through the AMI	14-39
Shared Memory Booting	14-40
SPI Port Booting	14-42
32-Bit SPI Host Boot	14-43
16-Bit SPI Host Boot	14-44
8-Bit SPI Host Boot	14-46
Slave Boot Mode	14-47
Master Boot	14-48
Booting From an SPI Flash	14-51
Booting From an SPI PROM (16-Bit address)	14-52
Booting From an SPI Host Processor	14-52
Data Delays, Latencies, and Throughput	14-52
Execution Stalls	14-53
DAG Stalls	14-54
Memory Stalls	14-54
IOP Register Stalls	14-55

Contents

DMA Stalls	14-56
IOP Buffer Stalls	14-56

REGISTER REFERENCE

I/O Processor Registers	A-2
Notes on Reading Register Drawings	A-3
System Control Register (SYSCTL)	A-5
System Status Register (SYSTAT)	A-9
External Port Registers	A-10
External Port Control Register (EPCTL)	A-10
External Port DMA Control Registers (DMACx)	A-14
AMI Control Registers (AMICTLx)	A-17
AMI Status Register (AMISTAT)	A-20
SDRAM Control Register (SDCTL)	A-21
SDRAM Control Status Register (SDSTAT)	A-26
SDRAM Refresh Rate Control Register (SDRRC)	A-26
Memory-to-Memory DMA Register	A-28
Serial Port Registers	A-29
SPORT Serial Control Registers (SPCTLx)	A-29
SPORT Multichannel Control Registers (SPMCTLx)	A-40
SPORT Transmit Buffer Registers (TXSPx)	A-43
SPORT Receive Buffer Registers (RXSPx)	A-44
SPORT Divisor Registers (DIVx)	A-44
SPORT Count Registers (SPCNTx)	A-45
SPORT Active Channel Select Registers (SPxCSy)	A-46

SPORT Compand Registers (SPxCCSy)	A-47
SPORT Error Control Register (SPERRCTLx)	A-48
SPORT Error Status Register (SPERRSTAT)	A-49
SPORT DMA Index Registers (IISPx)	A-50
SPORT DMA Modifier Registers (IMSPx)	A-50
SPORT DMA Count Registers (CSPx)	A-51
SPORT Chain Pointer Registers (CPSPx)	A-51
Serial Peripheral Interface Registers	A-52
SPI Control Registers (SPICTL, SPICTLB)	A-52
SPI Port Status (SPISTAT, SPISTATB) Registers	A-56
SPI Port Flags Registers (SPIFLG, SPIFLGB)	A-58
SPI Receive Buffer Registers (RXSPI, RXSPIB)	A-59
RXSPI Shadow Registers (RXSPI_SHADOW, RXSPIB_SHADOW)	A-59
SPI Transmit Buffer Registers (TXSPI, TXSPIB)	A-59
SPI Baud Rate Registers (SPIBAUD, SPIBAUDB)	A-60
SPI DMA Registers	A-61
SPI DMA Configuration Registers (SPIDMAC, SPIDMACB)	A-62
SPI DMA Start Address Registers (IISPI, IISPIB)	A-64
SPI DMA Address Modify Registers (IMSPI, IMSPIB)	A-64
SPI DMA Word Count Registers (CSPI, CSPIB)	A-64
SPI DMA Chain Pointer Registers (CPSPI, CPSPIB)	A-65
Input Data Port Registers	A-65
Input Data Port Control Register 0 (IDP_CTL0)	A-66

Contents

Input Data Port Control Register 1 (IDP_CTL1)	A-68
Input Data Port FIFO Register (IDP_FIFO)	A-69
Input Data Port DMA Control Registers	A-70
IDP_DMA_Ix	A-70
IDP_DMA_Mx	A-71
IDP_DMA_Cx	A-71
Input Data Port Ping-Pong DMA Registers	A-72
IDP Ping-Pong Index Registers (IDP_DMA_IxA)	A-72
IDP Ping-Pong Count Registers (IDP_DMA_PCx)	A-73
Parallel Data Acquisition Port Control Register (IDP_PP_CTL)	A-74
Pulse Width Modulation Registers	A-78
PWM Global Control Register (PWMGCTL)	A-78
PWM Global Status Register (PWMGSTAT)	A-79
PWM Control Register (PWMCTLx)	A-80
PWM Status Registers (PWMSTATx)	A-81
PWM Period Registers (PWMPERIODx)	A-81
PWM Output Disable Registers (PWMSEGx)	A-82
PWM Polarity Select Registers (PWMPOLx)	A-83
PWM Channel Duty Control Registers (PWMAx, PWMBx)	A-84
PWM Channel Low Duty Control Registers (PWMALx, PWMBLx)	A-84
PWM Dead Time Registers (PWMDTx)	A-85

Sony/Philips Digital Interface Registers	A-86
Transmitter Control Register (DITCTL)	A-86
Left Channel Status for Subframe A Registers (DITCHANAx)	A-89
Right Channel Status for Subframe B Registers (DITCHANBx)	A-90
User Bits Buffer Registers for Subframe A Registers (DITUSRBITAx)	A-90
User Bits Buffer Registers for Subframe B Registers (DITUSRBITBx)	A-91
Receiver Control Register (DIRCTL)	A-92
Receiver Status Register (DIRSTAT)	A-94
Left Channel Status for Subframe A Register (DIRCHANL)	A-96
Right Channel Status for Subframe B Register (DIRCHANR)	A-96
Sample Rate Converter Registers	A-97
SRC Control Registers (SRCCTLx)	A-97
SRC Mute Register (SRCMUTE)	A-107
SRC Ratio Registers (SRCRATx)	A-108
DAI/DPI Registers	A-109
Digital Audio Interface Status Register (DAI_STAT)	A-109
DAI Resistor Pull-up Enable Register (DAI_PIN_PULLUP)	A-111
DAI Pin Buffer Status Register (DAI_PIN_STAT)	A-112
DAI Interrupt Controller Registers	A-112

Contents

DPI Resistor Pull-up Enable Register (DPI_PIN_PULLUP)	A-115
DPI Pin Buffer Status Register (DPI_PIN_STAT)	A-116
DPI Interrupt Controller Registers	A-116
UART Control and Status Registers	A-118
Line Control Registers (UARTxLCR)	A-118
Line Status Registers (UARTxLSR)	A-120
Transmit Hold Registers (UARTxTHR)	A-121
Receive Buffer Registers (UARTxRBR)	A-122
Interrupt Enable Registers (UARTxIER)	A-123
Interrupt Identification Registers (UARTxIIR)	A-124
Divisor Latch Registers (UARTxDLL, UARTxDLH)	A-125
Scratch Registers (UARTxSCR)	A-126
Mode Registers (UARTxMODE)	A-126
UART DMA Registers	A-127
DMA Control Registers (UARTxTXCTL, UARTxRXCTL)	A-128
DMA Status Registers (UARTxTXSTAT, UARTxRXSTAT)	A-129
Two Wire Interface Registers	A-130
Master Internal Time Register (TWIMITR)	A-131
Clock Divider Register (TWIDIV)	A-132
Slave Mode Control Register (TWISCTL)	A-133
Slave Address Register (TWISADDR)	A-135
Slave Status Register (TWISSTAT)	A-135

Master Control Register (TWIMCTL)	A-136
Master Address Register (TWIMADDR)	A-139
Master Status Register (TWIMSTAT)	A-140
FIFO Control Register (TWIFIFOCTL)	A-143
FIFO Status Register (TWIFIFOSTAT)	A-145
Interrupt Source Register (TWIIRPTL)	A-147
Interrupt Enable Register (TWIIMASK)	A-150
8-Bit Transmit FIFO Register (TXTWI8)	A-152
16-Bit Transmit FIFO Register (TXTWI16)	A-153
8-Bit Receive FIFO Register (RXTWI8)	A-154
16-Bit Receive FIFO Register (RXTWI16)	A-154
Precision Clock Generator Registers	A-155
Control Registers (PCG_CTLxx)	A-155
PCG Pulse Width Registers	A-158
PCG Frame Synchronization Registers (PCG_SYNCx)	A-160
Peripheral Interrupt Priority Control Registers	A-164
Peripheral Interrupt Priority Control Registers (PICRx)	A-164
Peripheral Interrupt Priority0 Control Register (PICR0)	A-167
Peripheral Interrupt Priority1 Control Register (PICR1)	A-168
Peripheral Interrupt Priority2 Control Register (PICR2)	A-169
Peripheral Interrupt Priority3 Control Register (PICR3)	A-170

Contents

Power Management Control Register (PMCTL)	A-170
Hardware Breakpoint Control Register	A-175
Enhanced Emulation Status Register	A-179

INTERRUPTS

Interrupt Vector Tables	B-1
Interrupt Priorities	B-4
Interrupt Registers	B-6
Interrupt Register (LIRPTL)	B-6
Interrupt Latch Register (IRPTL)	B-13
Interrupt Mask Register (IMASK)	B-18
Interrupt Mask Pointer Register (IMASKP)	B-22

INDEX

PREFACE

Thank you for purchasing and developing systems using the ADSP-21367/8/9 and ADSP-2137x SHARC® processors from Analog Devices.

Purpose of This Manual

The *ADSP-21368 SHARC Processor Hardware Reference* contains information about the architecture and assembly language for ADSP-21367/8/9 and ADSP-2137x. These are 32-bit, fixed- and floating-point digital signal processors from Analog Devices for use in computing, communications, and consumer applications.

The manual provides information on the processor's I/O architecture and the operation of the peripherals associated with each model.

Intended Audience

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

Manual Contents

The manual consists of:

- Chapter 1, [“Introduction”](#)
Provides an architectural overview of the ADSP-21367/8/9 and ADSP-2137x SHARC processors.
- Chapter 2, [“I/O Processor”](#)
Describes ADSP-21367/8/9 and ADSP-2137x processors input/output processor architecture and direct memory accesses (DMA) for the peripherals that have this feature.
- Chapter 3, [“External Port”](#)
Describes the operation of the asynchronous memory interface (AMI).
- Chapter 4, [“Digital Audio/Digital Peripheral Interfaces”](#)
Provides information about the digital applications interface (DAI) which allows you to attach an arbitrary number and a variety of peripherals to the ADSP-21367/8/9 and ADSP-2137x processors while retaining high levels of compatibility.
- Chapter 5, [“Serial Ports”](#)
Describes the up to eight dual data line serial ports. Each SPORT contains a clock, a frame sync, and two data lines that can be configured as either a receiver or transmitter pair.
- Chapter 6, [“Serial Peripheral Interface Ports”](#)
Describes the operation of the SPI port. SPI devices communicate using a master-slave relationship and can achieve high data transfer rates because they can operate in full-duplex mode.
- Chapter 7, [“Input Data Port”](#)
Discusses the function of the input data port (IDP) which provides a low overhead method of routing signal routing unit (SRU) signals back to the core’s memory.

- Chapter 8, “[Pulse Width Modulation](#)”
Describes the implementation and use of the pulse width modulation module which provides a technique for controlling analog circuits with the microprocessor’s digital outputs.
- Chapter 9, “[S/PDIF Transmitter/Receiver](#)”
Provides information on the use of the Sony/Philips Digital Interface which is a standard audio file transfer format that allows the transfer of digital audio signals from one device to another without having to be converted to an analog signal.
- Chapter 10, “[Asynchronous Sample Rate Converter](#)”
Provides information on the sample rate converter module. This module performs synchronous or asynchronous sample rate conversions across independent stereo channels, without using any internal processor resources.
- Chapter 11, “[UART Port Controller](#)”
Describes the operation of the Universal Asynchronous Receiver/Transmitter (UART) which is a full-duplex peripheral compatible with PC-style industry-standard UART.
- Chapter 12, “[Two Wire Interface Controller](#)”
The two wire interface is fully compatible with the widely used I²C bus standard. It is designed with a high level of functionality and is compatible with multi-master, multi-slave bus configurations.
- Chapter 13, “[Precision Clock Generators](#)”
Details the precision clock generators (PCG) each of which generates a pair of signals derived from a clock input signal.
- Chapter 14, “[System Design](#)”
Describes system design features of the ADSP-21367/8/9 and ADSP-2137x processors. These include power, reset, clock, JTAG, and booting, as well as pin descriptions and other system level information.

What's New in This Manual

- Appendix A, “[Register Reference](#)”
Provides a graphical presentation of all registers and describes the bit usage in each register.
- Appendix B, “[Interrupts](#)”
Provides a complete listing of the registers that are used to configure and control interrupts.



This hardware reference is a companion document to the *ADSP-2136x/ADSP-2137x SHARC Processor Programming Reference*. The programming reference provides information relating to the processor core, such as processing elements, internal memory, and program sequencing. It also provides programming specific information, such as complete descriptions of the ADSP-21xxx instruction set and the compute operations, including their assembly language syntax and opcode fields.

What's New in This Manual

Revision 1.0 of the *ADSP-21368 SHARC Processor Hardware Reference* is the first general release of this manual. The following changes should be noted.

- In the preliminary version this manual was titled *ADSP-2136x SHARC Processor Hardware reference for the ADSP-21367/8/9 Processors*. The title change to *ADSP-21368 SHARC Processor Hardware Reference* was done to reflect the fact that the ADSP-21368 processor contains the super set of features of the ADSP-21367 and ADSP-21369 models as well as the new ADSP-21371 and ADSP-21375 models.
- This version of the manual contains information about the ADSP-21371 and ADSP-21375 SHARC processors. These new models contain the same core as the ADSP-21367/8/9 processors

and as such are completely code compatible. The primary differences in these new models is the ability to execute programs from external memory and a running reset feature.

For more information on these topics, see “[Direct Execution of Instructions From External Memory](#)” on page 3-3 and “[Running Reset \(ADSP-2137x\)](#)” on page 14-22.

Technical or Customer Support

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

- Visit the Embedded Processing and DSP products Web site at <http://www.analog.com/processors/manuals>
- E-mail tools questions to processor.tools.support@analog.com
- E-mail processor questions to processor.support@analog.com (World wide support)
processor.europe@analog.com (Europe support)
processor.china@analog.com (China support)
- Phone questions to **1-800-ANALOGD**
- Contact your Analog Devices, Inc. local sales office or authorized distributor
- Send questions by mail to:

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

Supported Processors

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

Blackfin® (ADSP-BFxxx) Processors

The name *Blackfin* refers to a family of 16-bit, embedded processors. VisualDSP++ currently supports the following Blackfin families: ADSP-BF53x, ADSP-BF54x, and ADSP-BF56x.

SHARC (ADSP-21xxx) Processors

The name *SHARC* refers to a family of high-performance, 32-bit, floating-point processors that can be used in speech, sound, graphics, and imaging applications. VisualDSP++ currently supports the following SHARC families: ADSP-2106x, ADSP-2116x, ADSP-2126x, ADSP-2136x, and ADSP-2137x.

TigerSHARC® (ADSP-TSxxx) Processors

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

Product Information

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

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

MyAnalog.com

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

Registration

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

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

Processor Product Information

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

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

- E-mail processor questions to
processor.support@analog.com (World wide support)
processor.europe@analog.com (Europe support)
processor.china@analog.com (China support)

Product Information

- Fax questions or requests for information to
1-781-461-3010 (North America)
+49-89-76903-157 (Europe)
- Access the FTP Web site at
`ftp ftp.analog.com` or `ftp://137.71.25.69`
`ftp://ftp.analog.com`

Related Documents

The following publications that describe the ADSP-2136x SHARC processors (and related processors) can be ordered from any Analog Devices sales office:

- *ADSP-21362 SHARC Processor Data Sheet*
- *ADSP-21363 SHARC Processor Data Sheet*
- *ADSP-21364 SHARC Processor Data Sheet*
- *ADSP-21365 SHARC Processor Data Sheet*
- *ADSP-21366 SHARC Processor Data Sheet*
- *ADSP-21367/ADSP-21368/ADSP-21369 SHARC Processor Data Sheet*
- *ADSP-21371 SHARC Processor Preliminary Data Sheet*
- *ADSP-21375 SHARC Processor Preliminary Data Sheet*
- *ADSP-2136x/ADSP-2137x SHARC Processor Programming Reference*

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

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

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

<http://www.analog.com/processors/manuals>

Online Technical Documentation

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

Each documentation file type is described as follows.

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

Product Information

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

Accessing Documentation From VisualDSP++

From the VisualDSP++ environment:

- Access VisualDSP++ online Help from the Help menu's **C**ontents, **S**earch, and **I**ndex commands.
- Open online Help from context-sensitive user interface items (toolbar buttons, menu commands, and windows).

Accessing Documentation From Windows

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

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

Using Windows Explorer

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

Using the Windows Start Button

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

Accessing Documentation From the Web

Download manuals at the following Web site:

<http://www.analog.com/processors/resources/manuals>

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

Printed Manuals

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

VisualDSP++ Documentation Set

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

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

Product Information

Hardware Tools Manuals

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

Processor Manuals

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




Data Sheets

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

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

Conventions

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

Example	Description
Close command (File menu)	Titles in reference sections indicate the location of an item within the VisualDSP++ environment's menu system. For example, the Close command appears on the File menu.
{this that}	Alternative items in syntax descriptions appear within curly brackets and separated by vertical bars; read the example as <i>this</i> or <i>that</i> . One or the other is required.
[this that]	Optional items in syntax descriptions appear within brackets and separated by vertical bars; read the example as an optional <i>this</i> or <i>that</i> .
[this,...]	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipse; read the example as an optional comma-separated list of <i>this</i> .
.SECTION	Commands, directives, keywords, and feature names are in text with letter gothic font.
<i>filename</i>	Non-keyword placeholders appear in text with italic style format.
	Note: For correct operation, ... A Note: provides supplementary information on a related topic. In the online version of this book, the word Note appears instead of this symbol.
	Caution: Incorrect device operation may result if ... Caution: Device damage may result if ... A Caution: identifies conditions or inappropriate usage of the product that could lead to undesirable results or product damage. In the online version of this book, the word Caution appears instead of this symbol.
	Warning: Injury to device users may result if ... A Warning: identifies conditions or inappropriate usage of the product that could lead to conditions that are potentially hazardous for devices users. In the online version of this book, the word Warning appears instead of this symbol.

Conventions



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

1 INTRODUCTION

The ADSP-21367/8/9 and ADSP-2137x SHARC processors are high performance, 32-bit processors used for high quality audio, medical imaging, communications, military, test equipment, 3D graphics, speech recognition, motor control, imaging, and other applications. By adding on-chip SRAM, integrated I/O peripherals, and an additional processing element for single-instruction, multiple-data (SIMD) support, this processor builds on the ADSP-21000 family DSP core to form a complete system-on-a-chip.

Design Advantages

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

The SHARC processor architecture balances a high performance processor core with high performance program memory (PM), data memory (DM),

Design Advantages

and input/output (I/O) buses. In the core, every instruction can execute in a single cycle. The buses and instruction cache provide rapid, unimpeded data flow to the core to maintain the execution rate.

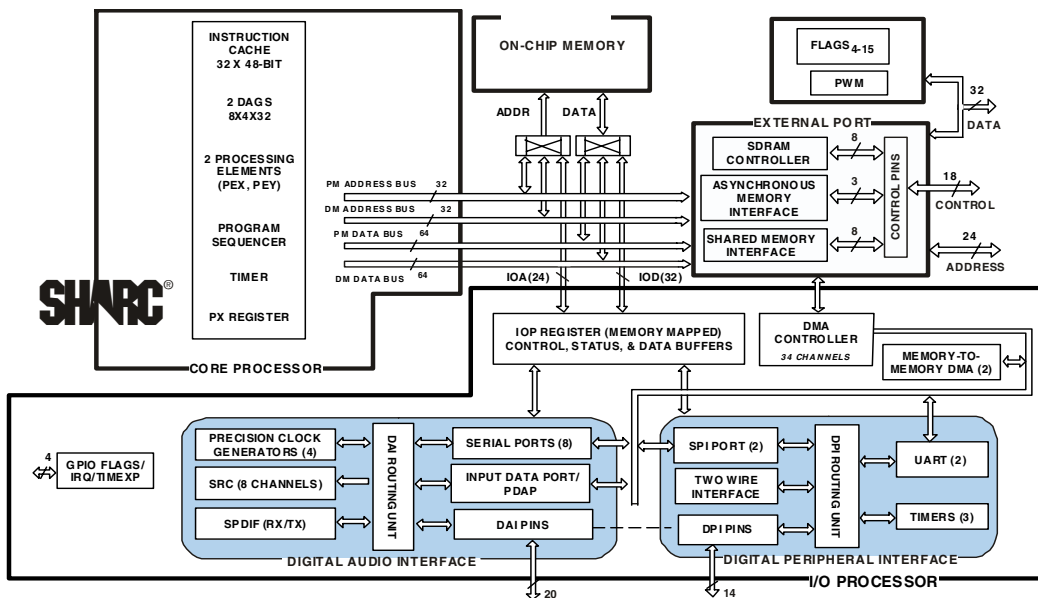
Figure 1-1 shows a detailed block diagram of the processor core and the I/O processor (IOP). This figure illustrates the following architectural features:

- Two processing elements (PE_x and PE_y), each containing 32-bit, IEEE, floating-point computation units—multiplier, arithmetic logic unit (ALU), shifter, and data register file
- Program sequencer with related instruction cache, interval timer, and data address generators (DAG1 and DAG2)
- An SDRAM controller that provides an interface up to four separate banks of industry-standard SDRAM devices or DIMMs, at speeds up to f_{SCLK}
- Up to 2M bits of SRAM and 6M bits of on-chip, mask-programmable ROM
- IOP with integrated direct memory access (DMA) controller, serial peripheral interface (SPI) compatible port, and serial ports (SPORTs) for point-to-point multiprocessor communications
- A variety of audio centric peripheral modules including a Sony/Philips Digital Interface (S/PDIF), sample rate converter (SRC) and pulse width modulation (PWM). Table 1-1 on page 1-5 provides details on these and other features for the current members of the ADSP-21367/8/9 and ADSP-2137x processors families.
- JTAG test access port for emulation

Figure 1-1 also shows the three on-chip buses: the PM bus, DM bus, and I/O bus. The PM bus provides access to either instructions or data. During a single cycle, these buses let the processor access two data operands

from memory, access an instruction (from the cache), and perform a DMA transfer.

Figure 1-1 also shows the asynchronous memory interface available on the ADSP-21368 processor.



*THE ADSP-21368 PROCESSOR INCLUDES A CUSTOMER-DEFINABLE ROM BLOCK.
PLEASE CONTACT YOUR ANALOG DEVICES SALES REPRESENTATIVE FOR ADDITIONAL DETAILS

Figure 1-1. ADSP-21368 Block Diagram

The ADSP-21367/8/9 and ADSP-2137x processors address the five central requirements for signal processing:

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

Design Advantages

Unconstrained Data Flow. The ADSP-21367/8/9 and ADSP-2137x processors have a Super Harvard Architecture combined with a ten-port data register file. In every cycle, the processor can write or read two operands to or from the register file, supply two operands to the ALU, supply two operands to the multiplier, and receive three results from the ALU and multiplier. The processor's 48-bit orthogonal instruction word supports parallel data transfers and arithmetic operations in the same instruction.

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

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

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

The ADSP-21367/8/9 and ADSP-2137x processors also provide the following features which increase the variety processor applications.

High Bandwidth I/O. The processors contain a dedicated, 6M bits on-chip ROM, an external port, an SPI port, serial ports, digital audio interface (DAI), and JTAG. The DAI incorporates a precision clock generator, input data port, and a signal routing unit.

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

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

Two Serial Peripheral Interfaces (SPI). The primary SPI has dedicated pins and the secondary is controlled through the DAI. The SPI provides master or slave serial boot through the SPI, full-duplex operation, master-slave mode, multimaster support, open drain outputs, programmable baud rates, clock polarities, and phases.

Digital Audio Interface and Digital Peripheral Interface. The digital audio interface (DAI) and the digital peripheral interface (DPI) are comprised of groups of peripherals and their signal routing units (SRU1 and SRU2 respectively). This allows peripherals to be interconnected to suit a wide variety of systems. It also allows the processors to include an arbitrary number and variety of peripherals while retaining high levels of compatibility without increasing pin count.

Signal Routing Units (SRU1/SRU2). The SRUs provide configuration flexibility by allowing software-programmable connections to be made between the DAI/DPI components and the 20 DAI pins and 14 DPI pins.

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

Table 1-1. SHARC Processor Features

Feature	ADSP-21367	ADSP-21368	ADSP-21369	ADSP-21371	ADSP-21375
RAM	2M bit	2M bit	2M bit	1M bit	0.5M bit
ROM	6M bit	6M bit ¹	6M bit ¹	4M bit ¹	2M bit ¹
Audio Decoders in ROM ²	Yes	No	No	No	No

Architectural Overview

Table 1-1. SHARC Processor Features (Cont'd)

Feature	ADSP-21367	ADSP-21368	ADSP-21369	ADSP-21371	ADSP-21375
Pulse Width Modulation	Yes	Yes	Yes	Yes	No
S/PDIF	Yes	Yes	Yes	Yes	No
Shared Memory	No	Yes	No	No	No
SRC Performance	128dB	140dB	128dB	128dB	N/A
Package Option ³	256-ball SBGA 208 Lead MQFP	256-ball SBGA	256-ball SBGA 208 Lead MQFP	208-lead MQFP	208-lead MQFP
Processor Speed	333 MHz	333 MHz	333 MHz	266 MHz	266 MHz

- 1 The ADSP-21367 processor include a customer-definable ROM block. Please contact your Analog Devices sales representative for additional details.
- 2 Audio decoding algorithms include PCM, Dolby Digital EX, PCM, Dolby Digital EX, Dolby Prologic IIx, DTS 96/24, Neo:6, DTS ES, MPEG2 AAC, MPEG2 2channel, MP3, and functions like bass management, delay, speaker equalization, graphic equalization, and more. Decoder/post-processor algorithm combination support will vary depending upon the chip version and the system configurations. Please visit www.analog.com/SHARC for complete information.
- 3 Analog Devices offers these packages in lead-free (Pb) versions.

Architectural Overview

The ADSP-21367/8/9 and ADSP-2137x processors form a complete system-on-a-chip, integrating a large, high speed SRAM and I/O peripherals supported by a dedicated I/O bus. The following sections summarize the features of each functional block in the processor architecture, which appears in [Figure 1-1](#).

Processor Core

The processor core of the ADSP-21367/8/9 and ADSP-2137x processors contain two processing elements (each with three computation units and data register file), a program sequencer, two data address generators, a timer, and an instruction cache. All digital signal processing occurs in the processor core. For complete information, see the ADSP-21367/8/9 and ADSP-2137x SHARC processors.

Processor Peripherals

The term processor peripherals refers to the multiple on-chip functional blocks used to communicate with off-chip devices. The peripherals include the JTAG, UART, serial ports, SPI ports, DAI/DPI components (PCG, timers, and IDP are a few), and any external devices that connect to the processor.

I/O Processor

The ADSP-21367/8/9 and ADSP-2137x processors input/output processor (IOP) manages the off-chip data I/O to alleviate the core of this burden. Up to thirty-four channels of DMA are available on the ADSP-21367/8/9 and ADSP-2137x processors—sixteen via the serial ports, 8 via the input data port, 4 for the UARTs, 2 for the SPI interface, 2 for the external port, and 2 for memory-to-memory transfers. The I/O processor can perform DMA transfers between the peripherals and internal memory at the full core clock speed. The architecture of the internal memory allows the IOP and the core to access internal memory simultaneously with no reduction in throughput.

Serial Ports. The processors feature up to eight synchronous serial ports that provide an inexpensive interface to a wide variety of digital and mixed-signal peripheral devices. The serial ports can operate at up to half of the processor core clock rate with maximum of 50M bits per second. Each serial port features two data pins that function as a pair based on the

same serial clock and frame sync. Accordingly, each serial port has two DMA channels and serial data buffers associated with it to service the dual serial data pins. Programmable data direction provides greater flexibility for serial communications. Serial port data can automatically transfer to and from on-chip memory using DMA. Each of the serial ports offers a TDM multichannel mode (up to 128 channels) and supports μ -law or A-law companding. I²S support is also provided with the ADSP-21367/8/9 and ADSP-2137x processors.

The serial ports can operate with least significant bit first (LSBF) or most significant bit first (MSBF) transmission order, with word lengths from 3 to 32 bits. The serial ports offer selectable synchronization and transmit modes. Serial port clocks and frame syncs can be internally or externally generated.

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

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

SDRAM Controller. The SDRAM controller provides an interface of up to four separate banks of industry-standard SDRAM devices or DIMMs, at speeds up to f_{SCLK} . Fully compliant with the SDRAM standard, each bank has its own memory select line ($\overline{\text{MS0}}\text{--}\overline{\text{MS3}}$), and can be configured to contain between 16M bytes and 256M bytes of memory.

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

Digital Audio Interface (DAI)

The digital audio interface (DAI) unit is a new addition to the SHARC processor peripherals. This set of audio peripherals consists of an interrupt controller, an interface data port, and a signal routing unit, four precision clock generators (PCGs) and three timers. Some family members have an S/PDIF receiver/transmitter and eight channels asynchronous sample rate converters (SRC).

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

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

Signal Routing Unit One (SRU1). Conceptually similar to a “patch-bay” or multiplexer, the SRU provides a group of registers that define the inter-connection of the serial ports, the input data port, the DAI pins, and the precision clock generators.

Development Tools

Digital Peripheral Interface (DPI)

The digital peripheral interface (DPI) unit is a new addition to the SHARC processor peripherals. This set of audio peripherals consists of an interrupt controller, a two wire interface port (TWI), and a signal routing unit, three timers and a Universal Asynchronous Receiver/Transmitter (UART).

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

Two Wire Interface (TWI). The two wire interface (TWI) controller allows a device to interface to an Inter IC bus as specified by the *Philips I²C Bus Specification version 2.1* dated January 2000.

Universal Asynchronous Receiver/Transmitter (UART). A full-duplex peripheral compatible with PC-style, industry-standard UARTs. The UART converts data between serial and parallel formats. The UART includes interrupt handling hardware. Interrupts can be generated from 12 different events.

Signal Routing Unit Two (SRU2). Conceptually similar to a “patch-bay” or multiplexer, SRU2 provides a group of registers that define the inter-connection of the DPI’s peripherals, the DPI pins, and the timers.

Development Tools

The ADSP-21367/8/9 and ADSP-2137x processors are supported by VisualDSP++, an easy-to-use integrated development and debugging environment (IDDE). VisualDSP++ allows you to manage projects from start to finish from within a single, integrated interface. Because the project development and debug environments are integrated, you can move easily between editing, building, and debugging activities.

Differences From Previous Processors

This section identifies differences between the ADSP-21367/8/9 and ADSP-2137x processors and previous SHARC processors: ADSP-21161, ADSP-21160, ADSP-21060, ADSP-21061, ADSP-21062, and ADSP-21065L. Like the ADSP-2116x family, the ADSP-2136x SHARC processor family is based on the original ADSP-2106x SHARC family. The ADSP-21367/8/9 and ADSP-2137x processors preserve much of the ADSP-2106x architecture and is code compatible to the ADSP-21160, while extending performance and functionality. For background information on SHARC processors and the ADSP-2106x family DSPs, see the *ADSP-2106x SHARC User's Manual* or the *ADSP-21065L SHARC DSP Technical Reference*.

I/O Architecture Enhancements

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

The DMA controller supports up to 34 channels compared to 14 channels on the ADSP-21161 processor. DMA transfers occur at clock speed in parallel with full speed processor execution. The ADSP-21367/8/9 and ADSP-2137x processors also provide delay line DMA functionality. This allows processor reads and writes to external delay line buffers (and hence to external memory) with limited core interaction.

In addition to the above, the ADSP-21367/8/9 and ADSP-2137x processors have up to eight serial ports (SPORTs), a 32-bit external memory interface, a universal asynchronous transmitter/receiver (UART) and an I²C compatible interface called the TWI (two wire interface).

Instruction Set Enhancements

The ADSP-21367/8/9 and ADSP-2137x processors provide source code compatibility with the previous SHARC processor family members to the application assembly source code level. All instructions, control registers, and system resources available in the ADSP-2106x core programming model are also available in the ADSP-21367/8/9 and ADSP-2137x processors. Instructions, control registers, or other facilities required to support the new feature set of the ADSP-2116x core include:

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

These name changes can be managed through reassembly by using the development tools to apply the ADSP-21367/8/9 and ADSP-2137x processor symbol definitions header file and linker description file. While these changes have no direct impact on existing core applications, system and I/O processor initialization code and control code do require modifications.

Although the porting of source code written for the ADSP-2106x family to the ADSP-21367/8/9 and ADSP-2137x processors has been simplified, code changes are required to take full advantage of the new features. For more information, see the *ADSP-2136x SHARC Processor Programming Reference*.

2 I/O PROCESSOR


In applications that use extensive off-chip data I/O, programs may find it beneficial to use a processor resource other than the processor core to perform data transfers. The ADSP-21367/8/9 and ADSP-2137x processors contain an I/O processor (IOP) that supports a variety of DMA (direct memory access) operations. Each DMA operation transfers an entire block of data. These operations include the transfer types listed below and shown in [Figure 2-2 on page 2-25](#).

- Internal memory ↔ external memory devices (through the external port)
- Internal memory ← digital audio/digital peripheral interfaces (DAI/DPI)
- Internal memory ↔ serial port I/O
- Internal memory ↔ serial peripheral interface I/O
- Internal memory ↔ UART I/O
- Internal memory ↔ internal memory

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

General Procedure for Configuring DMA

To further increase off-chip I/O, multiple DMAs can occur at the same time. The IOP accomplishes this by managing DMAs of processor memory through the TWI, UART, SPI, input data port (IDP), and serial ports.

-  Accesses to IOP spaces (from the processor core) should not use Type 1 (dual access) or LW instructions.

General Procedure for Configuring DMA

To configure the ADSP-21367/8/9 and ADSP-2137x processors to use DMA, use the following general procedure.

1. Determine which DMA options you want to use:
 - IOP/core interaction method – interrupt-driven or status-driven (polling)
 - DMA transfer method – chained, non-chained, or delay line
 - Channel priority scheme – fixed or rotating
2. Determine how you want the DMA to operate:
 - Set up the data's source and/or destination addresses (INDEX)
 - Set up the word COUNT (data buffer size)
 - Configure the MODIFY values (step size)
3. Configure the peripheral(s):
 - External port (includes AMI, SDRAM)
 - Serial ports (SPORTs)
 - Universal asynchronous receive/transmit (UART)

- Serial peripheral interface ports (SPI)
- Input data port (IDP)

4. Enable DMA

- Set the applicable bits in the appropriate control registers

For peripheral specific DMA information, see the following sections.

- [“External Port DMA” on page 2-35](#)
- [“Serial Port DMA” on page 2-40](#)
- [“Serial Peripheral Interface DMA” on page 2-42](#)
- [“UART DMA” on page 2-44](#)
- [“Memory-to-Memory DMA” on page 2-48](#)

Core Access to IOP Registers

In certain cases, extra core cycles are needed to process register accesses. The access cycles are shown in [Table 2-1](#) and the registers are shown in [Table 2-2](#).

Table 2-1. I/O Processor Stall Conditions

Type Of Access	Number of Core Cycles
Core write ¹	1
Core read ¹	2
Unconditional, isolated I/O processor register write ²	1
Unconditional I/O processor register write after a write ²	2 (back-to-back)
Unconditional I/O processor register read ²	7/8

Core Access to IOP Registers

Table 2-1. I/O Processor Stall Conditions (Cont'd)

Type Of Access	Number of Core Cycles
Aborted conditional I/O processor register read/write ²	3
Conditional I/O processor register read/write ²	9/10

1 Applies to memory-mapped registers from [Table 2-2](#).

2 Applies to all other memory-mapped registers not in [Table 2-2](#).

Table 2-2. Memory-Mapped Emulation/Breakpoint Registers

Register	Description	Address
EEMUIN	Emulator input FIFO	0x30020
EEMUSTAT	Enhanced emulation status	0x30021
EEMUOUT	Emulator output FIFO	0x30022
OSPID	Operating system process ID	0x30023
SYSCTL	System control	0x30024
BRKCTL	Breakpoint control	0x30025
REVPID	Emulation/revision ID	0x30026
PSA1S/E	Instruction breakpoint address number 1 start/end	0x300A0/ 0x300A1
PSA2S/E	Instruction breakpoint address number 2 start/end	0x300A2/ 0x300A3
PSA3S/E	Instruction breakpoint address number 3 start/end	0x300A4/ 0x300A5
PSA4S/E	Instruction breakpoint address number 4 start/end	0x300A6/ 0x300A7
EMUN	Number of breakpoints before EMU interrupt	0x300AE
IOAS/E	I/O address breakpoint start/end	0x300B0/ 0x300B1

Table 2-2. Memory-Mapped Emulation/Breakpoint Registers (Cont'd)

Register	Description	Address
DMA1S/E	Data memory breakpoint address number 1 start/end	0x300B2/ 0x300B3
DMA2S/E	Data memory breakpoint address number 2 start/end	0x300B3/ 0x300B4
PMDAS/E	Program memory breakpoint address start/end	0x300B8/ 0x300B9

In addition to the above, the following situations incur additional stall cycles.

1. An aborted conditional I/O processor register read can cause one or two extra core-clock stall cycles if it immediately follows a write. Such a read is expected to take three core cycles, but it takes four or five.
2. In case of a full write FIFO, the held-off I/O processor register read or write access incurs one extra core-clock cycle.
3. Interrupted reads and writes, if preceded by another write, creates an additional one core cycle stall.

Inside of an interrupt service routine (ISR), a write into an IOP register that clears the interrupt has some latency. During this delay, the interrupt may be generated a second time if the program executes an `RTI` instruction.

For example, in the following code the interrupt isn't cleared instantaneously. During the delay, if the program comes out of the ISR, the interrupt is generated again.

```
/*.... code .....*/
dm(TXSPI) = R0; /* Write to TXSPI FIFO; disable spi;
                  clears the interrupt */
rti;
```

Configuring IOP/Core Interaction

In order to resolve this issue, use one of the following methods.

1. Read an IOP register from the same peripheral block before executing the RTI. This read forces the write to occur first.

```
dm(TXSPI) = R0;      /* Write to TXSPI FIFO */
R0 = dm(SPICTL);     /* Dummy read. This read happens only
                      after write */
rti;
```

2. Add sufficient NOP instructions after a write. In all cases, ten NOP instructions after a write is sufficient to properly update the status.


```
R0 = 0x0;
dm(SPICTL) = R0;      /* Disable spi */
nop; nop; nop; nop; nop;
nop; nop; nop; nop; nop;
rti;
```

Configuring IOP/Core Interaction

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

Interrupt-Driven I/O

Interrupts are generated at the end of a DMA transfer. This happens when the count register for a particular channel decrements to zero. The default interrupt vector locations for each of the channels are listed in [Table 2-3 on page 2-9](#). The interrupt register diagrams and bit descriptions are given in [Appendix B, Interrupts](#) and “DAI Interrupt Controller Registers” on [page A-112](#).

 The processors also have programmable interrupts using the peripheral interrupt priority control registers, `PICRx`. [For more information, see “Peripheral Interrupt Priority Control Registers” on page A-164.](#)

Programs can check the appropriate status or configuration register to determine which channels are performing a DMA or chained DMA.


All DMA channels can be active or inactive. If a channel is active, a DMA is in progress on that channel. The I/O processor indicates the active status by setting the channel's bit in the status register. The only exception to this is the `IDP_DMAx_STAT` bits of the `DAI_STAT` register can become active even if DMA, through some IDP channel, is not intended.

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


- When an unchained (single block) DMA process reaches completion (as the count decrements to zero) on any DMA channel, the I/O processor latches that DMA channel's interrupt. It does this by setting the DMA channel's interrupt latch bit in the `IRPTL`, `LIRPTL`, `DAI_IRPTL_H`, or `DAI_IRPTL_L` registers.
- For chained DMA, the I/O processor generates interrupts in one of two ways:
If `PCI = 1`, bit 19 of the chain pointer register is the program controlled interrupts bit and an interrupt occurs for each DMA in the chain.
If `PCI = 0`, an interrupt occurs at the end of a complete chain. (For more information on DMA chaining, see [“DMA Controller Operation” on page 2-13.](#))
- When a DMA channel's buffer is not being used for a DMA process, the I/O processor can generate an interrupt on single word writes or reads of the buffer. This interrupt service differs slightly for each port. For more information on single-word interrupt-driven transfers, see [“Serial Port Control Registers \(SPCTLx\)” on page 5-59.](#)

Configuring IOP/Core Interaction

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

 The I/O processor only generates a DMA complete interrupt when the channel's count register decrements to zero as a result of actual DMA transfers. Writing zero to a count register does not generate the interrupt. To stop a DMA preemptively, write a one to the count register. This causes one more word to be transferred or received and an interrupt is then generated.

A channel interrupt mask in the `IMASK`, `LIRPTL`, `DAI_IRPTL_PRI`, `DAI_IRPTL_RE`, and `DAI_IRPTL_FE` registers determines whether a latched interrupt is to be serviced or not. When an interrupt is masked, it is latched but not serviced. For more information on the `IMASK` and `LIRPTL` registers, see [“Interrupt Registers” on page B-6](#).

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

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

The digital audio interface (DAI) has two interrupts—the lower priority option (`DAI_LI`) and higher priority option (`DAI_HI`). This allows two interrupts to have priorities that are higher and lower than serial ports.

Table 2-3. Default DMA Interrupt Vector Locations

Associated Register(s)	Bits	Vector Address	Interrupt Name	DMA Channel	Data Buffer
IRPTL/IMASK	14	0x38	SP1I	0	RXSP1A, TXSP1A
LIRPTL	0	0x44	SP0I	2	RXSP0A, TXSP0A
IRPTL/IMASK	15	0x3C	SP3I	4	RXSP3A, TXSP3A
LIRPTL	1	0x48	SP2I	6	RXSP2A, TXSP2A
IRPTL/IMASK	16	0x40	SP5I	8	RXSP5A, TXSP5A
LIRPTL	2	0x4C	SP4I	10	RXSP4A, TXSP4A
IRPTL/IMASK	5	0x58	SP7I	12	RXSP7A, TXSP7A
LIRPTL	19	0x6C	SP6I	14	RXSP6A, TXSP6A
IRPTL/IMASK	14	0x38	SP1I	1	RXSP1B, TXSP1B
LIRPTL	0	0x44	SP0I	3	RXSP0B, TXSP0B
IRPTL/IMASK	15	0x3C	SP3I	5	RXSP3B, TXSP3B
LIRPTL	1	0x48	SP2I	7	RXSP2B, TXSP2B
IRPTL/IMASK	16	0x40	SP5I	9	RXSP5B, TXSP5B
LIRPTL	2	0x4C	SP4I	11	RXSP4B, TXSP4B
IRPTL/IMASK	5	0x58	SP7I	13	RXSP7B, TXSP7B
LIRPTL	19	0x6C	SP6I	15	RXSP6B, TXSP6B
IRPTL/IMASK (high priority option) LIRPTL (low priority option)	11 6	0x2C 0x5C	DAIHI DAILI	16	IDP_FIF0 Channel 0
IRPTL/IMASK (high priority option) LIRPTL (low priority option)	11 6	0x2C 0x5C	DAIHI DAILI	17	IDP_FIF0 Channel 1

Configuring IOP/Core Interaction

Table 2-3. Default DMA Interrupt Vector Locations (Cont'd)

Associated Register(s)	Bits	Vector Address	Interrupt Name	DMA Channel	Data Buffer
IRPTL/IMASK (high priority option)	11	0x2C	DAIHI	18	IDP_FIF0 Channel 2
LIRPTL (low priority option)	6	0x5C	DAILI		
IRPTL/IMASK (high priority option)	11	0x2C	DAIHI	19	IDP_FIF0 Channel 3
LIRPTL (low priority option)	6	0x5C	DAILI		
IRPTL/IMASK (high priority option)	11	0x2C	DAIHI	20	IDP_FIF0 Channel 4
LIRPTL (low priority option)	6	0x5C	DAILI		
IRPTL/IMASK (high priority option)	11	0x2C	DAIHI	21	IDP_FIF0 Channel 5
LIRPTL (low priority option)	6	0x5C	DAILI		
IRPTL/IMASK (high priority option)	11	0x2C	DAIHI	22	IDP_FIF0 Channel 6
LIRPTL (low priority option)	6	0x5C	DAILI		
IRPTL/IMASK (high priority option)	11	0x2C	DAIHI	23	IDP_FIF0 Channel 7
LIRPTL (low priority option)	6	0x5C	DAILI		
IRPTL/IMASK (high priority option)	12	0x30	SPIHI	24	RXSPI, TXSPI
LIRPTL (low priority option)	9	0x74	SPILI	25	RXSPIB, TXSPIB
IRPTL/IMASK	18	0x68	MTMI	26	MTM Read FIFO
IRPTL/IMASK			UART0RXI	27	RBR0
IRPTL/IMASK			UART1RXI	28	RBR1
IRPTL/IMASK			UART0TXI	29	THR0

Table 2-3. Default DMA Interrupt Vector Locations (Cont'd)

Associated Register(s)	Bits	Vector Address	Interrupt Name	DMA Channel	Data Buffer
IRPTL/IMASK			UART1TXI	30	THR1
LIRPTL		0x50	EPDMA	31	EPDF0
LIRPTL			EPDMA	32	EPTF0
LIRPTL		0x50	EPDMA	33	EPDF1
IRPTL/IMASK			MTMI	34	MTM Write FIFO

For more information, see the program sequencer “Interrupts and Sequencing” section of Chapter 3 in the *ADSP-2136x SHARC Processor Programming Reference* and [Appendix B, Interrupts](#).

Interrupt Latency in Interrupt-Driven Transfers

During an interrupt-driven I/O transfer from any peripheral that uses an IOP interrupt service routine, a write into an IOP register to clear the interrupt causes a certain amount of latency. If the program comes out of the interrupt service routine during that period of latency, the interrupt is generated again.

To avoid the interrupt from being regenerated, use one of the following solutions.

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

```
isr_code:
    R0 = 0x0;
    dm(SPICTL) = R0; /* disable SPI */
    R0 = dm(SPICTL); /* dummy read, occurs only after
                     write */
    rti;
```

Configuring IOP/Core Interaction

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

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

3. Read a status register from the same peripheral block to check whether the interrupt has cleared.

Polling/Status-Driven I/O

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

- All the bits in the IRPTL and LIRPTL registers are shown in [“Interrupt Latch Register \(IRPTL\)” on page B-13](#) and [“Interrupt Register \(LIRPTL\)” on page B-6](#).
- [Figure A-44 on page A-114](#) lists all the bits in the DAI_IRPTL_H and DAI_IRPTL_L registers.

The DMA controller in the ADSP-21367/8/9 and ADSP-2137x processors maintains the status information of the channels in each of the peripherals registers, SPMCTLx, EPDMACCTL, DAI_STAT, DPI_PIN_STAT, RXSTAT_UACx, TXSTAT_UACx and SPIDMAC. More information on these registers can be found at the following locations.

- Bit definitions for the `SPIDMAC` register are illustrated in “[SPI Port Status \(SPISTAT, SPISTATB\) Registers](#)” on page A-56.
- Bit definitions for the `SPMCTLx` register are illustrated in “[SPORT Multichannel Control Registers \(SPMCTLx\)](#)” on page A-40.
- Bit definitions for the `DAI_STAT` register are illustrated in [Figure A-41](#) on page A-110.

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

DMA Controller Operation

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

Non-chained DMA. To start a new DMA sequence after the current one is finished, a program must first clear the DMA enable bit, write new parameters to the index, modify, and count registers, then set the DMA enable bit to re-enable DMA.

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



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

Configuring IOP/Core Interaction

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

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

A DMA sequence ends when one of the following occurs:

- The count register decrements to zero, and the chain pointer register is zero.
- Chaining is disabled and the channel's DMA enable bit transitions from high to low. If the DMA enable bit goes low (=0) and chaining is enabled, the channel enters chain insertion mode and the DMA sequence continues. [For more information, see “Inserting a TCB in an Active Chain” on page 2-41.](#)

Once a program starts a DMA process, the process is influenced by two external controls—DMA channel priority and DMA chaining. For more information, see [“Managing DMA Channel Priority” on page 2-19](#) or [“Chaining DMA Processes”](#) below.

Chaining DMA Processes

The location of the DMA parameters for the next sequence comes from the chain pointer register. In chained DMA operations, the processor automatically initializes and then begins another DMA transfer when the current DMA transfer is complete. In addition to the standard DMA parameter registers, each DMA channel (SPORT, eternal port, UART and SPI) also has a chain pointer register that points to the next set of DMA

parameters stored in the processor's internal memory. These are the `CPSPxy` registers for the SPORTs, the `CPEP` register for the external port, the `RXCP_UACx` registers for the UART, and the `CPSPI` register for the SPI. Each new set of parameters is stored in a four-word, user-initialized buffer in internal memory known as a transfer control block (TCB).

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

A DMA sequence is defined as the sum of the DMA transfers for a single channel, from when the parameter registers initialize to when the count register decrements to zero. Each DMA channel has a chaining enable bit (`CHEN`) in the corresponding control register. This bit must be set to one to enable chaining. When chaining is enabled, DMA transfers are initiated by writing a memory address to the chain pointer register. This is also an easy way to start a single DMA sequence, with no subsequent chained DMAs.

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

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




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

The chain pointer register is 20 bits wide. The lower 19 bits are the memory address field. Like other I/O processor address registers, the chain pointer register's value is offset to match the starting address of the

Configuring IOP/Core Interaction

processor's internal memory before it is used by the I/O processor. On the ADSP-21367/8/9 and ADSP-2137x processors, this offset value is 0x0008 0000.

Bit 19 of the chain pointer register is the program-controlled interrupts (PCI) bit. This bit controls whether an interrupt is latched after every DMA in the chain (when set), or whether the interrupt is latched after the entire DMA sequence completes (if cleared).

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

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

The DMA registers are shown in [Figure 2-1](#).

Transfer Control Block Chain Loading (TCB)

During TCB chain loading, the I/O processor loads the DMA channel parameter registers with values retrieved from internal memory. The address in the chain pointer register points to the highest address of the TCB (containing the index parameter). This means that if a program declares an array to hold the TCB, the chain pointer register should not point to the first location of the array.

[Table 2-4](#) shows the TCB-to-register loading sequence for the serial port and SPI port DMA channels. The I/O processor reads each word of the TCB and loads it into the corresponding register. Programs must set up the TCB in memory in the order shown in [Table 2-4](#), placing the index parameter at the address pointed to by the chain pointer register of the

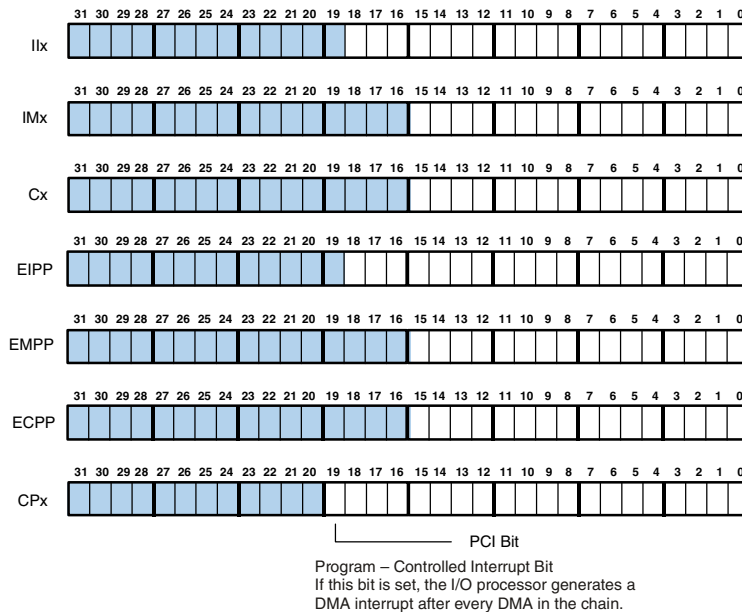


Figure 2-1. DMA Parameter Registers

previous DMA operation of the chain. The end of the chain (no further TCBs are loaded) is indicated by a TCB with a chain pointer register value of zero.

Configuring IOP/Core Interaction

Table 2-4. TCB Chain Loading Sequence¹

Address ²	External Port	Serial Ports	SPI Port
CPSP _x + 0x0008 0000	See Table 2-9 , Table 2-10 , Table 2-11	IISP _x	IISPI
CPSP _x – 1 + 0x0008 0000		IMSP _x	IMSPI
CPSP _x – 2 + 0x0008 0000		CSP _x	CSPI
CPSP _x – 3 + 0x0008 0000		CPSP _x	CPSPI
CPSP _x – 4 + 0x0008 0000			
CPSP _x – 5 + 0x0008 0000			
CPSP _x – 6 + 0x0008 0000			

¹ Chaining is not available using the IDP port.

² An “x” denotes the DMA channel used. While the TCB is eight locations in length, SPI and serial ports only use the first four locations.

A TCB chain load request is prioritized like all other DMA operations. The I/O processor latches a TCB loading request and holds it until the load request has the highest priority. If multiple chaining requests are present, the I/O processor services the TCB registers for the highest priority DMA channel first. A channel that is in the process of chain loading cannot be interrupted by a higher priority channel. For a list of DMA channels in priority order, see [Table 2-7 on page 2-32](#).

Setting Up DMA Channel Allocation and Priorities

There are between 24 and 34 channels of DMA available on the ADSP-21367/8/9 and ADSP-2137x processors, depending on the processor model. The maximum number is configured as—16 via the serial ports, 8 via the input data port, 4 for the UARTs, 2 for the SPI interface, 2 for the external port, and 2 for memory-to-memory transfers. Each channel has a set of parameter registers which are used to set up DMA

transfers. [Table 2-5](#) shows the DMA channel allocation and parameter register assignments for the ADSP-21367/8/9 and ADSP-2137x processors.



DMA channels vary by processor model. For a breakdown of DMA channels for a particular model, see the processor specific data sheet.


Managing DMA Channel Priority

The DMA channel prioritization scheme ranks channel 0 as highest priority and channel 34 as the lowest priority. [Table 2-7 on page 2-32](#) lists the DMA channels in priority order. When a channel becomes the highest priority requester, the I/O processor services the channel's request. In the next clock cycle, the I/O processor starts the DMA transfer.

The I/O data (IOD) bus is 32 bits wide and is the only path that the IOP uses to transfer data between internal memory and the peripherals. When there are two or more peripherals with active DMAs in progress, they may all require data to be moved to or from memory in the same cycle. For example, the input data port may fill its `RXPP` buffer just as a SPORT shifts a word into its `RXn` buffer. To determine which word is transferred first, the DMA channels for each of the processor's I/O ports negotiate channel priority with the I/O processor using an internal DMA request/grant handshake.

Each I/O port has one or more DMA channels, and each channel has a single request and a single grant. When a particular channel needs to read or write data to internal memory, the channel asserts an internal DMA request. The I/O processor prioritizes the request with all other valid DMA requests. When a channel becomes the highest priority requester, the I/O processor asserts the channel's internal DMA grant. In the next clock cycle, the DMA transfer starts. [Figure 2-3 on page 2-30](#) shows the paths for internal DMA requests within the I/O processor.

Configuring IOP/Core Interaction

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

The default DMA channel priority is *fixed prioritization* by DMA channel group (serial ports, TWI, UART, IDP, or SPI port). [Table 2-7 on page 2-32](#) lists the DMA channels in descending order of priority.

For information on programming serial port priority modes, see [Table 5-11 on page 5-74](#).

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

Processor core accesses of I/O processor registers and TCB chain loading (both of which occur after the IOD transfer) are subject to the same prioritization scheme as the DMA channels. Applying this scheme uniformly prevents I/O bus contention, because these accesses are also performed over the internal I/O bus. For more information, see [“Chaining DMA Processes” on page 2-14](#).

DMA Bus Arbitration

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

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

IOP channel arbitration can be set to use either a *fixed* or *rotating* algorithm by setting or clearing bit 7 (DCPR) in the SYSCTL register:

- fixed SYSCTL[7] cleared (0)
- rotating SYSCTL[7] set (1)

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

In the rotating priority scheme, the default priorities at reset are the same as that of the fixed priority. However, the peripheral priority is determined by group, not individually. Peripheral groups are shown in [Table 2-5](#).

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

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

Within a peripheral group the priority is highest for the higher indexed peripheral (see [Table 2-5](#)). For example, in SP01 (which is in group A), SP1 has the highest priority.

Table 2-5. DMA Channel Allocation and Parameter Register Assignments

DMA Channel Number	Data Buffer	Group	IOP Address of Data Buffers	Description
0 (highest priority)	RXSP1A, TXSP1A	A	0xC65, 0xC64	Serial Port 1A Data
1	RXSP1B, TXSP1B	A	0xC67, 0xC66	Serial Port 1B Data

Configuring IOP/Core Interaction

Table 2-5. DMA Channel Allocation and Parameter Register Assignments (Cont'd)

DMA Channel Number	Data Buffer	Group	IOP Address of Data Buffers	Description
2	RXSP0A, TXSP0A	A	0xC61, 0xC60	Serial Port 0A Data
3	RXSP0B, TXSP0B	A	0xC63, 0xC62	Serial Port 0B Data
4	RXSP3A, TXSP3A	B	0x465, 0x464	Serial Port 3A Data
5	RXSP3B, TXSP3B	B	0x467, 0x466	Serial Port 3B Data
6	RXSP2A, TXSP2A	B	0x461, 0x460	Serial Port 2A Data
7	RXSP2B, TXSP2B	B	0x463, 0x462	Serial Port 2B Data
8	RXSP5A, TXSP5A	C	0x865, 0x864	Serial Port 5A Data
9	RXSP5B, TXSP5B	C	0x867, 0x866	Serial Port 5B Data
10	RXSP4A, TXSP4A	C	0x861, 0x860	Serial Port 4A Data
11	RXSP4B, TXSP4B	C	0x863, 0x862	Serial Port 4B Data
12	RXSP7A, TXSP7A	C	0x04865, 0x04864	Serial Port 7A Data
13	RXSP7B, TXSP7B	C	0x04867, 0x04866	Serial Port 7B Data
14	RXSP6A, TXSP6A	C	0x04861 or 0x04860	Serial Port 6A Data
15	RXSP6B, TXSP6B	C	0x04863 or 0x04862	Serial Port 6B Data
16	IDP_FIF0	D	0x24D0	DAI IDP Channel 0
17	IDP_FIF0	D	0x24D0	DAI IDP Channel 1
18	IDP_FIF0	D	0x24D0	DAI IDP Channel 2
19	IDP_FIF0	D	0x24D0	DAI IDP Channel 3
20	IDP_FIF0	D	0x24D0	DAI IDP Channel 4
21	IDP_FIF0	D	0x24D0	DAI IDP Channel 5

Table 2-5. DMA Channel Allocation and Parameter Register Assignments (Cont'd)

DMA Channel Number	Data Buffer	Group	IOP Address of Data Buffers	Description
22	IDP_FIF0	D	0x24D0	DAI IDP Channel 6
23	IDP_FIF0	D	0x24D0	DAI IDP Channel 7
24	RXSPI, TXSPI	E	0x1004, 0x1003	SPI Data
25	RXSPIB, TXSPIB	G	0x2804, 0x2803	SPI Data
26	MTM Read FIFO	G	Not accessible	Memory-to-Memory Read Data
27	THR0	G	0x3C00	UART0 Transmit Holding Register
28	RBR0	G	0x3C00	UART0 Receive Buffer Register
29	THR1	G	0x4000	UART1 Transmit Holding Register
30	RBR1	G	0x4000	UART1 Receive Buffer Register
31	EPDF0	G	0x182C	External Port Channel0 Data FIFO
32	EPTF0	G	0x182D	External Port Channel0 Tap List FIFO
33	EPDF1	G	0x183C	External Port Channel1 Data FIFO
34	EPTF1	G	0x183D	External Port Channel1 Tap List FIFO
35 (lowest priority)	MTM Write FIFO	G	Not accessible	Memory-to Memory Write Data

Setting Up DMA Parameter Registers

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

DMA Transfer Direction

DMA transfers between internal memory and external memory devices use the processor's external port. For these types of transfers, a program provides the DMA controller with the internal memory buffer size, address, and address modifier, as well as the external memory buffer size, address and address modifier and the direction of transfer. After setup, the DMA transfers begin when the program enables the channel and continues until the I/O processor transfers the entire buffer to processor memory. [Table 2-6 on page 2-29](#) shows the parameter registers for each DMA channel.

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

Additionally, the ADSP-21367/8/9 and ADSP-2137x processors can use DMA to transfer 64-bit blocks of data between internal memory locations.

The direction (receive or transmit) of the peripheral determines the direction of data transfer. When the port receives data, the I/O processor automatically transfers the data to internal memory. When the port needs to transmit a word, the I/O processor automatically fetches the data from internal memory. [Figure 2-2](#) shows the processor's I/O processor, related ports, and buses. [Figure 2-3 on page 2-30](#) shows more detail on DMA channel data paths.

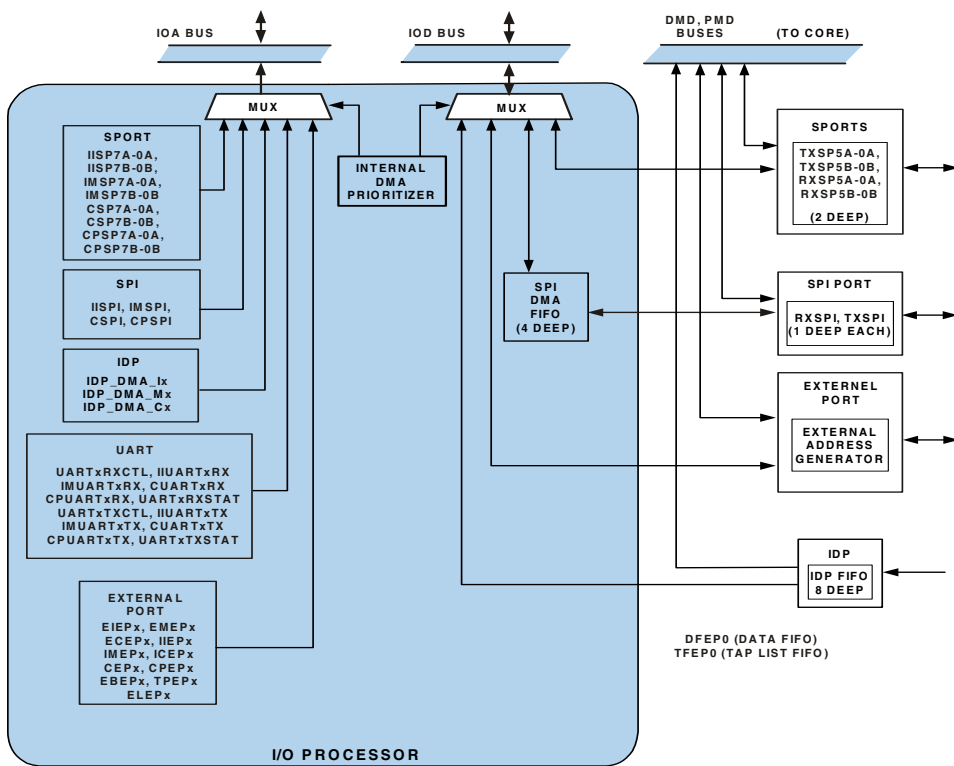


Figure 2-2. I/O Processor Block Diagram

Data Buffer Registers

Figure 2-2 shows the data buffer registers for each port. These registers include:

- **Serial port receive buffers (RXSPx).** These receive buffers for the serial ports have two-position FIFOs for receiving data when connected to another serial device.

Setting Up DMA Parameter Registers

- **Serial port transmit buffers** (TXSPx). These transmit buffers for the serial ports have two-position FIFOs for transmitting data when connected to another serial device.
- **SPI receive buffers** (RXSPI, RXSPIB). These receive buffers for the SPI ports have a single-position buffer for receiving data when connected to another serial device.
- **SPI transmit buffers** (TXSPI, TXSPIB). These transmit buffers for the SPI ports have a single-position buffer for transmitting data when connected to another serial device.
- **Input data port buffers** (IDP_FIFO). This receive buffer for the input data port has eight-position buffers for receiving data when connected to another device.

Port, Buffer, and DMA Control Registers

The port, buffer, and DMA control registers in [Figure 2-2](#) shows the control registers for the ports and DMA channels. These registers include:

- **External port control registers** (DMACx). These are the control registers for the external port DMA channels.
- **Input data port control register** (IDP_CTL). This is the control register for the input data ports.
- **Serial port control registers** (SPCTLx, SPMCTLx). These control registers select the receive or transmit format, monitor FIFO status, enable chaining, and start DMA for each serial port.
- **SPI port control registers** (SPICTL, SPICTLB). These control registers configure and enable the two SPI interfaces, selecting the devices as masters or slaves, and determine the data transfer and word size. The SPIDMAC and SPIDMACB registers also control SPI DMA and FIFO status.

- **Universal asynchronous receiver/transmitter registers** (RXCTL_UACx, TXCTL_UACx). These control registers configure and enable the UART receiver and transmitter DMA, (chaining and non chaining).
- **Memory-to-memory DMA control register** (MTMCTL). This control register contains the MTM DMA read and write channel enable and status bits.

Table 2-6 shows the parameter registers for each DMA channel. These registers function similarly to data address generator registers and include:

- **Internal index registers** (IISPx, IISPI, IISPIB, IIEP, IDP_DMA_Ix, RXI_UAC/TXI_UAC). Index registers provide an internal memory address, acting as a pointer to the next internal memory DMA read or write location.
- **Internal modify registers** (IMSPx, IMEP, IMSPI, IMSPIB, IDP_DMA_Mx, RXM_UAC/TXM_UAC). Modify registers provide the signed increment by which the DMA controller post-modifies the corresponding internal memory index register after the DMA read or write.
- **Count registers** (CSPx, ICEP, CSPI, CSPIB, IDP_DMA_Cx, RXC_UAC/TXC_UAC). Count registers indicate the number of words remaining to be transferred to or from internal memory on the corresponding DMA channel.
- **Chain pointer registers** (CPSPx, CPSPI, CPSPIB, CPEP, RXCP_UAC/TXCP_UAC). Chain pointer registers hold the starting address of the TCB (parameter register values) for the next DMA operation on the corresponding channel. These registers also control whether the I/O processor generates an interrupt when the current DMA process ends.
- **External index registers** (EIEPx). Index registers provide an external memory address, acting as a pointer to the next external memory DMA read or write location.

Setting Up DMA Parameter Registers

- **External modify registers** ($EMEP_x$). Modify registers provide the increment by which the DMA controller post-modifies the corresponding external memory index register after the DMA read or write.
- **External count registers** ($ECEP_x$). External count registers indicate the number of words remaining to be transferred to or from external memory on the corresponding DMA channel.
- **Memory-to-memory write index register** ($IIMTMW$). This register provides the base address in memory where DMA writes start.
- **Memory-to-memory write modify register** ($IMMTMW$). The MTM modify register modifies the write index register after each 32-bit write.
- **Memory-to-memory write counter register** ($CMTMW$). The MTM counter register indicates the quantity of 32-bit data to be transferred to memory. The counter is decremented by one after each data write.
- **Memory-to-memory read index register** ($IIMTMR$). This register provides the base address in memory where DMA reads start.
- **Memory-to-memory read modify register** ($IMMTMR$). The MTM modify register modifies the write index register after each 32-bit read.
- **Memory-to-memory read counter register** ($CMTMR$). The MTM counter register indicates the quantity of 32-bit data to be read from memory. The counter is decremented by one after each data write.

Table 2-6. DMA Parameter Registers

Register	Function	Width	Description
Ily	Internal Index Register	19 bits	Address of buffer in internal memory
IMxy	Internal Modify Register	16 bits ¹	Stride for internal buffer
Cxy	Internal Count Register	16 bits	Length of internal buffer
CPxy	Chain Pointer Register	20 bits	Chain pointer for DMA chaining
EIEP	External Index Register	19 bits	Address of buffer in external memory
EMEP	External Modify Register	16 bits	Stride for external buffer
ECEP	External Count Register	16 bits	Length of external buffer

1 IDP_DMA_Mx registers are 6 bits wide only.

Addressing

Figure 2-3 shows a block diagram of the I/O processor's address generator (DMA controller). Table 2-6 lists the parameter registers for each DMA channel. The parameter registers are uninitialized following a processor reset.

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

Setting Up DMA Parameter Registers

All addresses in the index registers are offset by a value that matches the processor's first internal normal word addressed RAM location (before the I/O processor uses the addresses). For the ADSP-21367/8/9 and ADSP-2137x processors, this offset value is 0x0008 0000.

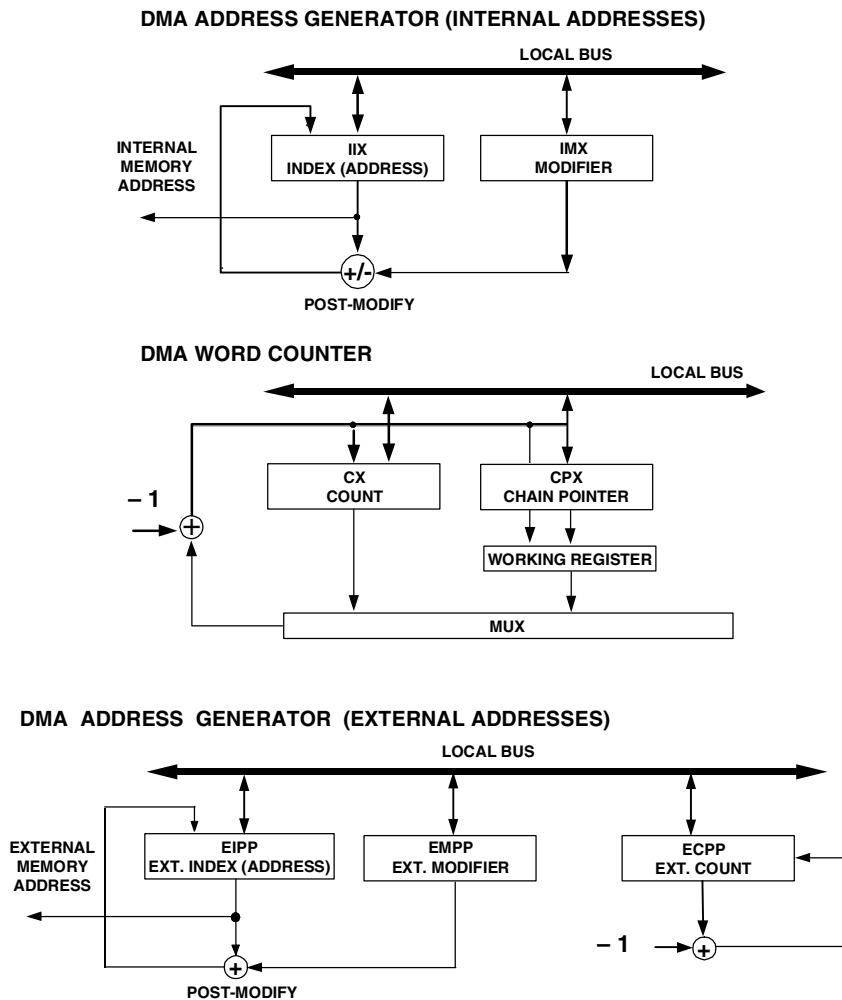



Figure 2-3. DMA Address Generator

The following rules for data transfers must be followed.

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

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

- If the I/O processor modifies the index register past the maximum 18-bit value to indicate an address out of internal memory, the index wraps around to zero. With the offset for the ADSP-2136x SHARC processor processors, the wraparound address is 0x0008 0000.
 - If a DMA channel is disabled, the I/O processor does not service requests for that channel, (whether or not the channel has data to transfer).
-  If a program loads the count register with zero, the I/O processor does not disable DMA transfers on that channel. The I/O processor interprets the zero as a request for 2^{16} transfers. This count occurs because the I/O processor starts the first transfer before testing the count value. The only way to disable a DMA channel is to clear its DMA enable bit.

The processor's 34 DMA channels are numbered as shown in [Table 2-7](#). This table also shows the control, parameter, and data buffer registers that correspond to each channel.

Setting Up DMA Parameter Registers

In the serial port pair SP0/1, SP1 has a higher priority. For multichannel pairs, the odd numbered channels have a higher priority (for example SP3, SP5).

Table 2-7. DMA Channel Registers: Controls, Parameters, and Buffers

DMA Channel Number	Control Registers	Parameter Registers	Buffer Registers	Description
0	SPCTL1	IISP1A, IMSP1A, CSP1A, CPSP1A	RXSP1A, TXSP1A	Serial Port 1A Data
1	SPCTL1	IISP1B, IMSP1B, CSP1B, CPSP1B	RXSP1B, TXSP1B	Serial Port 1B Data
2	SPCTL0	IISP0A, IMSP0A, CSP0A, CPSP0A	RXSP0A, TXSP0A	Serial Port 0A Data
3	SPCTL0	IISP0B, IMSP0B, CSP0B, CPSP0B	RXSP0B, TXSP0B	Serial Port 0B Data
4	SPCTL3	IISP3A, IMSP3A, CSP3A, CPSP3A	RXSP3A, TXSP3A	Serial Port 3A Data
5	SPCTL3	IISP3B, IMSP3B, CSP3B, CPSP3B	RXSP3B, TXSP3B	Serial Port 3B Data
6	SPCTL2	IISP2A, IMSP2A, CSP2A, CPSP2A	RXSP2A, TXSP2A	Serial Port 2A Data
7	SPCTL2	IISP2B, IMSP2B, CSP2B, CPSP2B	RXSP2B, TXSP2B	Serial Port 2B Data
8	SPCTL5	IISP5A, IMSP5A, CSP5A, CPSP5A	RXSP5A, TXSP5A	Serial Port 5A Data
9	SPCTL5	IISP5B, IMSP5B, CSP5B, CPSP5B	RXSP5B, TXSP5B	Serial Port 5B Data
10	SPCTL4	IISP4A, IMSP4A, CSP4A, CPSP4A	RXSP4A, TXSP4A	Serial Port 4A Data
11	SPCTL4	IISP4B, IMSP4B, CSP4B, CPSP4B	RXSP4B, TXSP4B	Serial Port 4B Data

Table 2-7. DMA Channel Registers: Controls, Parameters, and Buffers (Cont'd)

DMA Channel Number	Control Registers	Parameter Registers	Buffer Registers	Description
12	SPCTL7	IISP7A, IM7P5A, CSP7A, CPSP7A	RXSP7A, TXSP7A	Serial Port 7A Data
13	SPCTL7	IISP7B, IMSP7B, CSP7B, CPSP7B	RXSP7B, TXSP7B	Serial Port 7B Data
14	SPCTL6	IISP6A, IMSP6A, CSP6A, CPSP6A	RXSP6A, TXSP6A	Serial Port 6A Data
15	SPCTL6	IISP6B, IMSP6B, CSP6B, CPSP6B	RXSP6B, TXSP6B	Serial Port 6B Data
16	IDP_CTL	IDP_DMA_I0, IDP_DMA_M0, IDP_DMA_C0	IDP_FIFO	DAI IDP Channel 0
17	IDP_CTL	IDP_DMA_I1, IDP_DMA_M1, IDP_DMA_C1	IDP_FIFO	DAI IDP Channel 1
18	IDP_CTL	IDP_DMA_I2, IDP_DMA_M2, IDP_DMA_C2	IDP_FIFO	DAI IDP Channel 2
19	IDP_CTL	IDP_DMA_I3, IDP_DMA_M3, IDP_DMA_C3	IDP_FIFO	DAI IDP Channel 3
20	IDP_CTL	IDP_DMA_I4, IDP_DMA_M4, IDP_DMA_C4	IDP_FIFO	DAI IDP Channel 4
21	IDP_CTL	IDP_DMA_I5, IDP_DMA_M5, IDP_DMA_C5	IDP_FIFO	DAI IDP Channel 5
22	IDP_CTL	IDP_DMA_I6, IDP_DMA_M6, IDP_DMA_C6	IDP_FIFO	DAI IDP Channel 6

Setting Up DMA Parameter Registers

Table 2-7. DMA Channel Registers: Controls, Parameters, and Buffers (Cont'd)

DMA Channel Number	Control Registers	Parameter Registers	Buffer Registers	Description
23	IDP_CTL	IDP_DMA_I7, IDP_DMA_M7, IDP_DMA_C7	IDP_FIFO	DAI IDP Channel 7
24	SPICTL	IISPI, IMSPI, CSPI, CPSPi	RXSPI, TXSPI	SPI Data
25	SPICTLB	IISPIB, IMSPIB, CSPIB, CPSPiB	RXSPIB, TXSPIB	SPI B Data
26	MTMCTL	IIMTMW IMMTMW, CMTMW	N/A	MTM Write Channel
27	MTMCTL	IIMTMR, IMMTMR, CMTMR	N/A	MTM Read Channel
28	AMICTL	EIEP0, EMEP0, ECEP0, IIEP0, IMEP0, ICEP0, CEP0, CPEP0, EBEP0, TPEP0, ELEP0	DFEP0, TFEP0	External Port Channel 0
29	AMICTL	EIEP1, EMEP1, ECEP1, IIEP1, IMEP1, ICEP1, CEP1, CPEP1, EBEP1, TPEP1, ELEP1	DFEP1, TFEP1	External Port Channel 1
30	RXCTL_UAC0	RXI_UAC0, RXM_UAC0, RXC_UAC0, RXCP_UAC0, RXSTAT_UAC0	RBR0, RBRSH_UAC0	UART0 Rx
31	RXCTL_UAC1	RXI_UAC1, RXM_UAC1, RXC_UAC1, RXCP_UAC1, RXSTAT_UAC1	RBR1, RBRSH_UAC1	UART1 Rx

Table 2-7. DMA Channel Registers: Controls, Parameters, and Buffers (Cont'd)

DMA Channel Number	Control Registers	Parameter Registers	Buffer Registers	Description
32	TXCTL_UAC0	TXI_UAC0, TXM_UAC0, TXC_UAC0, TXCP_UAC0, TXSTAT_UAC0	THR0	UART0 Tx
33	TXCTL_UAC1	TXI_UAC1, TXM_UAC1, TXC_UAC1, TXCP_UAC1, TXSTAT_UAC1	THR1	UART1 Tx

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

External Port DMA

The external port has two DMA channels that can use either the SDRAM controller (SDC) or the asynchronous memory interface (AMI). The DMA chooses the correct interface (AMI or SDC) based on the external address as determined by bits 0–3 in the external port global control register (EPCTL, [Table A-3 on page A-11](#)). The DMA controllers support conventional DMA, chained and circular DMA, and delay line DMA. The priority of the two DMA channels is fixed with external port 0 having priority over external port 1.

The DMA controllers have two FIFOs, a four deep data FIFO for received/transmitted data and a four deep tap list FIFO for the tap list entries for the delay line DMA.

The registers that control external port DMA are described in [Table 2-8](#).

Table 2-8. External Port Registers

Register	Description	Address
EPCTL	External Port Global Control Register	0x1801
DMAC1–0	External Port DMA Control Register	0x180B, 0x180C
IIEP1–0	Internal Index Register	0x1823, 0x1833
IMEP1–0	Internal Modifier Register	0x1824, 0x1834
ICEP1–0	Internal Count Register	0x1825, 0x1835
EIEP1–0	External Index Register	0x1820, 0x1830
EMEP1–0	External Modifier Register	0x1821, 0x1831
CPEP1–0	Chain Pointer Register	0x1826, 0x1836
TPEP1–0	Tap List Pointer Register	0x1828, 0x1838
ELEP1–0	Circular Buffer Length Register	0x1829, 0x1839
EBEP1–0	External Base Address Register	0x1827, 0x1837

Setting Up and Starting Chained DMA

Use the following procedure to set up and run a chained DMA on the external port.

1. Configure the `AMICTLx` registers to enable the AMI, set the desired wait states, set the data bus width, and so on. Configure the `SDCTL` register to enable the SDRAM, set the desired clock and timing settings, set the data bus width, and so on.
2. Initialize the `CPEP` register—set the `PCI` bit if interrupts are needed after the end of each DMA block.

3. If circular buffering is needed, then program additional writes to the `ELEP` and `EBEP` registers. Note that for normal chained DMA, the `ELEP` and `EBEP` registers are not part of the TCB. So if circular buffering is used with the normal chained DMA, all the DMA blocks will have same `ELEP` and `EBEP` values.
4. Enable DMA (`DEN`), chaining (`CHEN`), and circular buffering (`CBEN`) if needed, in the `DMACx` registers. It is advised that the DMA FIFOs are flushed (`DFLSH`) when DMA is enabled.

Once the DMA control register is initialized, the DMA controller fetches the DMA descriptors from the address pointed to by the external port chain pointer register (`CPEP`). The order the descriptors are fetched is shown in [Table 2-9](#).

Table 2-9. Chain Pointer Loading Sequence (Normal DMA)

Address	Register Value
<code>CPEP[18:0]</code>	<code>IIEP</code>
<code>CPEP[18:0] – 0x1</code>	<code>IMEP</code>
<code>CPEP[18:0] – 0x2</code>	<code>ICEP</code>
<code>CPEP[18:0] – 0x3</code>	<code>EIEP</code>
<code>CPEP[18:0] – 0x4</code>	<code>EMEP</code>
<code>CPEP[18:0] – 0x5</code>	<code>CPEP</code>

The order the descriptors are fetched with circular buffering enabled is shown in [Table 2-10](#)

External Port DMA

Table 2-10. Chain Pointer Loading Sequence (Circular Buffering Enabled)

Address	Register Value
ELEP[18:0]	IIEP
CPEP[18:0] – 0x1	IMEP
CPEP[18:0] – 0x2	ICEP
CPEP[18:0] – 0x3	EIEP
CPEP[18:0] – 0x4	EMEP
CPEP[18:0] – 0x5	CPEP
EPCP[18:0] – 0x5	EPEB
EPCP[18:0] – 0x6	EPEL

Once the DMA descriptors are fetched, the normal DMA process starts. Upon completion, new DMA descriptors are loaded and the process is repeated until `CPEP = 0x00000`. A DMA completion interrupt is generated at the end of each DMA block or at the end of an entire chained DMA, depending on the `PCI` bit setting.

Delay Line DMA

Delay line DMA is used to support reads and writes to external delay line buffers with limited core interaction. In this sense, delay line DMA is basically a quantity of integrated writes followed by reads from external memory—called a *delay line DMA access*. The delay line DMA access consists of the following accesses in the order listed.

1. Writes to external memory. The number of writes are determined by the external port internal count `ICEP` register. The data is fetched from the external port internal index register (`IIEP`) and the external port internal modify register (`IMEP`) is used as the internal modifier. The external port external index register (`EIEP`) serves

as the external index and is incremented by the external modifier register (EMEP) after each write. These writes are circular buffered if circular buffering is enabled.

2. In chained DMA, when the writes are complete, (ICEP = zero) the EPEI register, which serves as the write pointer of the delay line, is written back to the internal memory location from where it was fetched.
3. Reads from external memory. For reads, the tap list (TL) modifiers are used and the number of reads is determined by the external port read count register (RCEP). The write pointer in the external port external index register (EIEP) serves as the index address for these reads (reads start from where writes end). The EIEP register, along with tap list modifiers, are used in a pre-modify addressing mode to create the external address for the writes. For each 32-bit read the external index is:
 - $EIEP - TL[0]$ is the first read address (where $TL[0]$ is the first tap list entry in internal memory as pointed to by the external port tap list pointer register TPEP).
 - $EIEP - TL[1]$ is the second address, and so on.

Therefore, for each read, the DMA controller fetches the external modifier from the tap list and the reads are circular buffered (if enabled).



The external address generation follows pre-modify addressing for reads in delay line DMA and therefore EIEP values are not modified. Also the EMEP register does not have any effect during delay line reads.

Serial Port DMA

4. Once the read count completes, the delay line DMA access is complete and the DMA complete interrupt is generated. Note that if chaining is enabled, the interrupt is generated based on the `PCI` bit setting. For more information on the `PCI` bit, see [“Interrupt-Driven I/O” on page 2-6](#).

Table 2-11. Chain Pointer Loading Sequence (Delay Line DMA)

Address	Register Value
EPCP[18:0]	EPII (Write Index)
EPCP[18:0] – 0x1	EPIM
EPCP[18:0] – 0x2	EPIC (Write Count)
EPCP[18:0] – 0x3	EPEI
EPCP[18:0] – 0x4	EPEM
EPCP[18:0] – 0x5	EPEB
EPCP[18:0] – 0x6	EPEL
EPCP[18:0] – 0x7	EPRI
EPCP[18:0] – 0x8	EPRC
EPCP[18:0] – 0x9	EPTP
EPCP[18:0] – 0xA	EPCP

Serial Port DMA

The serial ports support standard as well as chained DMA.

Setting Up and Starting Chained DMA

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

1. Set up all TCBs in internal memory.

2. Write to the appropriate DMA control register, setting the DMA enable bit to one and the chaining enable bit to one.
3. Write the address containing the index register value of the first TCB to the chain pointer register, which starts the chain.

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

Inserting a TCB in an Active Chain

It is possible to insert a single DMA operation or another DMA chain within an active DMA chain. Programs may need to perform insertion when a high priority DMA requires service and cannot wait for the current chain to finish.

When DMA on a channel is disabled and chaining on the channel is enabled, the DMA channel is in chain insertion mode. This mode lets a program insert a new DMA or DMA chain within the current chain without effecting the current DMA transfer. Use the following sequence to insert a DMA subchain for the serial port 0A channel while another chain is active:

1. Enter chain insertion mode by setting `SCHEN_A = 1` and `SDEN_A = 0` in the channel's DMA control register, `SPCTL0`. The DMA interrupt indicates when the current DMA sequence is complete.
2. Copy the address currently held in the chain pointer register to the chain pointer position of the last TCB in the chain that is being inserted.
3. Write the start address of the first TCB of the new chain into the chain pointer register.
4. Resume chained DMA mode by setting `SCHEN_A = 1` and `SDEN_A = 1`.

Serial Peripheral Interface DMA

Chain insertion mode operates the same as non-chained DMA mode. When the current DMA transfer ends, an interrupt request occurs and no TCBs are loaded. This interrupt request is independent of the `PCI` bit state.

Chain insertion should not be set up as an initial mode of operation. This mode should only be used to insert one or more TCBs into an active DMA chaining sequence.

Serial Peripheral Interface DMA

The serial peripheral interface supports both standard and chained DMA. However, unlike the serial ports, programs cannot insert a TCB in an active chain using the SPI.


Setting Up and Starting Chained DMA over the SPI

Configuring and starting chained DMA transfers over the SPI port is the same as for the serial port, with one exception. Contrary to SPORT DMA chaining, (where the first DMA in the chain is configured by the first TCB), for SPI DMA chaining, the first DMA is not initialized by a TCB. Instead, the first DMA in the chain must be loaded into the SPI parameter registers (`IISPI`, `IMSPI`, `CSPI`), and the chain pointer register (`CPSPI`) points to a TCB that describes the second DMA in the sequence.

[Table 2-12](#) shows the order of register loading.

Table 2-12. DMA Chaining Sequence

Address	Register	Description
CPSPI	DMA Start Address	Address in memory
CPSPI – 1	DMA Address Modifier	Address increment
CPSPI – 2	DMA Word Count	Number of words to transfer
CPSPI – 3	DMA Next TCB	Pointer to address of next TCB

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

The sequence for setting up and starting a chained DMA is outlined in the following steps and can also be seen in [Listing 6-3 on page 6-43](#).

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

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

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

UART DMA

In the UART, separate receive and transmit DMA channels move data between the UART and memory. The software does not have to move data, it just has to set up the appropriate transfers either through the descriptor mechanism or through auto buffer mode. See also “[DMA Controller Operation](#)” on page 2-13.

To perform DMA transfers, the UART has a special set of receive and transmit registers. These registers are listed in [Table 2-14](#).

Table 2-13. UART DMA Registers

Register	Description
UARTxRXCTL (3 bits)	DMA Config/Control register for UART Rx
IUARTxRX (19 bits)	Address for DMA
IMUARTxRX (16 bits)	Modifier
CUARTxRX (16 bits)	Count
CPUARTxRX (20 bits)	Chain Pointer
UARTxRXSTAT (3 bits)	DMA Status register
UARTxTXCTL (3 bits)	DMA Config/Control register for UART Tx
IUARTxTX (19 bits)	Address for DMA
IMUARTxTX (16 bits)	Modifier
CUARTxTX (16 bits)	Count
CPUARTxTX (20 bits)	Chain Pointer
UARTxTXSTAT (3 bits)	DMA Status register

No additional buffering is provided in the UART DMA channel, so the latency requirements are the same as in non-DMA mode. However, the latency is determined by the bus activity and arbitration mechanism and not by the processor loading and interrupt priorities.

DMA through the UART is started by setting up values in the DMA parameter registers and then writing to the transmit and receive control registers, enabling the module using the `UARTEN` bits (in the `UARTxTXCTL` and `UARTxRXCTL` registers) and enabling DMA using the `UARTDEN` bits. A DMA can be interrupted by resetting the `UARTDEN` bit in the control register. A DMA request that is already in the pipeline completes normally.

DMA chaining is enabled by setting the `UARTCHEN` bit in the transmit and receive control registers. When chaining is enabled at the end of a current DMA, the next set of DMA parameters are loaded from internal memory and a new DMA starts. The index of the memory location is written in the chain pointer register. DMA parameter values reside in consecutive memory locations as shown in [Table 2-14](#). Chaining ends when the `CP` register contains address `0x00000` for the next parameter block.

Table 2-14. Transfer Control Block Chain Loading Sequence

Address	Register Value
<code>CPUARTxTX/RX18-0</code>	<code>IIUARTxTX/RX</code>
<code>CPUARTxTX/RX18-0 - 0x1</code>	<code>IMUARTxTX/RX</code>
<code>CPUARTxTX/RX18-0 - 0x2</code>	<code>CUARTxTX/RX</code>
<code>CPUARTxTX/RX18-0 - 0x3</code>	<code>CPUARTxTX/RX</code>

To start a DMA use the following steps.

1. Initialize index, modify, and count registers with the appropriate values, keeping `DEN` and `EN` bits disabled in the `UARTxTX/RXCTL` register.
2. Write to the control register with the required control values and set the `DEN`, `EN` and `CHEN` (if chaining is needed) bits.

UART DMA

When performing DMA using the UART module, receive interrupts are generated when:

- The receive word block is complete/the DMA is complete.
- A receive overrun error is detected.
- A receive parity error is detected.
- A receive framing error is detected.
- An address detect interrupt (for 9 bit mode) is detected.

The transmit interrupt is generated when the transmit word block is complete/the DMA is complete.

To start a chain pointer DMA use the following steps.

1. Initialize the chain pointer register with the address of the DMA descriptor table. Set the `PCI` bit if an interrupt is needed at the end of each DMA block.
2. Set up the appropriate control register to enable the UART transmitter and receiver, chain pointer, and DMA. Once chain pointer DMA is enabled, the DMA engine fetches the index, modify, count, and chain pointer values from the memory address specified in the chain pointer register. Once the DMA descriptors are fetched, normal DMA starts. This process is continued until the chain pointer register contains all zeros.

Notes On Using DMA With the UART

The following should be noted when performing DMA in conjunction with the UART module.

1. DMA can be interrupted by resetting the `DEN` bit, but none of the other control settings should be changed. If the UART is enabled again, then interrupted DMA can be resumed by resetting the `DEN` bit.
2. Disabling the UART by resetting the enable (`EN`) bit flushes data in the transmit/receive buffer. Resetting the UART during a DMA operation is prohibited and leads to data loss.
3. Do not disable chaining (`CHEN` bit in the control registers) when a chaining DMA is in progress. If this occurs, a DMA completion interrupt will not be generated when the `PCI` bit = 1.
4. During a receive DMA, a read of the receiver buffer (`UARTxRBR`) is not allowed. If needed, programs should read the receiver shadow buffer (`UARTxRBRSH`).
5. During DMA, the `UARTDR` bit in the `UARTxLSR` register is cleared automatically.
6. DMA may be used in 9-bit mode, once the address has been detected. If, between DMAs, another address is received, an address detect interrupt is generated (if enabled). At this point, the `UARTxRBRSH` shadow register can be read to find the 9-bit word (the address). The `UARTxLSR` register also shows the `UARTRX9D` (address detect) bit.

Memory-to-Memory DMA

Memory-to-memory (MTM) DMA allows programs to transfer blocks of 64-bit data from one internal memory location to another. This transfer method uses two DMA channels, one for reading data and one for writing data. This data transfer can be set up using the following procedure.

1. Program the DMA registers for both channels.
2. Set (=1) the `MTMFLUSH` bit in the `MTMCTL` register to flush the FIFO and reset the read/write pointers.
3. Set (=1) the `MTMEN` bit in the `MTMCTL` register and clear (=0) the `MTMFLUSH` bit.

A two-deep, 32-bit FIFO regulates the data transfer through the DMA channels.

If the `MTMI` bit in the `IMASK` register is enabled, an interrupt occurs at the end of each DMA (read/write). [For more information, see “Interrupts” in Appendix B, Interrupts.](#)

Summary

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

The SPI port, external port, serial ports, input data ports and the MTM DMA engine each have a DMA enable bit (`SPIDEN`, `DMAEN`, `SDEN`, `IDP_DMA_EN`, or `MTM_DEN`) in their channel control register. Setting this bit for a DMA channel with configured DMA parameters starts the DMA on

that channel. If the parameters configure the channel to receive, the I/O processor transfers data words received at the buffer to the destination in internal memory. If the parameters configure the channel to transmit, the I/O processor transfers a word automatically from the source memory to the channel's buffer register. These transfers continue until the I/O processor transfers the selected number of words as determined by the count parameter. DMA through the IDP ports occurs in internal memory only.

Programming Example

The following example demonstrates setting up a memory-to-memory direct memory access (DMA).

```
/* Register Definitions */

#define MTMCTL 0x2c01
#define IIMTMW 0x2c10
#define IIMTMR 0x2c11
#define IMMTMW 0x2c0e
#define IMMTMR 0x2c0f
#define CMTMW 0x2c16
#define CMTMR 0x2c17

/* Bit Definitions */

#define MTMI    0x00040000
#define IRPTEN 0x00001000
#define MTMEN   0x1
#define MTMFLUSH 0x2

/* Buffer Declarations */

.section/dm seg_dmda;
.align 2;
.var dest[100];
```

Programming Example

```
.align 2;
.var source[100];

/* Main code section */

.global _main;
.section/pm seg_pmco;
_main:

r0=0x11111111;

i0=source;

/* Fill the source buffer */

lcntr=LENGTH(source), do fill until lce;
    dm(i0,1)=r0;
fill: r0=rot r0 by 1;

/* Set the interrupt mask for MTMDMA */

bit set imask MTMI;
bit set model IRPTEN;

/* Flush the MTMDMA FIFO */

r0=MTMFLUSH;
dm(MTMCTL)=r0;

/* Set up the source address to read */

r0=source;
dm(IIMTMR)=r0;

/* Set up the destination address to write */

r0=dest;
dm(IIMTMW)=r0;
```

```
/* Read and write sequentially with a step of 1 */  
  
r0=1;  
dm(IMMTMW)=r0;  
dm(IMMTMR)=r0;  
  
/* Read the number of words in source */  
  
r0=@source;  
dm(CMTMR)=r0;  
  
/* Write the number of words in destination */  
  
r0=@dest;  
dm(CMTMW)=r0;  
  
/* Enable MTMDMA */  
  
r0=MTMEN;  
dm(MTMCTL)=r0;  
  
_main.end: jump(pc,0);
```

Programming Example

3 EXTERNAL PORT

The external ports of the ADSP-21367/8/9 and ADSP-2137x processors are comprised of the following modules.

- An [“Asynchronous Memory Interface” on page 3-20](#) which communicates with SRAM, FLASH, and other devices that meet the standard asynchronous SRAM access protocol. The AMI supports 16M words of external memory in bank1, bank2, and bank3 and 14M words of external memory in bank0. The maximum external data is 64M bytes on bank1.

Each bank is individually programmed to access a single memory, either SDRAM or asynchronous.

- An SDRAM controller that supports a glueless interface with any of the standard SDRAMs of 32, 64, 128, 256, and 512M bit. See [“SDRAM Controller” on page 3-30](#).
- A [“Shared Memory Interface” on page 3-79](#) that allows the connection of up to four ADSP-21368 processors to create shared external bus systems (ADSP-21368 only).
- A DMA interface between internal and external memory that directs processor core accesses to external memory locations. [For more information, see Chapter 2, I/O Processor.](#)


External Memory Interface

The external memory interface provides a glueless interface to external memories. The processor's I/O processor (IOP) supports synchronous DRAMs (SDRAMs), SRAMs, FIFOs, flash memory, and ASIC/FPGA devices. The external memory interface and the SDRAM memory that interfaces to the external port is clocked by the SDRAM clock (SDCLK). The ratio of core clock (CCLK) to SDCLK is determined by programming bits in the power management control (PMCTL) registers. [For more information, see “Power Management Control Register \(PMCTL\)” in Appendix A, Register Reference.](#)

The various possible SDRAM clock to core clock frequency ratios are 1:2, 1:2.5, 1:3, 1:3.5, and 1:4. This ratio is independent of the peripheral clock (PCLK) used by the other peripherals. For more information on timing, see [Chapter 14, System Design](#) and the appropriate ADSP-21367/8/9 and ADSP-2137x processor data sheet.


The asynchronous external interface follows the standard asynchronous SRAM access protocol. Programmable wait states, idle cycles, and hold cycles are provided to interface memories that have different access times. To extend access, an acknowledge (ACK) signal can be pulled low by the external device.

The external memory interface supports access to the external memory by direct core accesses and DMA accesses. The external memory address space for non-SDRAM addresses is shown in [Table 3-1 on page 3-10](#). The external memory address space for SDRAM addresses is shown in [Table 3-20 on page 3-52](#). The external memory is divided into four banks. Any bank can be programmed as either asynchronous or synchronous memory.

-  External memory address space is supported in normal word addressing mode only. Single-instruction, multiple-data (SIMD), extended-precision, short word and long word addressing modes are not supported. Program execution from external memory on the ADSP-2136x processors is also not supported.

External Memory Interface on the ADSP-2137x Processors

The ADSP-2137x SHARC processors support direct execution of instructions from external memory, using the 16-bit external port (on the ADSP-21375) or the 32-bit external port (as on the ADSP-21371). Execution is supported from external memory bank0 space which is selected by $\overline{MS0}$. This external memory can either be SDRAM, or asynchronous memory, such as SRAM or flash.

-  While 16-bit to 48-bit packing, and 32-bit to 48-bit packing are supported when the external memory is SDRAM, the external asynchronous memory interface (AMI) also supports 8-bit to 48-bit, 16-bit to 48-bit, and 32-bit to 48-bit instruction packing.

Direct Execution of Instructions From External Memory

While the earlier SHARC processor families (including ADSP-2126x and ADSP-2136x) supported data storage in external memory, and core as well as DMA accesses to and from external memory, the ADSP-2137x processors extends this capability to support direct execution of instructions from external memory.

Throughput and Instruction Execution Rate

Since instructions on the SHARC processor are 48 bits wide, instruction throughput when executing code from external SDRAM memory is 2 instructions every 3 $SDCLK$ (peripheral) clock cycles over a 32-bit wide external port, and 2 instructions every 6 $SDCLK$ clock cycles over a 16-bit

External Memory Interface

wide external port. When executing from external asynchronous memory, instruction throughput depends on the settings of asynchronous memory such as the number of wait states, the ratio of core to peripheral clock and other settings. For details, please refer to the external port global control register (EPCTL), asynchronous memory interface control register (AMICTLx), and SDRAM control register (SDCTL) documentation in the *ADSP-2136x SHARC Processor Hardware Reference for the ADSP-21367/8/9 Processors*.

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

The address map for code is same as the data—each address refers to a 32-bit word. Any address produced by the sequencer is checked to determine if it falls in the external memory and if it does, the SDRAM controller initiates access to the SDRAM. Because the sequencer address bus is limited to 24 bits, only part of the external memory address area can be used to store code. As explained in the following section, “[Location of Interrupt Vector Table \(IVT\)](#)”, the address generated by the sequencer undergoes translation to produce physical address, since the SDRAM data bus width is less than 48 bits.

Location of Interrupt Vector Table (IVT)

On ADSP-2137x processors, the interrupt vector table always starts at the top of internal memory at address 0x90000. This is set by programming the IIVT bit in the system control register (SYSCTL). Therefore, if code execution from external memory is desired upon reset, the program needs to set up the appropriate interrupt vector tables in internal memory as part of the boot-up code before beginning to execute instructions from external memory.

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

Instruction Cache

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

In previous generations of SHARC processors, the function of the conflict cache had been to cache only those instructions whose fetching conflicted with access of a data operand from memory over the PM bus. The enhancements to the cache architecture mean that the functionality of the cache remains intact for execution from internal memory whereas it behaves as instruction cache for external memory execution. Every instruction that is fetched from external memory into the program sequencer is also simultaneously loaded into the cache. The next time that this instruction needs to be fetched from external memory, it is first searched for in the cache. The instruction is stored using the entire 24-bit address. [Figure 3-1](#) shows the format for storing an instruction and [Figure 3-2](#) shows the cache architecture.

External Memory Interface

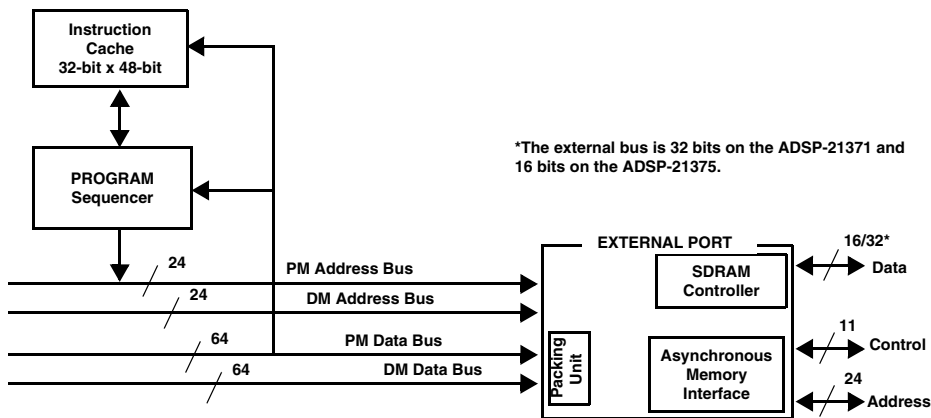


Figure 3-1. Cache Operation During Instruction Execution from External Memory

		LRU BIT	VALID BIT	INSTRUCTIONS	ADDRESSES BITS (23-4)	ADDRESSES BITS (3-0)
SET 0	ENTRY 0	<input type="checkbox"/>	<input type="checkbox"/>			0000
	ENTRY 1		<input type="checkbox"/>			
SET 1	ENTRY 0	<input type="checkbox"/>	<input type="checkbox"/>			0001
	ENTRY 1		<input type="checkbox"/>			
SET 2	ENTRY 0	<input type="checkbox"/>	<input type="checkbox"/>			0010
	ENTRY 1		<input type="checkbox"/>			
<div style="background-color: #e0f0ff; padding: 5px; text-align: center;"> <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;">↕</div> <div style="text-align: center;">↕</div> <div style="text-align: center;">↕</div> <div style="text-align: center;">↕</div> <div style="text-align: center;">↕</div> <div style="text-align: center;">↕</div> <div style="text-align: center;">↕</div> </div> </div>						
SET 13	ENTRY 0	<input type="checkbox"/>	<input type="checkbox"/>			1101
	ENTRY 1		<input type="checkbox"/>			
SET 14	ENTRY 0	<input type="checkbox"/>	<input type="checkbox"/>			1110
	ENTRY 1		<input type="checkbox"/>			
SET 15	ENTRY 0	<input type="checkbox"/>	<input type="checkbox"/>			1111
	ENTRY 1		<input type="checkbox"/>			

Figure 3-2. Instruction Cache Architecture

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

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

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

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

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

External Memory Interface

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

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

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

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

Previously, in the absence of the internal instruction cache, the number of cycles taken by the loop for a case of `FILTER_TAPS = 16` would have been a minimum of 48 cycles over a 16-bit wide external bus, and 24 cycles over a 32-bit wide external bus (excluding any conflicts for data operand fetches). However, with the presence of the instruction cache, and assuming that the execution is from external SDRAM, and that the instructions are on the same SDRAM page, the number of cycles is reduced to 17 over a 16-bit wide external bus, and either 15 cycles or 16 cycles over a 32-bit wide bus (depending on whether instruction 1 begins on an even 32-bit address, or odd 32-bit address).

Thus, the internal cache improves the efficiency of execution from 16-bit wide external memory by approximately 64.5% for this example.

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

Instruction Storage and Packing

The ADSP-2137x processors incorporate a 32-bit SDRAM controller. However, the SDRAM controller supports SDRAMs with data bus widths of 16 as well as 32 bits. The packing logic in the SDRAM controller packs the data from SDRAM into 48-bit instructions. Any address produced by the sequencer which falls in external memory is first translated into the physical address in external memory based on the actual data bus width of external memory as shown in [Figure 3-3](#).

The controller completes the required number of accesses from consecutive locations for returning a 48-bit word instructions. For a 16-bit SDRAM bus, it performs three accesses. For 32-bit SDRAM, three accesses are performed for two instructions. In this packing mode, all the even addresses in external memory are translated by multiplying the address by a factor of 3/2. For example, if A is an even address falling in external memory region ($A > 0x200000$), the translated address is $((A \gg 1) + A)$. For an odd address, the translated address for the previous even address is incremented by 1. Note that it is the absolute address rather than the offset from the base of external memory that is translated. Therefore, the beginning of external memory, 0x200000, is translated to 0x300000. Two 32-bit accesses are performed for each even address. For an odd address in the sequence, only one access is necessary. Two accesses are necessary however, if the odd address happens to be first in a sequence.



Only bank0 can be populated for external code execution on ADSP-2137x processors.

External Memory Interface

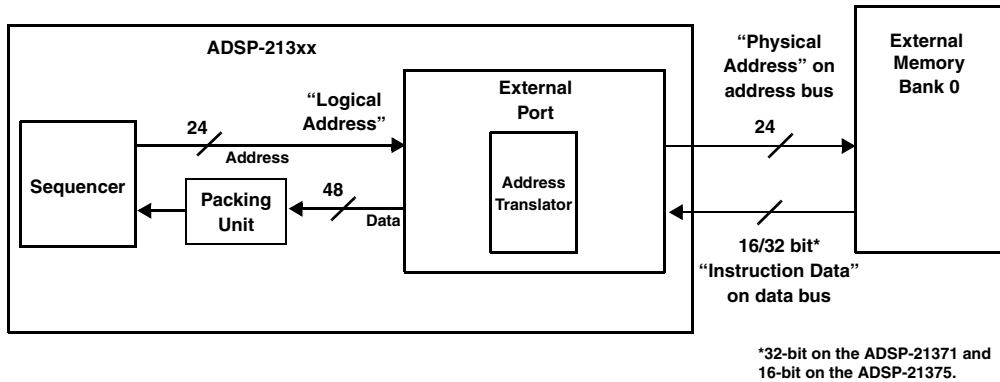


Figure 3-3. Logical Versus Physical Addresses

The address range of external memory (asynchronous memory) is shown in [Table 3-1](#). External bank 0 can be used to execute instructions.

Table 3-1. External Memory Address Space for Non SDRAM Addresses

Bank	Size in words	Address Range
Bank 0	14M	0x0020 0000 – 0x00FF FFFF
Bank 1	16M	0x0400 0000 – 0x04FF FFFF
Bank 2	16M	0x0800 0000 – 0x08FF FFFF
Bank 3	16M	0x0C00 0000 – 0x0CFF FFFF

[Table 3-2](#) shows the addressable range for SDRAM memory space. Bank 0 can be used for executing instructions. Note that the external memory bank addresses shown are for normal word accesses.

The actual throughput execution from external SDRAM is dependent on the configuration of the SDRAM. The SDRAM can be programmed to run at a number of different frequency ratios with respect to the core clock, the fastest being half of the core clock (or the same as the peripheral clock). The core and SDRAM controller have been enhanced so that

Table 3-2. External Address Space for SDRAM Memory Accesses

Bank	Size in words	Address Range
Bank 0	62M	0x0020 0000 – 0x03FF FFFF
Bank 1	64M	0x0400 0000 – 0x07FF FFFF
Bank 2	64M	0x0800 0000 – 0x0BFF FFFF
Bank 3	64M	0x0C00 0000 – 0x0FFF FFFF

throughput is maximized when SDRAM is programmed to run at half the core clock frequency and the instructions being fetched are sequential. After the initial latency, accesses follow a 4-2-4-2-4 core cycles pattern for the above setting for a 32-bit SDRAM meaning that because of the way the instructions are packed in external memory, the first instruction takes four core clock cycles to execute, while the second instruction only takes two core cycles, and so on.

It is possible to store both 48-bit instructions as well as 32-bit data in the external memory bank 0. However, care must be taken while specifying the proper starting addresses if 48-bit instructions are stored or interleaved with 32-bit data in the same memory bank. For example, in case of 32-bit wide external SDRAM memory, two instructions are packed into three 32-bit memory locations, while 32-bit data occupies one memory location each. If 2K instructions are placed starting at the bank 0 base address (0x0020 0000), then the starting address for placing data has to be at least 0x0020 0C00 (in other words, an offset of 3K 32-bit words).

The ADSP-2137x processors support the execution of 48-bit wide program instructions from external memory devices of various widths. The processor can transparently pack and execute instructions stored in 16-bit (ADSP-21375) or 32-bit wide (ADSP-21371) external memory.

[Table 3-3](#) shows the format of stored instructions in external 32-bit wide memory. The sequencer automatically places the normal word address corresponding to the starting address of the first instruction to be fetched from external memory on the appropriate address bus, fetches three 16-bit

External Memory Interface

words and packs them to form the 48-bit instruction to be executed. The address is automatically incremented, and program execution continues with placing the next address on the external address bus, and so on.

In [Table 3-3](#), the logical to physical translation is a multiplication by a factor of 3/2 and $N = 0x8AAAA9$. Therefore, the 32-bit wide memory supports 8.6 million instructions.

Table 3-3. Logical Versus Physical Address Mapping, 32-Bit Asynchronous Memory

Logical Address Dispatched by Program Sequencer	Physical Address Observed on the External Address Bus	Data	
		31	0
0x200000	0x300000	Instr0[31:0]	
	0x300001	Instr1[15:0]	Instr0[47:32]
0x200001	0x300001	Instr1[47:16]	
	0x300002	Instr2[31:0]	
0x200002	0x300002	Instr3[15:0]	Instr2[47:32]
	0x300003	Instr3[47:16]	
...	...		
	...		
...	...		
	...		
0xAAAA9	0xFFFFFE	InstrN[31:0]	
	0xFFFFF	0000	InstrN[47:32]

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

Table 3-4. Logical Versus Physical Address Mapping, 16-Bit Asynchronous Memory

Logical Address Dispatched by Program Sequencer	Physical Address Observed on the External Address Bus	Data	
		16	0
0x200000	0x600000	Instr0[15:0]	
	0x600001	Instr0[31:16]	
	0x600002	Instr0[47:32]	
0x200001	0x600003	Instr1[15:0]	
	0x600004	Instr1[31:16]	
	0x600005	Instr1[47:32]	
0x200002	0x600006	Instr2[15:0]	
	0x600007	Instr2[31:16]	
	0x600008	Instr2[47:32]	
...	...		
...	...		
...	...		
0x555554	0xFFFFFD	InstrN[15:0]	
	0xFFFFFE	InstrN[31:16]	
	0xFFFFF	InstrN[47:32]	

Table 3-5 and Table 3-6 show the logical to physical address translation maps for external SDRAM. In SDRAM, there is an additional 2 bits of address generation available to the SDRAM controller. Therefore, the external addressable range is larger than with asynchronous memory and the entire allowable internal address range of 24 bits can be accessed in external memory.

In Table 3-5, $N = 0xE00000$. Therefore, the total number of external memory instructions for a 32-bit wide SDRAM memory is 14 million.

External Memory Interface

Table 3-5. Logical Versus Physical Address Mapping, 32-Bit SDRAM Memory

Logical Address Dispatched by Program Sequencer	Physical Address Observed on the External Address Bus	Data	
		31	0
0x200000	0x300000	Instr0[31:0]	
	0x300001	Instr1[15:0]	Instr0[47:32]
0x200001	0x300001	Instr1[47:16]	
	0x300002	Instr2[31:0]	
0x200002	0x300002	Instr3[15:0]	Instr2[47:32]
	0x300003	Instr3[47:16]	
...	...		
	...		
...	...		
	...		
0xFFFFF	0x17FFFFE	InstrP[31:0]	
	0x17FFFFF	0000	InstrN[47:32]

In [Table 3-6](#), P = 0xE00000. Therefore, the total number of external memory instructions for a 16-bit wide SDRAM memory is 14 million.

Table 3-6. Logical Versus Physical Address Mapping, 16-Bit SDRAM Memory

Logical Address Dispatched by Program Sequencer	Physical Address Observed on the External Address Bus	Data	
		16	0
0x200000	0x600000	Instr0[15:0]	
	0x600001	Instr0[31:16]	
	0x600002	Instr0[47:32]	


Table 3-6. Logical Versus Physical Address Mapping, 16-Bit SDRAM Memory (Cont'd)

Logical Address Dispatched by Program Sequencer	Physical Address Observed on the External Address Bus	Data	
		16	0
0x200001	0x600003	Instr1[15:0]	
	0x600004	Instr1[31:16]	
	0x600005	Instr1[47:32]	
0x200002	0x600006	Instr2[15:0]	
	0x600007	Instr2[31:16]	
	0x600008	Instr2[47:32]	
...	...		
...	...		
...	...		
0xFFFFF	0x2FFFFFD	InstrN[15:0]	
	0x2FFFFFE	InstrN[31:16]	
	0x2FFFFFF	InstrN[47:32]	

Register Configurations for External Memory Execution

If bank 0 memory is asynchronous memory (such as SRAM or flash) then programs need to appropriately configure the asynchronous memory interface control register (AMICTL) and the external port control register (EPCTL). Bits [2:1] of the AMICTL register configure the external bus width as either 16 or 32 bits wide. If bank 0 memory is SDRAM, then programs need to configure the SDRAM control register (SDCTL). Bit 16 determines whether the external bus data width is either 16-bits wide, or 32-bits wide.

The default packing mode on the ADSP-21375 is 16-to-48 bit mode packing, while for the ADSP-21371 it is 32-to-48 bit mode packing.

 For the ADSP-21371 processor, the `SDCTL` register needs to be explicitly programmed for 16-bit wide external memory by setting bit 16 (`X16DE`) of this register.

EMI Registers and Signals

The external port global control register is used to set the priority between core and DMA memory accesses and to determine whether SDRAM or asynchronous memory is used on each bank. The bits in this register are described in [Table 3-7](#).

In multiple clock domains, the effect latency of the control register bits of the external port varies. The worst case is 1:4, where the time from a control bit write, to the time the write takes effect, is a maximum of four IOP clock cycles/eight core clock cycles. It is essential to put `NOP` commands (no operation) in the program to accommodate this time. It is also advised that programs not perform any external memory accesses until the effect takes place. The status bits of the control register can also have a maximum latency of up to one cycle.


 The `EPCTL` register bits should not be changed during external accesses.

Table 3-7. External Port Control Register Bit Descriptions

Bit	Name	Description	Default
0	B0SD	Bank 0 SDRAM. 1 = Bank 0 ($\overline{MS0}$) connected to SDRAM 0 = Bank 0 ($\overline{MS0}$) connected to asynchronous memory	0
1	B1SD	Bank 1 SDRAM. 1 = Bank 1 ($\overline{MS1}$) connected to SDRAM 0 = Bank 1 ($\overline{MS1}$) connected to asynchronous memory	0
2	B2SD	Bank 2 SDRAM. 1 = Bank 2 ($\overline{MS2}$) connected to SDRAM 0 = Bank 2 ($\overline{MS2}$) connected to asynchronous memory	0

Table 3-7. External Port Control Register Bit Descriptions (Cont'd)

Bit	Name	Description	Default
3	B3SD	Bank 3 SDRAM. 1 = Bank 3 ($\overline{MS3}$) connected to SDRAM 0 = Bank 3 ($\overline{MS3}$) connected to asynchronous memory	0
5–4	EPBR	External Port Bus Priority. 11 = Rotating priority 10 = Core has high priority 01 = DMA has high priority 00 = Reserved	11
7–6	DMAPR	DMA channel Priority for CH0 and CH1. 11 = Rotating priority 10 = Fixed priority 01 = Reserved 00 = Reserved	11
8	Reserved		
10–9	FRZDMA	Arbitration Freezing Length for DMA. 0 = No freezing 1 = 4 Accesses 2 = 8 Accesses 3 = 16 Accesses	0
12–11	Reserved		
14–13	FRZCR	Arbitration Freezing Length for CORE Accesses. 0 = No freezing 1 = 4 Accesses 2 = 8 Accesses 3 = 16 Accesses	0
18–15	DATE	DATA Enable. When the SDRAM/AMI memory controller is in no pack mode, these bits of the data lane are masked with zeros. The data lane is 8 bits. The 32-bit data bus has four data lanes. DATA[31:0] is mapped to DL3, DL2, DL1, DL0. For example, if DATE is 1010, then DL3 and DL1 are masked with zeros.	0000
19	Reserved		

External Memory Interface

External Port Arbitration Logic

The external port arbitration logic controls the arbitration between the two DMA channels and processor core. The following control the arbitration logic.

- The EPCTL register can be programmed to use the various features of arbitration between different channels. DMA channels 0 and 1 can be programmed for rotating or fixed priority.
- The winning DMA channel can be arbitrated with the core channel. The EBPR and DMAPR bits define the priorities.

Channel Freezing

The external port is idle when DMA engines are idle and no core access is pending. When multiple DMA channels are reading data from SDRAM memory, channel *freezing* can improve the data throughput. By setting the freeze bits (FRZDMA, bits 10–9 and FRZCR, bits 14–13), each channel is frozen for programmed accesses. For example, if the processor core is frozen for 16 accesses, and if the core requests 16 accesses to SDRAM sequentially, data throughput improves. Freezing is based on the fact that sequential accesses to the SDRAM provide better throughput than non-sequential accesses. Freezing does not add value for write accesses. For details on throughput, see [“SDRAM Timing” on page 3-74](#).

Managing Data Paths

The DATE bits (bits 18–15) are used to enable the data paths in the input path. If the DATE bits are set (=1), then the incoming data that corresponds to the set bits are connected to zero. This helps to avoid the floating pin data coming in to the processor.

External Memory Interface Pins

The pins used by the external memory interface are described in [Table 3-8](#).

Table 3-8. External Memory Pin Descriptions

Pin Name	I/O	Description for AMI	Description for SDRAM
DATA31–0	I/O	Data bus	Data bus
ADDR23–0	O	Address bus	Address bus, includes bank selects ADDR [23:0]
SDCLK	O	N/A	SDRAM clock
SDCKE	O	N/A	SDRAM clock enable
SDA10	O	N/A	SDRAM address bit 10 used for auto refresh
$\overline{\text{SDRAS}}$	O	N/A	SDRAM row address strobe
$\overline{\text{SDCAS}}$	O	N/A	SDRAM column address strobe
$\overline{\text{SDWE}}$	O	N/A	SDRAM write enable
$\overline{\text{MS3}}\text{--}0$	O	Chip select	Chip select. $\overline{\text{MS1}}\text{--}0$, FLAG2 and FLAG3 are muxed to form $\overline{\text{MS3}}\text{--}2$, (FLAG3 is $\overline{\text{MS3}}$ and FLAG2 is $\overline{\text{MS2}}$)
$\overline{\text{RD}}$	O	Read output strobe	N/A
$\overline{\text{WR}}$	O	Write output strobe	N/A
ACK	I	Acknowledge signal	N/A

Asynchronous Memory Interface

Both the processor core and the I/O processor have access to external memory using the AMI. [Table 3-9](#) describes the processor pins used for interfacing to external memory.

The processor's memory control signals also permit direct connection to fast static RAM devices. Memory-mapped peripherals and slower memories can also connect to the processor using a user-defined combination of programmable waitstates and hardware acknowledge signals.

External memory can hold data and packed instructions (the ADSP-2136x cannot execute from external memory). Data packing of 16 to 32 bits or 8 to 32 bits is supported for transfers directly from 32-bit, 16-bit, or 8-bit wide external memories to and from internal memory.

Table 3-9. Asynchronous Memory Interface Signals

Pin	Type	Description
ACK	I	Memory Acknowledge. External devices can deassert ACK (low) to hold off an external memory access. ACK is used by I/O devices, memory controllers, or other peripherals to hold off completion of an external memory access. ACK has a 22.5 k Ω internal pull-up resistor that is enabled during reset.
ADDR23-0	O	External Bus Address. The processor outputs addresses for external memory and peripherals on these pins. A pull-up enabled on the processor's ADDR23-0 pins maintains the input at the level it was last driven. This pull-up is only enabled on the processor with ID2-0=01x in shared memory system during reset. Note that only the ADSP-21368 processor has shared memory capability.
DATA31-0	I/O	External Bus Data. The processor inputs and outputs data on these pins. Pull-up resistors on unused data pins are not necessary. A pull-up on the processor's DATA31-0 pins maintains the input at the level it was last driven. This pull-up is only enabled on the processors with ID2-0=01x in shared memory system during reset.

Table 3-9. Asynchronous Memory Interface Signals (Cont'd)

Pin	Type	Description
$\overline{\text{MS3-0}}$	O	Memory Select Lines [FLAG2-3 are muxed and used as MS2 and MS3]. Memory select lines 0–1 are asserted (low) as chip selects for the corresponding banks of external memory. The $\overline{\text{MS3-0}}$ lines are decoded memory address lines that change at the same time as the other address lines. When no external memory access is occurring, the $\overline{\text{MS3-0}}$ lines are inactive; they are active however when a conditional memory access instruction is executed, whether or not the condition is true.
$\overline{\text{RD}}$	O	Memory Read Strobe. $\overline{\text{RD}}$ is asserted whenever the processor reads a word from external memory. In a shared memory system, $\overline{\text{RD}}$ is driven by the bus master. $\overline{\text{RD}}$ has a 22.5 k Ω internal pull-up resistor that is enabled for processors with ID2-0=01x during reset.
$\overline{\text{WR}}$	O	Memory Write Strobe. $\overline{\text{WR}}$ is asserted when the processor writes a word to external memory. In a shared memory system, $\overline{\text{WR}}$ is driven by the bus master. $\overline{\text{WR}}$ has a 22.5 k Ω internal pull-up resistor that is enabled for processors with ID2-0=01x during reset.

AMI Timing Control

The following three sections introduce the available control settings that affect the timing used to make AMI accesses. Note that all accesses are in terms of SDCLK since the SDRAM controller and AMI share this clock.

Wait States

Wait states and acknowledge signals are used to allow the processors to connect to memory-mapped peripherals and slower memories. Wait states are programmable from 1 to 31 using the WS bits (bits 10–6) in the AMI control register. Wait states are programmed relative to SDCLK .

When ACK is enabled, the wait state value should be set to indicate when the processor can sample ACK after the $\overline{\text{RD}}/\overline{\text{WR}}$ edge goes low. The minimum wait state value that can be used is $\text{WS} = 1$ when ACK is enabled. If ACK is not

enabled, the minimum value is $WS = 2$ (a wait state value of 0 corresponds to 32 wait cycles). The processor samples the `ACK` signal after the programmed wait state count expires—it is imperative that the `WS` value is initialized when the acknowledge enable bit (`ACKEN`) is set.

Bus Idle Cycles

A bus idle cycle (`IC` bits 16–14 in the `AMICTLx` registers) is an inactive bus cycle that the processor automatically generates to avoid data bus driver conflicts. Such a conflict can occur when a device with a long output disable time continues to drive after \overline{RD} is deasserted, while another device begins driving on the following cycle. Idle cycles are also required to provide time for a slave in one bank to three-state its `ACK` driver, before the slave in the next bank enables its `ACK` driver in synchronous access modes. [Figure 3-4](#) shows idle cycle insertion between a synchronous read and a zero-wait, synchronous write in cycle 3.

To avoid this data bus driver conflict, the processor generates an idle cycle in the following cases:

- On a transition from a read operation to a write operation
- On a transition from read from one bank to another bank
- On a transition from read from one bank to external access from another device such as an SDRAM controller or another master in a shared memory system



Unlike previous SHARC processors, the ADSP-21367/8/9 and ADSP-2137x SHARC processors do not support idle cycle insertion on a page boundary crossing. If an idle cycle is programmed for a particular bank, then a minimum of one cycle is inserted for reads, even if they are from the same bank.

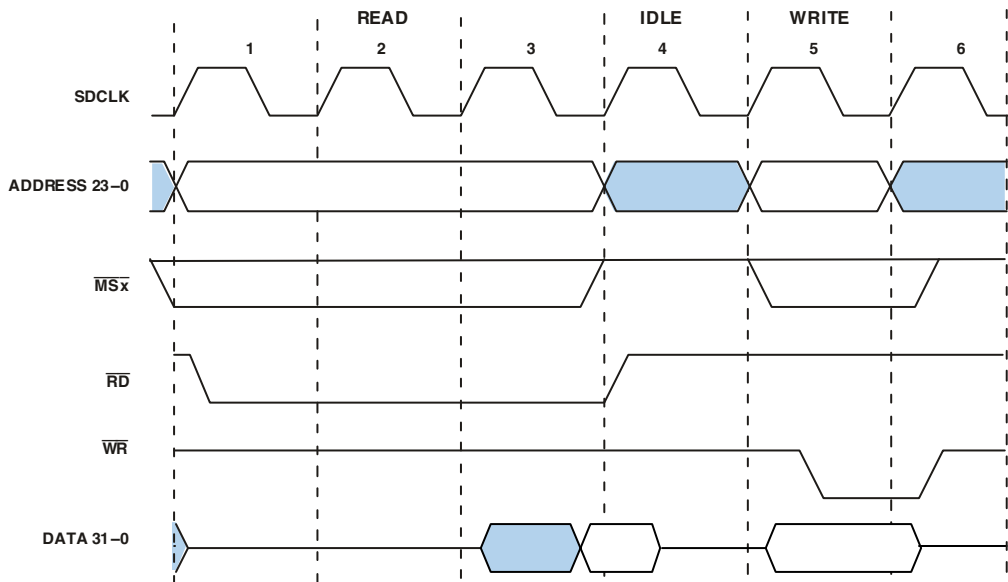


Figure 3-4. Idle Cycle Example, Wait State = 2

Bus Hold Cycles

The processor is able to insert bus hold and read hold cycles by setting bits in the AMI control register (AMICTLX). These two methods for holding off data processing are described below.

- **Bus hold cycle** (HC bits 13–11) is an inactive bus cycle that the processor automatically generates at the end of a read or write to allow a longer hold time for address and data. Programs may disable holds, or hold off processing for one or more external port processor cycles.
- **Read hold cycle** (RHC bits 20–18) is the delay between two reads at the end of a read access. Programs may disable the read hold cycle, or hold the address for one or more external port clock cycles.

Setting AMI Modes

The address, data (if a write), and bank select (if in banked external memory) remain unchanged and are driven for one or more cycles after the read or write strobes are deasserted. [Figure 3-5](#) demonstrates a hold time cycle appended to an asynchronous write access ($EBxWS = 011$).

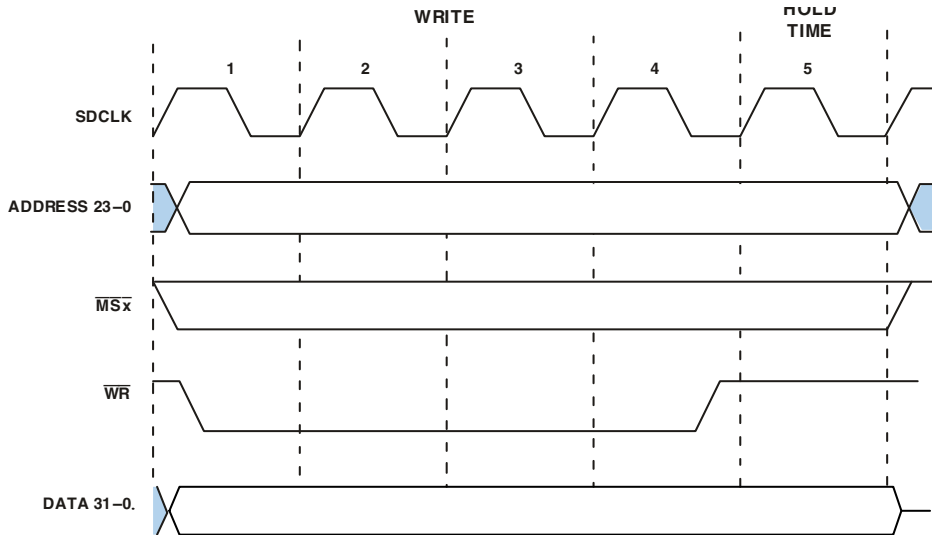



Figure 3-5. Hold Time Cycle Example

Setting AMI Modes

The `AMICTLx` registers control the asynchronous memory interface's operating mode and the `AMISTAT` register provides status information.

[Table A-5 on page A-18](#) lists all the bits in the `AMICTLx` registers and [Figure A-7 on page A-20](#) shows all the bits in the `AMISTAT` register.

- For information on using DMA through the external port, see [“External Port DMA” on page 2-35](#).
- For information on using external port interrupts, see [“Interrupt-Driven I/O” on page 2-6](#).

-  There is a 3:1 bus conflict resolution ratio at the external port interface (three internal buses to one external bus) in addition to the 2:1 or greater clock ratio between the processor's internal clock and the external SDRAM clock ($SDCLK$). Systems that fetch data through the external port must tolerate at least one cycle—and possibly many additional cycles—of latency for non-SDRAM accesses.

Data alignment through the external port is identical for these interfaces.

External Memory Reads

The AMI behaves as an external port bus slave and an external read access is performed only upon a read request from the external port control bus. Reads from external memory are done through the AMI. When an external address that is mapped to the AMI in the $EPCTL$ register is accessed, the module receives 8-, 16-, or 32-bit data and packs the data based on the packing and control modes in the AMI control register ($AMICTLx$). Once the data is received, the status bit RXS is set, indicating valid data is present. Updates occur on a feedback \overline{RD} signal and the status update occurs on the $SDCLK$ signal. The RXS status update is delayed by one cycle to accommodate the delay in the feedback \overline{RD} signal.

The AMI provides the interface to the external data pins as well as to the processor core or to the internal DMA controller. When the AMI receives data, it is passed by internal hardware to the DMA controller or to the external port control bus, depending on which entity requested the data.

Data Packing

Data packing for memory reads is accomplished using the packing disable ($PKDIS$, bit 3) and most significant word first (bit 4 in the $MSWF$ register) bits in the $AMICTLx$ register.

For packed data mode where $PKDIS = 0$ and $MSWF = 0$, the packing order for $BW = 8$ is: first byte is bits [7–0], the second byte is bits [15–8], and so on.

Setting AMI Modes

For packed data mode where `PKDIS = 0` and `MSWF = 1`, the packing order for `BW = 8` is: first byte received is bits 31–24, the second byte is bits 23–15, and so on.

If the `PKDIS` bit is set (`=1`), then the 8- or 16-bit data (based on the bus width) is zero appended to 32 bits.

Both of these methods apply to 16- to 32-bit packing as well. These modes are summarized in [Table 3-10](#).

Table 3-10. Data Packing Bit Settings (Reads)

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

External Memory Writes

The AMI behaves as an external port bus slave and external write accesses are performed when there is a write request from the external port control bus. Writes to external memory are done through the AMI module. When an external address that is mapped to the AMI in the `EPCTL` register is accessed, the module receives data from internal memory using the DMA

controller or through direct core writes. Writes to the AMI set a status bit (`AMITXS`, bit 2 in the `AMISTAT` register) and initiate the external write access.

Once a full word is transferred out of the AMI, the `AMITXS` bit is cleared and new writes are allowed. No more external transfers can start while the AMI module is empty.

Whenever the `AMITX` is empty, the DMA controller or a direct access from the processor core can write fresh data into the AMI. If the register is full, further writes from the core (or DMA controller) are stalled.

Data Packing

Data unpacking for memory writes uses the packing disable bit (`PKDIS`, bit 3) and the most significant word first (`MSWF`, bit 4) bits in the `AMICTLX` register.

For packed data mode where `PKDIS = 0` `MSWF = 0`, the order of unpacking for 32- to 8-bit data is: the first byte is bits 7–0, the second byte is bits 15–8, and so on.

For packed data mode where `PKDIS = 0` `MSWF = 1`, the unpacking order for 32- to 8-bit data is: first byte received is bits 31–24, the second byte is bits 23–16, and so on.

If `PKDIS` bit is set (=1), only the 16- or 8-bit least significant portion of the 32-bit data is written to external memory.

Both of these methods apply to 32- to 16-bit unpacking as well. These modes are summarized in [Table 3-10](#).



For direct access (core and DMA), the received data is also unpacked, depending on the setting of the `PKDIS` bit. The order of unpacking is dependent on the `MSWF` bit in `AMICTLX` registers.

Read/Write Throughput

For a wait state of 2 (which is the smallest wait state), the throughput is shown in [Table 3-11](#).

Table 3-11. Read/Write Throughput

Bus Width	Operation	Throughput in SDCLK Cycles
32-bit	Write	One 32-bit word per 3 SDCLK cycles
32-bit	Read	One 32-bit word per 3 SDCLK cycles
16-bit	Write	One 32-bit word per 6 SDCLK cycles
16-bit	Read	One 32-bit word per 6 SDCLK cycles
8-bit	Write	One 32-bit word per 12 SDCLK cycles
8-bit	Read	One 32-bit word per 12 SDCLK cycles

External Access Addressing

The AMI supports 16M words of external memory in bank1, bank2, and bank3 and 14M words of external memory in bank0. The maximum amount of external data is 64M bytes when the external bus width (set using the BW bits 2–1 in the `AMICTLx` registers) is 32 bits on bank1, bank2, or bank3.

The ADSP-21367/8/9 and ADSP-2137x processors have the ability to use logical addressing when an external memory smaller than 32 bits is used. When logical addresses are used, multiple external addresses seen by the memory correspond to a single internal address, depending on the width of the memory being accessed, and the packing mode setting of the AMI or SDRAM controller. The following are examples of logical addressing.

- For an external bus width of 32 bits, or when packing is disabled with other bus widths (PKDIS = 1 and BW = 16 bits or PKDIS = 1 and BW = 8 bits), then the external physical memory is the same as the lower 24 bits of bits 23–0 in the address being supplied to the external port by the core or DMA controller.
- For an external bus width of 16 bits with packing enabled (PKDIS = 0), the external physical address ADDR23–0 generation is ADDR23–1 = bits 22–0 in the address being supplied to the external port by the core or DMA controller. Here, ADDR[0] corresponds to the 1st/2nd 16-bit word.
- For an external bus width of 8 bits with packing enabled (PKDIS = 0), the external physical address ADDR23–0 generation is ADDR23–2 = bits 21–0 in the address being supplied to the external port by the core or DMA controller. Here, ADDR1–0 corresponds to the 1st/2nd/3rd/4th 8-bit word.

The external physical address map is shown in [Table 3-12](#).

Table 3-12. AMI Address Memory Map

Bus Width	External Memory BANK	External Physical Address (on ADDR23–0)
32-bit (or PKDIS=1)	0	0x20_0000 to 0xFF_FFFF
32-bit (or PKDIS=1)	1, 2, and 3	0x00_0000 to 0xFF_FFFF
16-bit (and PKDIS=0)	0	0x40_0000 to 0xFF_FFFF
16-bit (and PKDIS=0)	1, 2, and 3	0x00_0000 to 0xFF_FFFF
8-bit (and PKDIS=0)	0	0x80_0000 to 0xFF_FFFF
8-bit (and PKDIS=0)	1, 2, and 3	0x00_0000 to 0xFF_FFFF

External Port DMA

The AMI shares the two DMA channels of the external port with the SDRAM controller. Either of these DMA channels can be directed to the external asynchronous memories. For information on external port DMA, see [Chapter 2, I/O Processor](#).

Bootling Through the AMI

The AMI supports an 8-bit user boot called AMI boot. [For more information, see “Bootling Through the AMI” on page 14-39.](#)

SDRAM Controller

The ADSP-21367/8/9 and ADSP-2137x SHARC processors support a glueless interface with any of the standard SDRAMs of 64M bit, 128M bit, 256M bit, and 512M bit with configurations x4, x8, x16 and x32. The SDRAM controller (SDC) can support up to 254M words of SDRAM in four banks. Bank 0 can accommodate up to 62M words, and banks 1, 2, and 3 can accommodate up to 64M words each. The SDC includes timing options to support additional buffers between the processors and SDRAM. This allows the processor to handle the capacitive loads of large memory arrays. The following are additional features of the SDC.

- I/O width 16-bit or 32-bits, I/O supply 3.3 V
- Types of 32, 64, 128, 256, and 512M bit with I/O of x4, x8, x16 and x32
- Page sizes of 128, 256, 512, 1k, 2k words
- SDC uses no-burst mode (BL = 1) with sequential burst type
- SDC uses optional full page burst (ADSP-2137x only)

- SDC uses open page policy—any open page is closed only if a new access in another page of the same bank occurs
- Supports multibank operation within the SDRAM (ADSP-2137x only)
- Uses a programmable refresh counter to coordinate between varying clock frequencies and the SDRAM's required refresh rate
- Provides multiple timing options to support additional buffers between the processor and SDRAM
- Allows independent auto-refresh while the asynchronous memory interface (AMI) has control of the External Port
- Supports self-refresh mode for power savings
- Supports instruction fetch (ADSP-2137x only)
- Supports 32-bit data access by the processor core



All inputs are sampled and all outputs are valid on the rising edge of the SDRAM clock output (SDCLK).

Definition of Terms

The following are terms commonly used in SDRAM systems.

Bank activate command

The bank activate command causes the SDRAM to open an internal bank of memory (specified by the bank address) in a specific row (specified by the row address). [For more information, see “SDC Commands” on page 3-63.](#)

Burst length

The burst length determines the number of words that the SDRAM device stores or delivers after detecting a single write or read command, respectively.

The SDC supports burst length = 1 mode only.

Burst stop command

Use of this command is one of several ways to terminate or interrupt a burst read or write operation. Since the burst length is hardwired to 1, the SDC does not support the burst stop command. However, this command is optionally supported by ADSP-2137x processors.

Burst type

The burst type determines the address order in which the SDRAM delivers burst data after detecting a read command, or stores burst data after detecting a write command.

Since the burst length is always programmed to 1, the burst type does not apply. However, the SDC always sets the burst type to sequential-accesses-only during the SDRAM power-up sequence.

CAS latency

Also t_{AA} , t_{CAC} . The column address strobe (CAS) latency is the delay in clock cycles between when the SDRAM detects the read command and when it provides the data at its output pins. The CAS latency is programmed in the SDRAM mode register during the power-up sequence with the value programmed in the SDRAM control register (SDCTL bits 1-0).

The speed grade of the SDRAM and the SDCLK frequency determine the value of the CAS latency. The SDC supports CAS latencies of 2 or 3 clock cycles. The selected CAS latency value is programmed into the SDCTL register before the SDRAM power-up sequence.

CBR (CAS before RAS)

Refresh or auto-refresh. When the SDC refresh counter times out, the SDC precharges all four banks of SDRAM and then issues an auto-refresh command to them. This causes the SDRAMs to generate an internal CBR refresh cycle. When the internal refresh completes, the SDRAM banks are precharged.

Data mask feature

SDRAM's allow partial read or writes in byte addressing mode. Since SHARC processors do not support byte addressing, *DQM* pins are not controlled by the SDC.

Internal bank

There are several internal memory banks on a given SDRAM row. An internal bank in a specific row cannot be activated (opened) until the previous internal bank in that row has been precharged.

The SDC does not support multibank accesses. The bank address can be thought of as part of the row address. The SDC also assumes that all SDRAMs to which it interfaces have four internal banks.



Only the SDRAM controller on the ADSP-2137x processors supports multibank accesses.

Mode register

SDRAM devices contain an internal configuration register which allows specification of the SDRAM device's functionality. During power-up, and before executing a read or write to the SDRAM memory space, the application must trigger the SDC to write to the SDRAM's mode register. The write of this register is triggered by writing a 1 to the *SDPSS* bit (bit 14) in the processor's SDRAM control register (*SDCTL*), and then issuing a read or write transfer to the SDRAM address space.

The initial read or write triggers the SDRAM power-up sequence, which programs the SDRAM's mode register with the CAS latency from the `SDCTL` register. This initial read or write to SDRAM takes many cycles to complete. Note that for most applications the SDRAM power-up sequence and a write to the mode register is performed only once. Once the power-up sequence has completed, the `SDPSS` bit should not be set again unless a change to the mode register is desired.

Page size

Page size is the amount of memory which has the same *row address* and can be accessed with successive read or write commands without needing to activate another row.

Precharge command

The precharge command closes a specific internal bank in the active page or all internal banks in the page. [For more information, see “SDC Commands” on page 3-63.](#)

Self-refresh

When the SDRAM is in self-refresh mode, its internal timer initiates auto-refresh cycles periodically, without external control input. The SDC must issue a series of commands, including the self-refresh command, to put the SDRAM into this low power mode. It must issue another series of commands to exit self-refresh mode. Entering self-refresh mode is programmable in the SDRAM control register (`SDCTL`) and any access to the SDRAM address space causes the SDC to exit the SDRAM from self-refresh mode. [For more information, see “Self-Refresh Mode” on page 3-70.](#)

tRAS

Required delay between issuing a bank activate command and a precharge command, and between issuing the self-refresh command and the exit from self-refresh mode. The `SDTRAS` bits (7–4) in the `SDCTL` register can be set to 1 to 15 clock cycles.

tRP

Required delay between issuing a precharge command and issuing:

- a bank activate command
- an auto-refresh command
- a self-refresh command

The `SDTRP` bits (10–8) in the `SDCTL` register can be set to 1 to 7 clock cycles.

tMRD

Required delay between issuing a mode register set command and a successive bank activate command. This delay is not directly programmable and is assumed to be 2 clock cycles.

tRCD

Required delay between a bank activate command and the start of the first read or write command. The `SDTRCD` bits (26–24) in the `SDCTL` register can be set to 1 to 7 clock cycles.

tWR

Required delay between a write command (driving write data) and a precharge command. The `SDTWR` bits (18–17) in the `SDCTL` register can be set to 1 to 3 clock cycles.

SDRAM Controller

t_{RC}

Required delay between issuing successive bank activate commands to the same SDRAM internal bank. This delay is not directly programmable.

The t_{RC} delay is satisfied by programming the SDTRAS and SDTRP fields to ensure that $t_{RAS} + t_{RP} \geq t_{RC}$.

t_{RFC}

Required delay between issuing an auto-refresh command and a bank activate command, and between issuing successive auto-refresh commands.

This delay is not directly programmable and is assumed to be equal to t_{RC}.

The t_{RC} delay is satisfied by programming the SDTRAS and SDTRP fields to ensure that $t_{RAS} + t_{RP} \geq t_{RC}$.

t_{RRD}

This is the required delay between a bank A activate command and a bank B activate command. This delay is not programmable and fixed to t_{RCD} + 1 cycles. This delay is used for multibank operation (ADSP-2137x processors only).

t_{XS}

Required delay between exiting the self-refresh mode and issuing an auto-refresh command. This delay is not directly programmable and is assumed to be equal to t_{RC}. The t_{RC} delay is satisfied by programming the t_{RAS} and t_{RP} fields to ensure that $t_{RAS} + t_{RP} \geq t_{RC}$.

Timing External Memory Accesses

The SDRAM controller is capable of running at up to 166 MHz and can run at various frequencies, depending on the programmed SDRAM clock (SDCLK) to core clock (CCLK) ratios. These are shown in [Table 3-13](#).

The SDRAM CAS latency, (SDCL, SDTRAS bits), precharge (SDTRP bits), RAS to CAS delay (SDTRCD bits), and write before precharge timing (SDTWR bits) should be programmed based on the SDRAM clock frequency and the timing specifications of the SDRAM used. All timing parameters are written with valid values based on the SDCLK clock frequency and the timing specifications of the SDRAM before any access to SDRAM address space, including the power-up sequence. Note that the programmed parameters apply to all four external memory banks.

Also note that these timing parameters should not be changed while the SDC is active.

Table 3-13. SDC Clock Frequencies

CK/SDCLK Clock Period Ratio	SDRAM Frequency (400 MHZ)	SDRAM Frequency (333 MHZ)	SDRAM Frequency (266 MHZ)
1:2.0	166 (not supported)	166	133
1:2.5	160	133	106
1:3.0	133	111	88
1:3.5	114	95	76
1:4.0	100	83	66



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

The following procedure may be used to change the SDRAM clock ratio. Note that this procedure changes only the output divider.

1. Select the PLL divider by setting the PLLD_x bits (bits 6–7 in the PMCTL register) to one of the following values.
2. Select the SDCLK divider (SDCLK to CCLK ratio) by setting the SDCKR_x bits (bits 18–20 in the PMCTL register) to one of the following five values.

PLLD Bit Setting	Clock Ratio
00	CCLK divider of 1
01	CCLK divider of 2
10	CCLK divider of 4
11	CCLK divider of 8

SDCKR Bit Setting	Clock Ratio
000	SDCLK divider of 2
001	SDCLK divider of 2.5
010	SDCLK divider of 3
011	SDCLK divider of 3.5
100	SDCLK divider of 4

3. Enable the new divisors by setting the DIVEN bit (bit 9 in the PMCTL register). Do not set this bit at the same time as the PLLBP bit (bit 15 of the PMCTL register) is set. See [“Power Management Control Register” on page 14-14](#) for more information.

The new divisor ratios are picked up on the fly and the clocks smoothly transition to their new values within 14 core clock (CCLK) cycles.

The core clock frequency is:

$CCLK = CLKIN \times (PLL \text{ multiplier} \div \text{clock divider})$ where:

PLL multiplier = PLLM (1–64) and the PLL divider = 1, 2, 4 or 8.

The SDCLK frequency is $SDCLK = CCLK \div SDRATIO$.

If either the PLL divider or the SDCLK to CCLK ratio (or both) are changed, it may take up to 14 CCLK cycles for all the clocks to get the new value.


For more information on SDRAM clocking and programming the PLL, see [“Clock Derivation” on page 14-13](#), [“Power Management Control Register” on page 14-14](#), and [“Power Management Control Register \(PMCTL\)” on page A-170](#).

Parallel Connection of SDRAMs

To specify a SDRAM system, multiple possibilities are given based on the different memory sizes. For a 32-bit I/O capability, the following can be configured.

- 1 x 32-bit/page 256 words
- 2 x 16-bit/page 512 words
- 4 x 8-bit/page 1k words
- 8 x 4-bit/page 2k words

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

 Even if connecting SDRAMs in parallel, the SDC always considers the cluster as one external SDRAM bank because all address and control lines feed the parallel parts.

SDRAM Control Register (SDCTL)

The SDRAM memory control register includes all programmable parameters associated with the SDRAM access timing and configuration. These bits are described below. [For more information, see “SDRAM Control Register \(SDCTL\)” on page A-21.](#)

SDRAM CAS latency parameter setting. $SDCL$ bits 1–0. The column address strobe (CAS) latency is the delay in clock cycles between when the SDRAM detects the read command and when it provides the data at its output pins. Settings are: 10 = 2 cycles, 11 = 3 cycles.

Generally, the frequency of operation determines the value of the CAS latency. For specific information about setting this value, consult the SDRAM device documentation.

SDRAM controller disable. $DSDCTL$ bit 2. Enables or disables the SDC. If $DSDCTL$ is set (=1), any access to SDRAM address space does not occur externally. All SDC control pins are in their inactive states and the SDRAM clock, $SDCLK$, does not run.

The $DSDCTL$ bit is cleared (=0) by default, so that $SDCLK$ is running after reset deasserts. If the SDC is not used, the $DSDCTL$ bit can be set to stop the clock and reduce power dissipation. Even though the SDC is enabled at reset, the power-up sequence must be executed before reading or writing to SDRAM address space. Failure to execute the power-up sequence before reading or writing to SDRAM address space results in unpredictable operation. However, $DSDCTL$ must remain cleared at all times when the SDC is needed to generate auto-refresh commands to the SDRAM.

SDRAM t_{RAS} parameter setting. $SDTRAS$ bits 7–4. The t_{RAS} value (bank activate command delay) defines the required delay, in number of $SDCLK$ cycles, between the time the SDC issues a bank activate command and the time it issues a precharge command as shown below.

$$SDTRAS \geq \frac{t_{RASmin}}{t_{SDCLK}}$$

The SDRAM must also remain in self-refresh mode for a period of time of at least t_{RAS} . The t_{RP} and t_{RAS} values define the t_{RFC} , t_{RC} , and t_{XSR} values. [For more information, see “Timing External Memory Accesses” on page 3-36.](#)

The t_{RAS} parameter allows the ADSP-21367/8/9 and ADSP-2137x processors to adapt to the timing requirements of the system's SDRAM devices. Any value between 1 and 15 $SDCLK$ cycles can be selected as shown in [Table 3-14](#).

Table 3-14. Bank Activate Command Delay Bit Settings

Bit Setting	Clock Cycles	Bit Setting	Clock Cycles
SDTRAS1 = 0000	Reserved	SDTRAS8 = 1000	8
SDTRAS1 = 0001	1	SDTRAS9 = 1001	9
SDTRAS2 = 0010	2	SDTRAS10 = 1010	10
SDTRAS3 = 0011	3	SDTRAS11 = 1011	11
SDTRAS4 = 0100	4	SDTRAS12 = 1100	12
SDTRAS5 = 0101	5	SDTRAS13 = 1101	13
SDTRAS6 = 0110	6	SDTRAS14 = 1110	14
SDTRAS7 = 0111	7	SDTRAS15 = 1111	15

SDRAM t_{RP} parameter setting. $SDTRP$ bits 10–8. Defines the required precharge delay, in number of $SDCLK$ cycles, between the time the SDC issues a precharge command and the time it issues a bank activate command as shown in the following equation.

$$SDTRP \geq \frac{t_{RPmin}}{t_{SDCLK}}$$

The t_{RP} setting also specifies the time required between precharge and auto-refresh, and between precharge and self-refresh. Any value between 1 and 7 $SDCLK$ cycles may be selected as shown in [Table 3-15](#).

Table 3-15. Precharge Delay Bit Settings

Bit Setting	Clock Cycles	Bit Setting	Clock Cycles
000	Reserved	SDTRP4 = 100	4
SDTRP1 = 001	1	SDTRP5 = 101	5
SDTRP2 = 010	2	SDTRP6 = 110	6
SDTRP3 = 011	3	SDTRP7 = 111	7

SDRAM power-up mode. SDPM bit 11. If the SDPM bit is set (=1), the SDC does a precharge all command, followed by a load mode register command, and eight auto-refresh cycles. If the SDPM bit is cleared (=0), the SDC does a precharge all command, followed by eight auto-refresh cycles, and a load mode register command.

SDRAM bank column address width. SDCAW bits 13–12 shown in [Table 3-16](#). Sets the SDRAM page size. Page sizes of 256 and 512 bits, and 1K and 2K bits are supported. [Table 3-22 on page 3-53](#) shows the page size and breakdown of the internal address (IA31-0), as seen from the core into the row, bank, and column address. The column address makes up the address inside the page.

Table 3-16. SDRAM Bank Column Address Width Bit Settings

SDCAW Bit Setting	SDRAM Bank Column Address Width
SDCAW8 = 00	8 bits (256 words)
SDCAW9 = 01	9 bits (512 words)
SDCAW10 = 10	10 bits (1K words)
SDCAW11 = 11	11 bits (2K words)



Programming the SDC with non-supported page size values produces unpredictable results.

SDRAM power-up sequence start. SDPSS bit 14. The SDPM bit specifies the power-up mode and the SDPSS bit starts an SDRAM power-up (initialization) sequence. When this bit is set (=1), the SDRAM power-up sequence starts on the next SDRAM access. When cleared, this bit has no effect. This bit always reads zero. See also [“Load Mode Register” on page 3-64](#).

SDRAM self-refresh command. SDSRF bit 15. When set (=1), starts the self-refresh mode. When cleared (=0) this bit has no effect. This bit always reads zero. In self-refresh mode, the SDRAM performs refresh operations internally which reduces the SDRAM’s power consumption.

When SDSRF is set to 1, the SDC enters an idle state. In this state, it issues a precharge command (if necessary) and then issues a self-refresh command. If an internal access is pending, the SDC delays issuing the self-refresh command until it completes all pending SDRAM access requests. Once the SDRAM device enters into self-refresh mode, the SDRAM controller asserts the SDSRA bit in the SDRAM control status register (SDSTAT). The SDRAM controller ignores other self-refresh requests when the SDRAM device is already in self-refresh mode.



The SDRAM device exits self-refresh mode only when the SDC receives a request for SDRAM space access. There is no way to cancel entry into self-refresh mode.

SDRAM external data path width. X16DE bit 16. Selects whether the SDRAM interface is 32 or 16 bits wide.

- If set (=1), a 16-bit SDRAM should be used;
DATA[15:0] should be connected to the SDRAM data pins;
ADDR[14:0] should be connected to SDRAM address pins 14–0;
16 to 32-bit packing is performed.
Note that ADDR[18:17] pins are also used. See tables [Table 3-22 on page 3-53](#) to [Table 3-25 on page 3-57](#) for bank address usage.

SDRAM Controller

- If cleared (=0), a 32-bit SDRAM should be used;
DATA[31:0] should be connected to the SDRAM data pins;
ADDR[15:1] should be connected SDRAM address pins 14–0. [For more information, see “SDRAM Address Mapping” on page 3-51.](#)
Note that ADDR[18:17] pins are also used. See tables [Table 3-22 on page 3-53](#) to [Table 3-25 on page 3-57](#) for bank address usage.

SDRAM t_{WR} parameter setting. SDTWR bits 18–17. Defines the required delay, in number of SDCLK cycles, between the time the SDC issues a write command (drives write data) and a precharge command.

$$SDTWR \geq \frac{t_{WRmin}}{t_{SDCLK}}$$

The t_{WR} parameter enables applications to accommodate the SDRAM’s timing requirements. [For more information, see “Timing External Memory Accesses” on page 3-36.](#) Any value between 1 and 3 SDCLK cycles may be selected as shown in [Table 3-17](#).

Table 3-17. SDRAM t_{WR} Bit Settings

SDTWR Bit Setting	SDRAM Parameter Setting
00	Reserved
SDTWR1 = 01	One clock cycle
SDTWR2 = 10	Two clock cycles
SDTWR3 = 11	Three clock cycles

SDRAM optional refresh. SDORF bit 19. Used for memories built as SDRAM in FPGAs. When set (=1), auto-refresh is not performed and the Force AR bit does not have any effect. When cleared (=0), auto-refresh occurs when the refresh counter expires. See also [“SDRAM Refresh Rate Control Register \(SDRRC\)” on page 3-49.](#)

Force auto-refresh. Force AR bit 20. When set (=1), forces auto-refresh. When cleared (=0), has no effect. Note that when SDORF bit is set, setting this bit causes Force AR to have no effect.

Force precharge. Force PC bit 21. When set (=1), forces precharge. When cleared (=0), has no effect.

Force load mode register. Force LMR bit 22. If set (=1), when SDRAMs are in a precharged state, a mode register write to SDRAMs is initiated immediately. This is in contrast to the normal load mode register set which requires some delay. This command performs a precharge all (if not precharged already) followed by a mode register write (ADSP-2137x processors only).

External register buffer pipeline option. SDBUF bit 23. Enables, if set (=1), or disables, if cleared (=0), external buffer timing. When buffered SDRAM modules or discrete register buffers are used to drive the SDRAM control inputs, SDBUF should be set to 1. This adds a cycle of data buffering to read and write accesses. An example single processor system is shown in [Figure 3-7](#).

When no buffering is required, the example shown in [Figure 3-6](#) can be used.

SDRAM tRCD parameter setting. SDTRCD bits 26–24. Sets the required delay (in terms of SDCLK) between a bank activate command and the start of the first read or write command.

$$SDRCD = \frac{t_{RCDmin}}{t_{SDCLK}}$$

SDRAM Controller

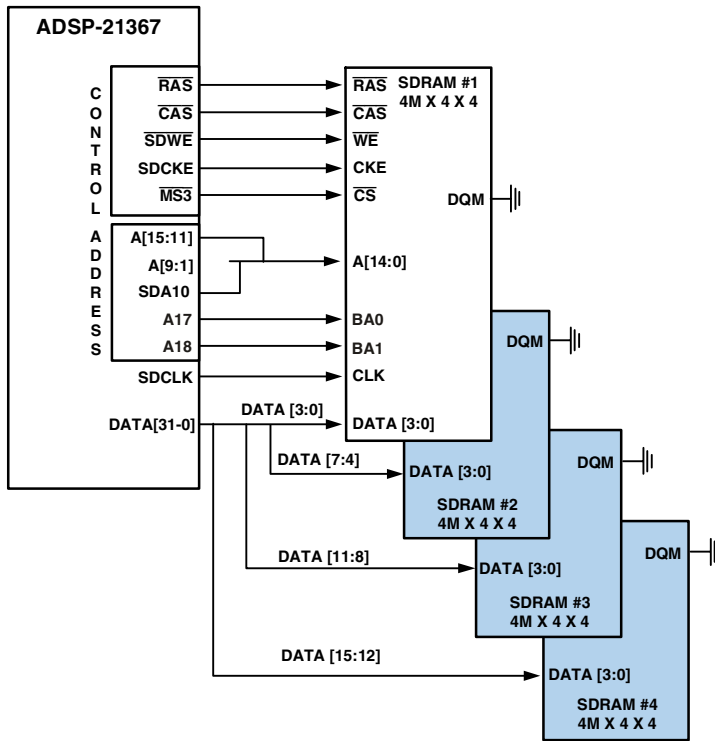


Figure 3-6. Uniprocessor System With Multiple SDRAM Devices

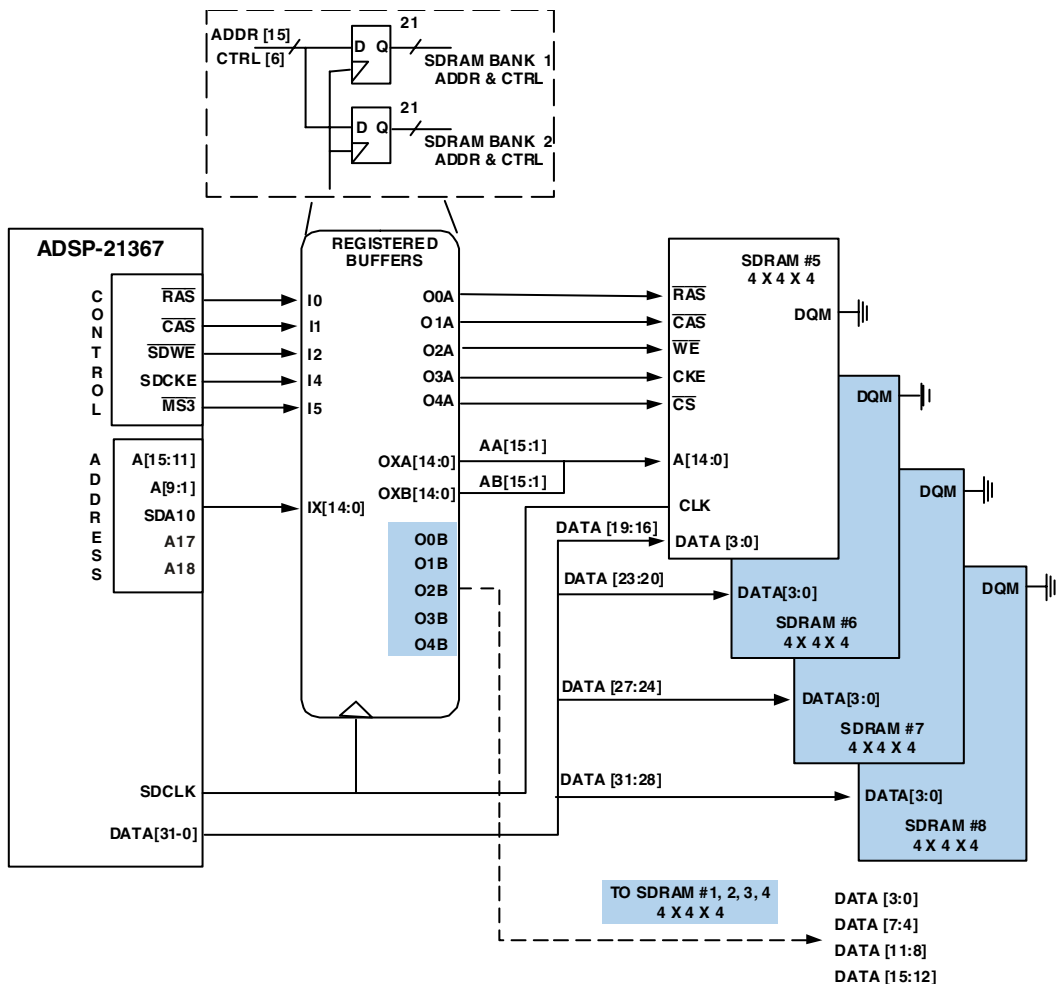


Figure 3-7. Uniprocessor System With Multiple Buffered SDRAM Devices

SDRAM Controller

For more information, see “Timing External Memory Accesses” on page 3-36. Any value between 1 and 7 $SDCLK$ cycles may be selected as shown in Table 3-18.

Table 3-18. SDRAM t_{RCD} Bit Settings

SDTRCD Bit Setting	SDRAM Parameter Setting	SDTWR Bit Setting	SDRAM Parameter Setting
000	Reserved	STDRC4 = 100	Four clock cycles
STDRC1 = 001	One clock cycle	STDRC5 = 101	Five clock cycles
STDRC2 = 010	Two clock cycles	STDRC6 = 110	Six clock cycles
STDRC3 = 011	Three clock cycles	STDRC7 = 111	Seven clock cycles

SDRAM row address width. $SDRAW$ bits 29–27. With the $X16DE$ and $SDCAW$ bits, defines the SDRAM core memory space (internal address to external address mapping). Any value between 0 to 7 can be selected as shown in Table 3-19. For more information, see “SDRAM Address Mapping” on page 3-51.

Program the SDRAM Controller for Page Size of 128 Words. $PGSZ$ 128 bit 30. This bit allows programs to configure the SDC for a page size of 128 words (7 bits) which supports most available 32 Mb SDRAMs.

No burst mode. NO_BSTOP bit 31. This bit is used to select between full page burst or no burst mode ($BL=1$). If set ($=1$), no burst mode is active and the burst stop command is ignored. If cleared, full page burst is active using the burst stop command for access interruption. This bit must be cleared if the SDRAM does not support no burst mode but supports full page burst.

Table 3-19. SDRAM Row Address Width Bit Settings

SDRAW Bit Setting	Row Address Width	SDRAW Bit Setting	Row Address Width
SDRAW8 = 000	8 bits (256)	SDRAW12 = 100	12 bits (4K)
SDRAW9 = 001	9 bits (512)	SDRAW13 = 101	13 bits (8K)
SDRAW10 = 010	10 bits (1K)	SDRAW14 = 110	14 bits (16K)
SDRAW11 = 011	11 bits (2K)	SDRAW15 = 111	15 bits (32K)

SDRAM Control Status Register (SDSTAT)

The SDRAM control status register provides information on the state of the SDC. This information can be used to determine when it is safe to alter SDC control parameters, or as a debug aid. The status bits that appear in this register are described in detail in [“SDRAM Control Status Register \(SDSTAT\)” on page A-26](#).

SDRAM Refresh Rate Control Register (SDRRC)

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

The delay (in number of $SDCLK$ cycles) between consecutive refresh counter time-outs must be written to the $RDIV$ field. A refresh counter time-out triggers an auto-refresh command to the external SDRAM bank. Programs should write the $RDIV$ value to the $SDRRC$ register before the SDRAM power-up sequence is triggered. Change this value only when the SDC is idle as indicated in the $SDSTAT$ register.

SDRAM Controller

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

$$RDIV \leq \left(\frac{f_{SDCLK} \times t_{REF}}{NRA} \right) - (t_{RAS} + t_{RP})$$

Where:

f_{SDCLK} = `SDCLK` frequency (SDRAM clock frequency)

t_{REF} = SDRAM refresh period

NRA = Number of row addresses in SDRAM (refresh cycles to refresh whole SDRAM)

t_{RAS} = Active to precharge time (`SDTRAS` bits in the SDRAM memory control register) in number of clock cycles

t_{RP} = RAS to precharge time (in the SDRAM memory control register) in number of clock cycles

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

$f_{SDCLK} = 133 \text{ MHz}$

$t_{REF} = 64 \text{ ms}$

$NRA = 8192 \text{ row addresses}$

$t_{RAS} = 6$

$t_{RP} = 3$

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

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

- i** The `RDIV` value must be programmed to a nonzero value if the SDRAM controller is enabled. When `RDIV` = 0, operation of the SDRAM controller is not supported and can produce undesirable behavior. Values for `RDIV` can range from 0x001 to 0xFFFF.

For details on the `SDROPT` and `SDMODIFY` bits see, [“SDRAM Read Optimization” on page 3-75](#).

SDRAM Initialization

Before executing the SDC power-up sequence, ensure that:

1. The SDRAM receives stable power and is clocked for the proper amount of time, as specified by the SDRAM specification.
2. The `SDPSS` bit is set to 1 to enable the SDRAM power-up sequence.
3. A read or write access occurs to enabled SDRAM address space in order to have the external bus granted to the SDC. This allows the SDRAM power-up sequence to occur.

- i** There is a long latency for this first access to SDRAM because the power-up sequence takes many cycles to complete.

SDRAM Address Mapping

The address that is seen from the processor core and DMA controller is referred to as internal address space `IA[31–0]` in the following sections. The internal address is divided into three parts to generate the SDRAM row, column, and bank addresses as shown in [Figure 3-8](#).

On the ADSP-21367/8/9 and ADSP-2137x processors, bank 0 starts at address 0x20 0000 in external memory and is followed in order by banks 1, 2, and 3. When the processor generates an address located within one of the four banks, it asserts the corresponding memory select line, $\overline{MS}3-0$.

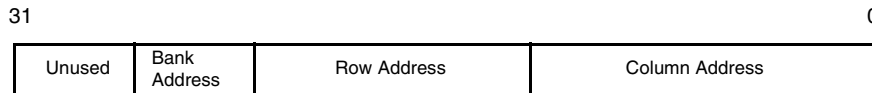


Figure 3-8. Core Address Mapping to Bank, Row, and Column Addresses

The external memory address ranges are shown in [Table 3-20](#).

i External memory address space is supported in normal word addressing mode only. Single-instruction, multiple-data (SIMD), extended-precision, short word, and long word addressing modes are not supported. Program execution from external memory is also not supported in the ADSP-21367/8/9 processors but is supported in the ADSP-2137x processors. For more information, [“External Memory Interface on the ADSP-2137x Processors” on page 3-3](#).

Table 3-20. External Memory Address Space for SDRAM Addresses

Bank	Size in Words	Address Range
Bank 0	62M	0x0020 0000 – 0x03FF FFFF
Bank 1	64M	0x0400 0000 – 0x07FF FFFF
Bank 2	64M	0x0800 0000 – 0x0BFF FFFF
Bank 3	64M	0x0C00 0000 – 0x0FFF FFFF

The $\overline{MS}3-0$ outputs serve as chip selects for memories or other external devices, eliminating the need for external decoding logic. For more information, see [“Timing External Memory Accesses” on page 3-36](#). The $\overline{MS}3-0$

lines are decoded memory address lines that change at the same time as the other address lines. When no external memory access is occurring, the $\overline{\text{MS3-0}}$ lines are inactive.

The width of the bank address is only two bits and is shown in [Table 3-21](#). The width of the column address is programmable. The row address is also programmable using SDRAW bits. The SDRAM bank address is calculated using the row address width and the column address width.

Table 3-21. External Memory Address Bank Decoding

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

In the following sections and in [Table 3-22](#) through [Table 3-25](#), the mapping of internal addresses to the external addresses is discussed. The mapping of the addresses depends on the row address width (SDRAW), column address width (SDCAW), and the X16DE bit setting.

In [Table 3-22](#), $\text{X16DE} = 0$, $\text{SDRAW}[2:0] = 100$ (12 bits), and $\text{SDCAW}[1:0] = 10$ (10 bits).

Table 3-22. 32-Bit Column, Row, and Bank Address Mapping (1K Words)

Pin	Column Address	Row Address	Bank Address	Pins of SDRAM
A[18]			IA[23]	BA[1]
A[17]			IA[22]	BA[0]
A[13]				A[12]
A[12]		IA[21]		A[11]

SDRAM Controller

Table 3-22. 32-Bit Column, Row, and Bank Address Mapping (1K Words) (Cont'd)

Pin	Column Address	Row Address	Bank Address	Pins of SDRAM
SDA10		IA[20]		A[10]
A[10]	IA[9]	IA[19]		A[9]
A[9]	IA[8]	IA[18]		A[8]
A[8]	IA[7]	IA[17]		A[7]
A[7]	IA[6]	IA[16]		A[6]
A[6]	IA[5]	IA[15]		A[5]
A[5]	IA[4]	IA[14]		A[4]
A[4]	IA[3]	IA[13]		A[3]
A[3]	IA[2]	IA[12]		A[2]
A[2]	IA[1]	IA[11]		A[1]
A[1]	IA[0]	IA[10]		A[0]
A[0]	Not USED for 32-bit SDRAMs			

In [Table 3-23](#), $X_{16DE} = 0$, $SDRAW[2:0] = id\ 100$ (12 bits), and $SDCAW[1:0] = 11$ (11 bits).

Table 3-23. 32-Bit Column, Row and Bank Address Mapping (2K Words)

Pin	Column Address	Row Address	Bank Address	Pins of SDRAM
A[18]			IA[24]	BA[1]
A[17]			IA[23]	BA[0]
A[13]				A[12]
A[12]	IA[10]	IA[22]		A[11]
SDA10		IA[21]		A[10]
A[10]	IA[9]	IA[20]		A[9]

Table 3-23. 32-Bit Column, Row and Bank Address Mapping
(2K Words) (Cont'd)

Pin	Column Address	Row Address	Bank Address	Pins of SDRAM
A[9]	IA[8]	IA[19]		A[8]
A[8]	IA[7]	IA[18]		A[7]
A[7]	IA[6]	IA[17]		A[6]
A[6]	IA[5]	IA[16]		A[5]
A[5]	IA[4]	IA[15]		A[4]
A[4]	IA[3]	IA[14]		A[3]
A[3]	IA[2]	IA[13]		A[2]
A[2]	IA[1]	IA[12]		A[1]
A[1]	IA[0]	IA[11]		A[0]
A[0]	Not USED for 32-bit SDRAMs			

Even if the external data width is 16 bits, the ADSP-21367/8/9 and ADSP-2137x SHARC processors support only 32-bit data accesses. If $X16DE$ is enabled ($=1$) the SDC performs two 16-bit accesses to get and place 32-bit data. The SDC takes the IA address and appends one extra bit to the LSB to generate the address externally.

For example, if the processor core requests address 0x200–0000 for a 32-bit access, the SDC performs two 16-bit accesses at 0x000–0000 and 0x000–0001, using $\overline{MS0}$ to get one 32-bit data word. The column and row addresses seen by 16-bit SDRAMs is shown in [Table 3-24](#) where $X16DE = 1$, $SDRAW[2:0] = 100$ (12 bits), and $SDCAW[1:0] = 10$ (10 bits) and [Table 3-25](#) where $X16DE = 1$, $SDRAW[2:0] = 100$ (12 bits), and $SDCAW[1:0] = 11$ (11 bits).

Table 3-24. 16-Bit Row and Column Address Mapping
(1K Words)

Pin	Column Address	Row Address	Bank Address	Pins of SDRAM
A[18]			IA[22]	BA[1]
A[17]			IA[21]	BA[0]
A[13]				
A[12]				A[12]
A[11]		IA[20]		A[11]
SDA10		IA[19]		A[10]
A[9]	IA[8]	IA[18]		A[9]
A[8]	IA[7]	IA[17]		A[8]
A[7]	IA[6]	IA[16]		A[7]
A[6]	IA[5]	IA[15]		A[6]
A[5]	IA[4]	IA[14]		A[5]
A[4]	IA[3]	IA[13]		A[4]
A[3]	IA[2]	IA[12]		A[3]
A[2]	IA[1]	IA[11]		A[2]
A[1]	IA[0]	IA[10]		A[1]
A[0]	1/0	IA[9]		A[0]

Table 3-25. 16-Bit Row and Column Address Mapping (2K Words)

Pin	Column Address	Row Address	Bank Address	Pins of SDRAM
A[18]			IA[23]	BA[1]
A[17]			IA[22]	BA[0]
A[13]				
A[12]				A[12]
A[11]	IA[9]	IA[21]		A[11]
SDA10		IA[20]		A[10]
A[9]	IA[8]	IA[19]		A[9]
A[8]	IA[7]	IA[18]		A[8]
A[7]	IA[6]	IA[17]		A[7]
A[6]	IA[5]	IA[16]		A[6]
A[5]	IA[4]	IA[15]		A[5]
A[4]	IA[3]	IA[14]		A[4]
A[3]	IA[2]	IA[13]		A[3]
A[2]	IA[1]	IA[12]		A[2]
A[1]	IA[0]	IA[11]		A[1]
A[0]	1/0	IA[10]		A[0]



If you are using 32-bit SDRAMs, the SDC on the ADSP-21367/8/9 and ADSP-2137x processors A0 pin is not connected to the SDRAM's A0 pin.

SDRAM Controller Address Mapping

To access SDRAM, the SDC multiplexes the internal 32-bit, non-multiplexed address into a row and column address. The row and column address mappings for 32-bit and 16-bit addresses are shown in [Table 3-25](#). The row and column addresses are muxed to pins A14-A0 of the processor. The SDRAM address pin A10 is connected to the processor's SDA10 pin. The SDC bank address pins BA[0] and BA[1], are connected to the processor's A[17] and A[18] pins.

 For 2 banked SDRAMs connect BA with A[17].

SDC Operation

The AMI normally generates an external memory address, which then asserts the corresponding \overline{CS} select on the SDRAM, along with \overline{RD} and \overline{WR} strobes. However these control signals are not used by the SDRAM controller. The internal strobes are used to generate pulsed commands (\overline{MSx} , \overline{SDCKE} , \overline{SDRAS} , \overline{SDCAS} , \overline{SDWE}) within a truth table [Table 3-26](#). The memory access to SDRAM is based by mapping ADDR[27:0] causing an internal memory select to SDRAM space.

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

A programmable refresh counter is provided which generates background auto-refresh cycles at the required refresh rate based on the clock frequency used. The refresh counter period is specified with the SDRRC field in the SDRAM refresh rate control register ("[SPERRCTLx Register](#)" on [page A-48](#)).

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

- SDRAM column address
- SDRAM row address
- Internal SDRAM bank address

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

The SDC uses no burst mode ($BL = 1$) for read and write operations. This requires the SDC to post every read or write address on the bus as for non-sequential reads or writes, but does not cause any performance degradation. For ADSP-2137x processors, optional full page burst can be activated. However, every single access is immediately interrupted by another access resulting in no burst mode.

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

For more information on commands used by the SDRAM controller, see [“SDC Commands” on page 3-63](#).

Single Bank Operation

The SDC keeps only one page open at a time, however, driving four external memory selects populated with SDRAM, the effective page size is increased up to four pages.

Multibank Operation (ADSP-2137x Processors)

Since an SDRAM contains four independent internal banks (A–D), the SDC on the ADSP-2137x processors is capable of supporting multibank operation, thus taking advantage of the architecture.

Any first access to SDRAM bank (A) forces an activate command before a read or write command. However, if any new access falls into the address space of the other banks (B, C, or D) the SDC leaves bank (A) open and activates any of the other banks (B, C, or D). Bank (A) to bank (B) active time is controlled by $t_{RRD} = t_{RCD} + 1$. This scenario is repeated until all four banks (A–D) are opened and results in an effective page size of up to four pages. This is because the absence of latency allows switching between these open pages (as compared to one page in only one bank at a time). Any access to any closed page in any opened bank (A–D) forces a precharge command only to that bank. If, for example, two external port DMA channels are pointing to the same internal SDRAM bank, this always forces precharge and activation cycles to switch between the different pages. However, if the two external port DMA channels are pointing to different internal SDRAM banks, there is no additional overhead. See [Figure 3-9](#).

Furthermore the SDC supports four external memory selects containing each SDRAM. However only the $\overline{MS0}$ and $\overline{MS1}$ signals provide multibank support, so the maximum number of open pages is $2 \times 4 + 2 \times 1 = 10$ pages.



Multibank operation reduces precharge and activation cycles by mapping opcode/data among different internal SDRAM banks driven by the A[18:17] pins and external memory selects (\overline{MSX}).

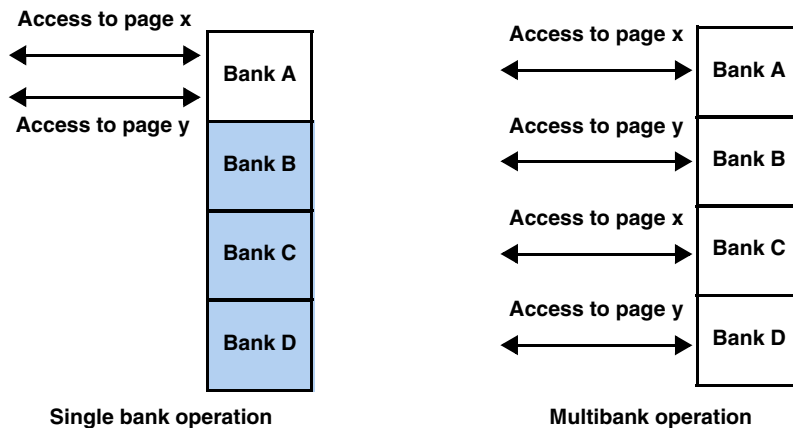


Figure 3-9. Single Versus Multibank Operation

Data Mask (DQM)

Since the ADSP-21367/8/9 and ADSP-2137x processors do not support byte addressing, there is no need to mask data during partial writes (for example, higher or lower byte on a 16-bit wide SDRAM).

i All SDRAM DQM pins must be tied low.

SDC Configuration

After reset, the SDC clocks are enabled. However, the SDC must be configured and initialized. Before programming the SDC and executing the power-up sequence, select the correct `CCLK` to `SDCLK` ratio, and ensure that the clock to the SDRAM is enabled (after the power has stabilized for the proper amount of time as specified by the SDRAM).

SDRAM Controller

In order to set up the SDC and start the SDRAM power-up sequence for the SDRAMs, use the following procedure. Note that the registers must be programmed in order.

1. External port control (EPCTL) register (assign external banks to SDC)
2. Refresh rate control (SDRRRC) register (program refresh counter)
3. SDRAM control (SDCTL) register (define global control for SDC and SDRAM based on speed and SDRAM specs)
4. Access to SDRAM address space (initiates power-up sequence)

The `SDRS` bit (bit 3) of the SDRAM control status register can be checked to determine the current state of the SDC. If this bit is set, the SDRAM power-up sequence has not been initiated.

In order to set up the SDC and start the SDRAM power-up sequence for the SDRAMs, the SDRAM refresh rate control register (SDRRRC) and SDRAM memory control register (SDCTL) must be written, and a transfer must be started to SDRAM address space. The `SDRS` bit of the SDRAM control status register can be checked to determine the current state of the SDC. If this bit is set, the SDRAM power-up sequence has not been initiated.

The `RDIV` field of the `SDRRRC` register is written to set the SDRAM refresh rate.

Write to the `SDCTL` register in order to:

- Set the SDRAM cycle timing options—`SDCL`, `SDTRAS`, `SDTRP`, `SDTRCD`, `SDTWR`, `SDBUF`
- Enable the SDRAM clock (`DSDCTL`)
- Set the data path width (`X16DE`)

- Select and enable the start of the SDRAM power-up sequence (SDPM, SDPSS)
- Select the column/row address widths

Once the SDPSS bit in the SDCTL register is set to 1, and a transfer occurs to enabled SDRAM address space, the SDC initiates the SDRAM power-up sequence. The exact sequence is determined by the SDPM bit in the SDCTL register. The transfer that is used to trigger the SDRAM power-up sequence can be either a read or a write. This transfer occurs when the SDRAM power-up sequence has completed. This initial transfer takes many cycles to complete since the SDRAM power-up sequence must take place.

SDC Commands

This section provides a description of each of the commands that the SDC uses to manage the SDRAM interface. These commands are handled automatically by the SDC. A summary of the various commands used by the on-chip controller for the SDRAM interface follows and is shown in [Table 3-26 on page 3-72](#).

- Load mode register—initializes the SDRAM operation parameters during the power-up sequence.
- Single precharge—closes a specific internal bank depending on user code (ADSP-2137x processors only).
- Precharge all—closes all internal banks, preceding any auto-refresh command.
- Activate—activates a page in the required internal SDRAM bank
- Read/write
- Auto-refresh—causes the SDRAM to execute an internal CAS before RAS refresh.

SDRAM Controller

- Self-refresh entry—places the SDRAM in self-refresh mode, in which the SDRAM powers down and controls its refresh operations internally.
- Self-refresh exit—exits from self-refresh mode by expecting auto-refresh commands from SDC.
- NOP/command inhibit—no operation used to insert wait states for activate and precharge cycles
- Burst Stop command—used to interrupt any full page burst operation (ADSP-2137x processors only).

Load Mode Register

This command initializes SDRAM operation parameters. It is a part of the SDRAM power-up sequence. Load mode register uses the address bus of the SDRAM as data input. The power-up sequence is initiated by writing 1 to the `SDPSS` bit in the `SDCTL` register and then writing or reading from any enabled address within the SDRAM address space to trigger the power-up sequence. The exact order of the power-up sequence is determined by the `SDPM` bit of the `SDCTL` register.

The load mode register command initializes the following parameters.


- Burst length = 1, bits 2–0, always zero
- Optional burst length = full page, bits 2–0, all ones (ADSP-2137x processors only).
- Wrap type = sequential, bit 3, always zero
- Ltmode = latency mode (CAS latency), bits 6–4, programmable in the `SDCTL` register
- Bits 14–7, always zero

While executing the load mode register command, the unused address pins are set to zero. During the first `SDCLK` cycle following load mode register, the SDC issues only `NOP` commands.

Alternatively, programs can use the Force LMR command by setting bit 22 (=1) in the `SDCTL` register. This command performs precharge all (if not precharged already) followed by a mode register write. Unlike the standard load mode register command, the eight CBR commands are not performed. The correct usage of this bit is:

1. Force precharge (set bit 21)
2. Wait
3. Force LMR (set bit 22)
4. Wait
5. Eight CBR refresh cycles (set bit 20 eight times)

The order of these commands (step 3 and step 5, depending on SDRAM) can be changed depending on the SDRAM requirements (see your SDRAM vendors data sheet).

 When the Force LMR bit is set, the load mode register command is performed immediately. This is in contrast to the normal load mode register command which requires a dummy access to be executed (ADSP-2137x processors only).

Single Bank Activation

The bank activation command is required for first access to any internal bank in SDRAM. Any subsequent access to the same internal bank but different row will be preceded by a precharge and activation command to that bank.

However, if an access to another bank occurs, the SDC closes the current page open and issues another bank activate command before executing the read or write command to that bank. With this method, called single bank operation, Only one page can be open at a time.

Multibank Activation (ADSP-2137x Processors)

Unlike this command for the ADSP-21367/8/9 processors, if any other access to another bank occurs, the SDC leaves the current page open and issues a bank activate command before executing the read or write command to that bank. With this method, called multibank operation, one page per bank can be open at a time, which results in a maximum of four pages. [For more information, see “Multibank Operation \(ADSP-2137x Processors\)” on page 3-60.](#)



Multibank activation is only supported for the external banks 0 and 1.

Single Precharge (ADSP-2137x Processors)

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



This command is only supported for the external banks 0 and 1.

Precharge All

The precharge all command is given to precharge all internal banks at the same time before executing an auto-refresh. All open banks are automatically closed. This is possible since the SDC uses a separate SDA10 pin which is asserted high during this command. This command proceeds the auto-refresh command. Also, for single bank operation, this command is used to close any open bank after a page miss detection.

Read/Write

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

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

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

Figure 3-10 and Figure 3-11 show the SDRAM write and read timing of the ADSP-21367/8/9 processors respectively. Figure 3-12 and Figure 3-13 show the SDRAM write and read timing for the ADSP-2137x processors.

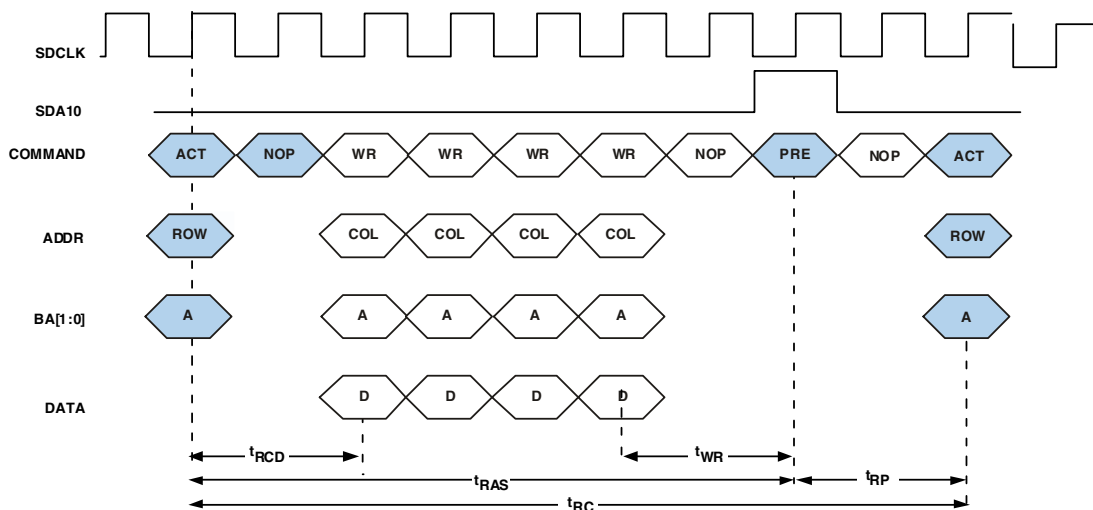


Figure 3-10. Write Timing Diagram ADSP-21367/8/9

SDRAM Controller

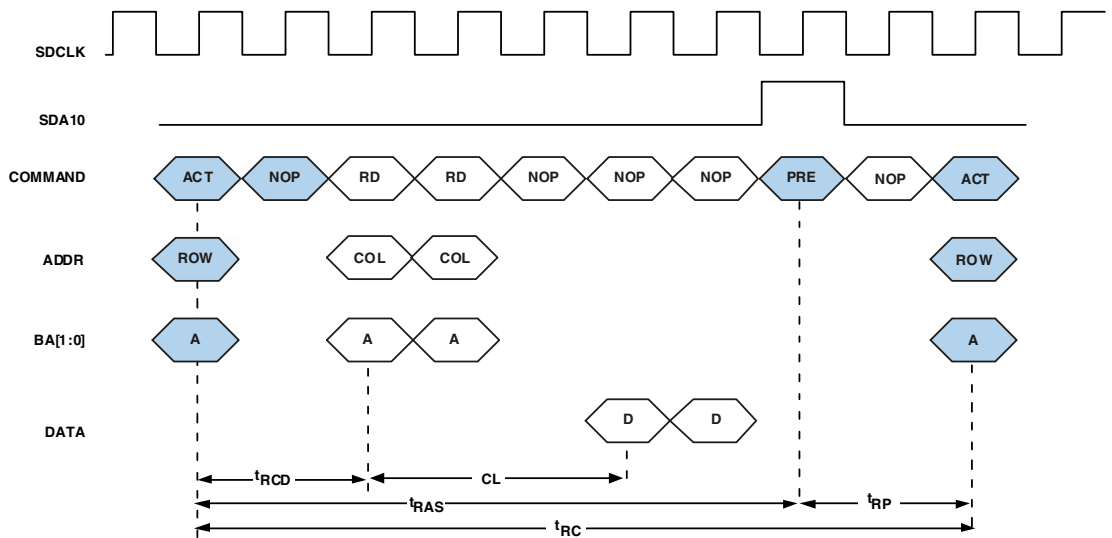


Figure 3-11. Read Timing Diagram ADSP-21367/8/9

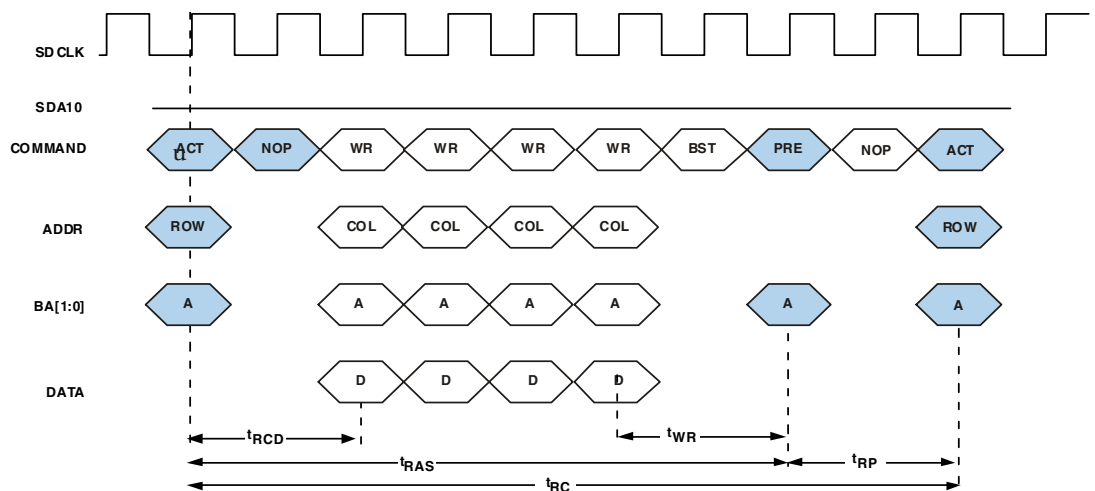


Figure 3-12. Write Timing Diagram (ADSP-2137x, Full Page Burst)

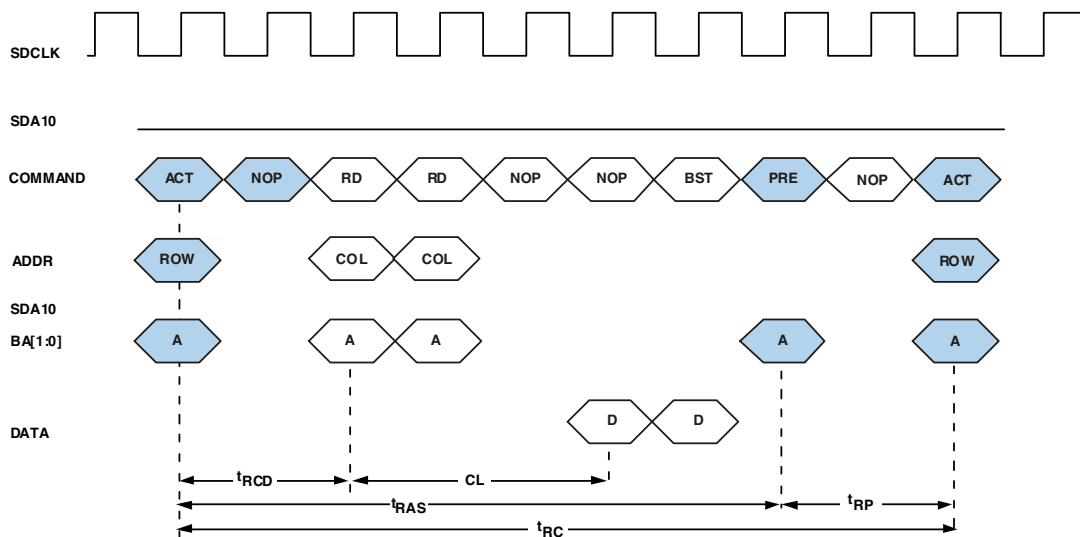


Figure 3-13. Read Timing Diagram (ADSP-2137x, Full Page Burst)

Read/Write (ADSP-2137x Processors)

If the optional full page burst is select in the `SDCTL` register, the SDC posts for every read and write an address on the bus. It does not burst, this causes that every start address of the burst is interrupted with another start address. This mode is equivalent to no burst mode.

Burst Stop (ADSP-2137x Processors)

If the optional full page burst is selected in the `SDCTL` register, the SDC posts an address on the bus for every read and write. However if a non-SDRAM access is latched, the SDC interrupts the full page burst protocol. By executing a burst stop command, the specific page remains open.

Auto-Refresh

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

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

Self-Refresh Mode

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

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

Self-refresh exit—When any SDRAM access occurs, the SDC asserts SDCKE high which causes the SDRAM to exit from self-refresh mode. The SDC waits to meet the t_{XSR} specification ($t_{XSR} = t_{RAS} + t_{RP}$) and then issues an auto-refresh command. After the auto-refresh command, the SDC waits for the t_{RFC} specification ($t_{RFC} = t_{RAS} + t_{RP}$) to be met before executing the activate command for the transfer that caused the SDRAM

to exit self-refresh mode. Therefore, the latency from when a transfer is received by the SDC while in self-refresh mode, until the activate command occurs for that transfer, is $2 \times (t_{RC} + t_{RP})$ cycles

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

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

1. Set the $SDSRF$ bit to enter self-refresh mode
2. Poll the $SDSRA$ bit in the SDRAM status register ($SDSTAT$) to determine if the SDRAM has already entered self-refresh mode.
3. Optionally: set the $DSDCTL$ bit to freeze $SDCLK$
4. Optionally: clear the $DSDCTL$ bit to re-enable $SDCLK$
5. SDRAM access occurs the SDRAM exits from self-refresh mode



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

No Operation/Command Inhibit

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

SDRAM Controller

commands with no effect, the NOP command is given. When the SDC is not accessing any SDRAM external banks, the command inhibit command is given.

A summary of pin states during SDC commands appears in [Table 3-26](#). Note that an X means do not care.


Table 3-26. SDRAM Pin States During SDC Commands

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

Changing System Clock During Runtime

All timing specifications are normalized to the system clock. Since most of these are minimum specifications, (except t_{REF} , which is a maximum specification), a variation of the system clock violates a specific specification and causes a performance degradation for the other specifications.

The reduction of system clock violates the minimum specifications, while increasing the system clock violates the maximum t_{REF} specification. Therefore, careful software control is required to adapt these changes.

 For most applications, the SDRAM power-up sequence and writing of the mode register needs to occur only once. Once the power-up sequence has completed, the `SDPSS` bit should not be set again unless a change to the mode register is desired.

The recommended procedure for changing the system frequency `SDCLK` is as follows.

1. Set the SDRAM to self-refresh mode by writing a 1 to the `SDSRF` bit of `SDCTL` register.
2. Poll the `SDSRA` bit of `SDSTAT` register for self-refresh grant.
3. Execute the desired PLL programming sequence. (For details see [“PLL Programming Examples” on page 14-16](#)).
4. Wait until the signal `RESETOUT/CLKOUT` is asserted which ensures that the PLL has settled to the new frequency.
5. Reprogram the SDRAM registers (`SDRRR`, `SDCTL`) with values appropriate to the new `SDCLK` frequency and assure that the `SDSRF` bit is set.
6. Bring the SDRAM out of self-refresh mode by performing a read or write access.

SDRAM Timing

To support key timing requirements and power-up sequences for different SDRAM vendors, the SDC provides programmability for t_{RAS} , t_{RP} , t_{RCD} , t_{WR} and the power-up sequence mode.

CAS latency is programmed in the $SDCTL$ register based on the frequency of operation. (Please refer to the SDRAM vendor's data sheet for more information.)

For other parameters, the SDC assumes:

- Bank cycle time is $t_{RC} = t_{RAS} + t_{RP}$
- Bank A to Bank B cycle time is $t_{RRD} = t_{RCD} + 1$ (ADSP-2137x processors only)
- Refresh cycle time is $t_{RFC} = t_{RAS} + t_{RP}$
- Exit self-refresh time is $t_{XSR} = t_{RAS} + t_{RP}$
- Load mode register to activate time is $t_{MRD} = 2 \text{ } SDCLK \text{ cycles}$.

Table 3-27 and Table 3-28 show the optimal data throughput for 32- and 16-bit data accesses respectively. Table 3-29 shows accesses between external memory banks.

Table 3-27. Optimal Data Throughput for 32-Bit Data Accesses
(CAS Latency = 2)

Access	Operation	Page	Throughput per SDCLK (32-Bit Data)
Sequential and uninterrupted	Read	Same	32 words per 37 cycles
Any	Write	Same	core = 1 word per cycle DMA = 1 word per 2 cycles
Non Sequential and uninterrupted	Read	Same	6 cycles

Table 3-28. Optimal Data Throughput for 16-Bit Data Accesses
(CAS Latency = 2)

Access	Operation	Page	Throughput per SDCLK (32-Bit Data)
Sequential and uninterrupted	Read	Same	32 words per 69 cycles
Sequential and uninterrupted	Write	Same	2 cycles
Non Sequential and uninterrupted	Read	Same	7 cycles

Table 3-29. Accesses Between External Memory Banks
(MSx, CAS Latency = 2)

Operation	Idle Cycles per SDCLK (32-Bit Data)
Read	6 cycles
Write	1 cycle



With external buffering enabled, each access takes one extra cycle.

SDRAM Read Optimization

To achieve better performance, read addresses can be provided in a predictive manner to the SDRAM memory. This is done by setting (=1) the `SDROPT` bit (bit 16) and correctly configuring the `SDMODIFY` bits (bits 20–17) in the `SDRRC` register. The predictive address given to the memory depends on the `SDMODIFY` bits values. If the `SDMODIFY` value is 2, then the address + 2 is the predictive value provided to the SDRAM address pins.

Programs may choose to determine whether read optimization is used or not. If read optimization is disabled, then each read takes 6 cycles for a CAS latency of 2. With read optimization enabled, 32 sequential reads, with offsets ranging from 0 to 15, take only 37 cycles. Read optimization should not be enabled while reading at the external bank boundaries. For

example, if `SDMODIFY = 1`, then 32 locations in the boundary of the external banks should not be used. These locations can be used without optimization enabled. If `SDMODIFY = 2` then 64 locations can not be used at the boundaries of the external bank (if it is fully populated).

Achieving Maximum Throughput Using Core Accesses

Any break of sequential reads of 32 accesses can cause a throughput loss due to a maximum of eight extra reads in 32-bit memories or four extra reads (eight 16-bit reads). [Listing 3-1](#) shows how to achieve maximum throughput using core accesses. Any cycle between consecutive reads to an SDRAM address results in nonsequential reads.

Listing 3-1. Maximum Throughput Using Sequential Reads

```
I0 = sdram_addr;
M0 = 1;    /* SET sdmodify to 1 */

Lcntr = 1024, do(PC,1) until lce;
R0 = R0 + R1, R0 = dm (I0, M0);
```

The examples in [Listing 3-2](#) show how read optimization can be used efficiently using core accesses. In [Listing 3-2](#), all reads are on the same page and it takes 1184 cycles to perform 1024 reads.

Listing 3-2. Sequential Reads With Read Optimization

```
I0 = sdram_addr;
M0 = 2;    /* SET sdmodify to 2 */

Lcntr = 1024, do(PC,1) until lce;
R0 = R0 + R1, R0 = dm (I0, M0);
```

In [Listing 3-3](#), reads are not performed sequentially and the read optimization reduces throughput.

Listing 3-3. Nonsequential Reads With Read Optimization

```

IO = sdram_addr;
MO = 2;      /* SET sdmodify to 2 */

Lcntr = 1024, do(PC,2) until lce;
R0 = R0 + R1, R0 = dm (IO, MO);
NOP;

```

Without read optimization, 1024 reads use 6144 processor cycles if all of the reads are on the same page. With read optimization, 1024 reads take 7168 cycles, due to the breaking of sequential reads.

SDRAM Read Optimization Restrictions

Incorrect data may be read from external memory in the presence of the following two types of sequences:

1. Any sequence of IOP register accesses (read or write) followed by an external memory read causes incorrect data to be read from external memory. For example:

```

r0=dm(RXSP3A);
r0=dm(ext_dest_seq_read);    /* r0 is filled with the
                               wrong data */

```

To avoid this error, separate the IOP and external memory access by adding a NOP or any other instruction which is NOT an IOP read/write or an external memory read.

2. In a dual-data move instruction, both accesses should not be to external memory.

```

r0=dm(IOP), r8=pm(Ext_Mem);

```

The workaround is to break up this instruction into separate instructions and use the workaround similar to case #1.

```
r0=dm(IOP);  
NOP;  
r8=pm(Ext_Mem);
```

External Memory Access Restrictions

The following restrictions should be noted when writing programs for the ADSP-21367/8/9 and ADSP-2137x processors.

1. The processor does not execute from external memory in SIMD mode and the LW mnemonic is not applicable to external memory.
2. In a dual-data move instruction, both accesses should not be to external memory.
3. Conditional accesses to external memory should not be based on any of the FLAG pin status.
4. If there is an aborted or interrupted conditional read to an external memory, the processor generates a spurious read. This is an issue for FIFO devices and not standard memories.
5. Any sequence of IOP register access (read or write) followed by an external memory read, causes incorrect data to be read from external memory. To workaround this restriction, separate the IOP and external memory access by adding a NOP instruction or any other instruction which is *not* either an IOP read/write, or an external memory read.

Shared Memory Interface

The ADSP-21368 processor supports connections to a common shared external memory of other ADSP-21368 processors. These connections create shared external bus processor systems. This support includes:

- Support for asynchronous memory and SDRAM
- Distributed, on-chip arbitration for the shared external bus
- Fixed and rotating priority bus arbitration
- Bus time-out logic
- Bus lock

Figure 3-14 illustrates a basic shared memory system. In a system with several processors sharing the external bus, any of the processors can become the bus master. The bus master has control of the bus, which consists of the DATA31-0 and ADDR23-0 pins and associated control lines.



In a shared memory system, programs should not reset the current bus master as this leads to system synchronization problems.

Shared Memory Bus Arbitration

Multiple processors can share the external bus with no additional arbitration logic as shown in Figure 3-14. Arbitration logic is included on chip to allow the connection of up to four ADSP-21368 processors.

The processor accomplishes bus arbitration through the $\overline{\text{BRI}}-4$ signals which arbitrate between multiple processors. The priority scheme for bus arbitration is determined by the RPBA pin setting. Table 3-30 defines the processor pins used in multiprocessing systems.

Shared Memory Interface

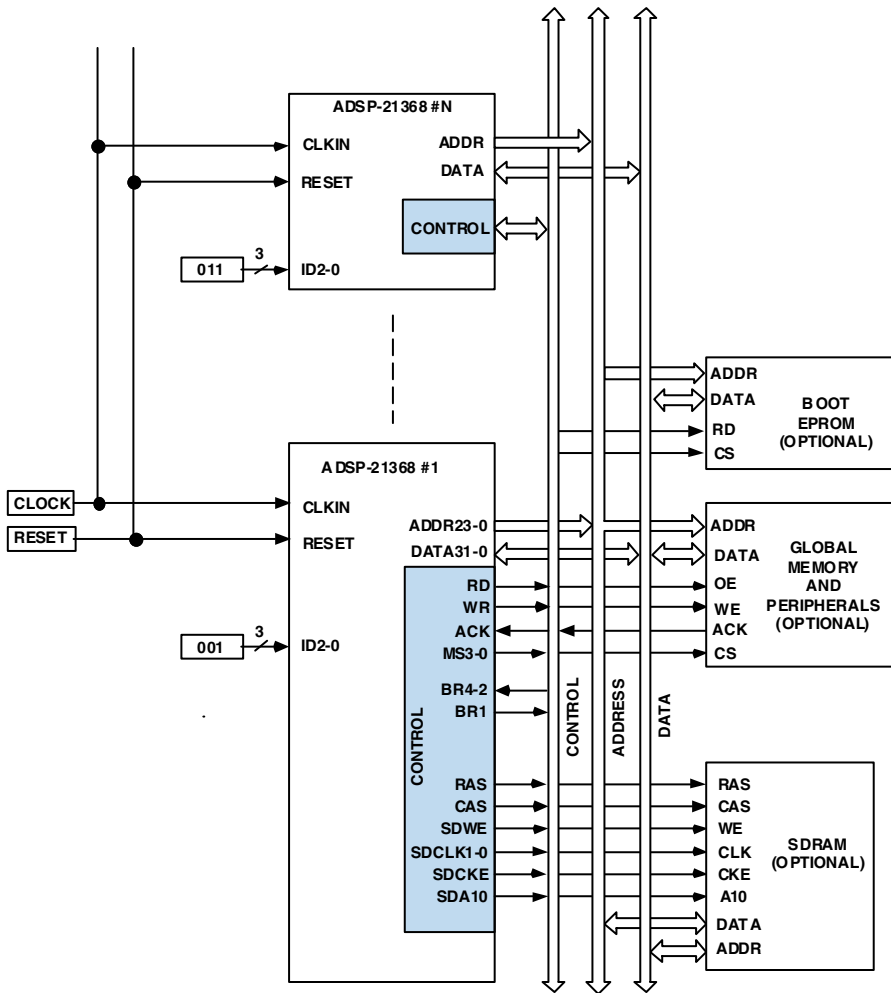


Figure 3-14. ADSP-21368 Shared Memory System

Table 3-30. Shared Memory Pins

Signal	Type	Definition
$\overline{\text{BR4-1}}$	I/O/S	Shared Memory Bus Requests. Used to arbitrate for bus mastership. A processor only drives its own $\overline{\text{BRx}}$ line (corresponding to the value of its ID2-0 inputs) and monitors all others. In a shared memory system with less than four processors, the unused $\overline{\text{BRx}}$ pins should be tied high; the processor's own $\overline{\text{BRx}}$ line must not be tied high or low because it is an output.
ID2-0	I	Shared Memory ID. Determines which bus request ($\overline{\text{BR1-4}}$) is used by the ADSP-21368. ID = 001 corresponds to $\overline{\text{BR1}}$, ID = 010 corresponds to $\overline{\text{BR2}}$, and so on. Use ID = 000 or ID = 001 in single processor systems. These lines are a system configuration selection that should be hardwired or only changed at reset.
RPBA	I	Rotating Priority Bus Arbitration Select. When RPBA is high, rotating priority for shared memory bus arbitration is selected. When RPBA is low, fixed priority is selected. This signal is a system configuration selection which must be set to the same value on every processor.
I = Input, S = Synchronous, O = Output		

The ID2-0 pins provide a unique identity for each processor in a multiprocessor system. The first processor should be assigned ID = 001, the second should be assigned ID = 010, and so on. One of the processors must be assigned ID = 001 in order for the bus synchronization scheme to function properly.



The processor with ID = 001 holds the external bus control lines stable (pull-up enabled) during reset.


A processor in a shared memory system can determine which processor is the current bus master by reading the CRBM2-0 bits of the SYSTAT register (see “[System Status Register \(SYSTAT\)](#)” on page A-9). These bits provide the values of the ID2-0 inputs of the current bus master.

Shared Memory Interface

Conditional instructions can be written that depend upon whether the processor is the current bus master in a shared memory system. The assembly language mnemonic for this condition code is `BM`, and its complement is `Not BM` (not bus master). The `BM` condition indicates whether the processor is the current bus master. For more information, see the “Conditional Sequencing” section in the *ADSP-2136x SHARC Processor Programming Reference*, “Program Sequencer” chapter. To use the bus master condition, the condition code select (`CSEL`) field in the `MODE1` register must be zero or the condition is always evaluated as false.

Bus Arbitration Protocol

The bus request ($\overline{\text{BR1-4}}$) pins are connected between each processor in a shared memory system, where the number of $\overline{\text{BRx}}$ lines used is equal to the number of processors in the system. Each processor drives the $\overline{\text{BRx}}$ pin that corresponds to its `ID2-0` inputs and monitors all others.

 If less than four processors are used in the system, the unused $\overline{\text{BRx}}$ pins should be tied high.

When one of the slave processors needs to become bus master, it automatically initiates the bus arbitration process by asserting its $\overline{\text{BRx}}$ line at the beginning of the cycle. Later in the same cycle, the processor samples the value of the other $\overline{\text{BRx}}$ lines.

The cycle in which mastership of the bus is passed from one processor to another is called a bus transition cycle (BTC). A BTC occurs when the current bus master's $\overline{\text{BRx}}$ pin is deasserted and one or more of the slave's $\overline{\text{BRx}}$ pins is asserted. The bus master can retain bus mastership by keeping its $\overline{\text{BRx}}$ pin asserted.

By observing all of the $\overline{\text{BRx}}$ lines, each processor can detect when a bus transition cycle occurs and which processor has become the new bus master. A bus transition cycle is the only time that bus mastership is transferred.

After conditions determine that a bus transition cycle is going to occur, every processor in the system evaluates the priority of the $\overline{\text{BRx}}$ lines asserted within that cycle. For a description of bus arbitration priority, see “[Bus Arbitration Priority \(RPBA\)](#)” on page 3-86. The processor with the highest priority request becomes the bus master on the following cycle, and all of the processors update their internal records to indicate which processor is the current bus master. [Figure 3-15](#) shows typical timing for bus arbitration.

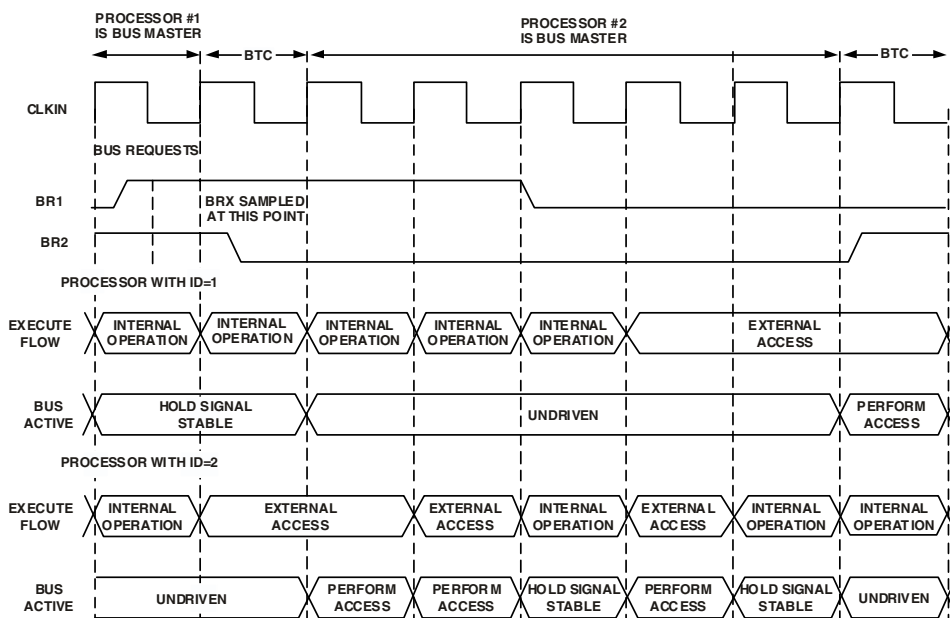


Figure 3-15. Bus Arbitration Timing

The actual transfer of bus mastership is accomplished by the current bus master three-stating the external bus— DATA31-0 , ADDR23-0 , $\overline{\text{RD}}$, $\overline{\text{WR}}$, and $\overline{\text{MS3-0}}$ —at the end of the bus transition cycle and the new bus master driving these signals at the beginning of the next cycle. The bus strobes $\overline{\text{RD}}$, $\overline{\text{WR}}$,

Shared Memory Interface

and $\overline{MS}3-0$ are driven high (inactive) before three-stating occurs. The ACK signal must be sampled high by the new master before it starts a new bus operation. For more information, see [Figure 3-16](#).

During bus transition cycle delays, execution of external accesses are delayed. When one of the slave processors needs to perform an external read or write, it automatically initiates the bus arbitration process by asserting its \overline{BRX} line. This read or write is delayed until the processor receives bus mastership. If the read or write was generated by the processor's core (not the I/O processor), program execution stops on that processor until the instruction is completed.

The following steps occur as a slave acquires bus mastership and performs an external read or write over the bus as shown in [Figure 3-16](#).

1. The slave determines that it is executing an instruction which requires an off-chip access. It asserts its \overline{BRX} line at the beginning of the cycle. Extra cycles are generated by the core processor (or I/O processor) until the slave acquires bus mastership.
2. To acquire bus mastership, the slave waits for a bus transition cycle in which the current bus master deasserts its \overline{BRX} line. If the slave has the highest priority request in the BTC, it becomes the bus master in the next cycle. If not, it continues waiting.
3. At the end of the BTC, the current bus master releases the bus and the new bus master starts driving.

During the $CLKIN$ cycle in which the bus master deasserts its \overline{BRX} output, it three-states its outputs in case another bus master wins arbitration and enables its drivers in the next $CLKIN$ cycle. If the current bus master retains control of the bus in the next cycle, it enables its bus drivers, even if it has no bus operation to run.

The processor with $ID = 001$ enables internal pull-up devices on key signals, including the address and data buses, strobes, and ACK . These devices provide a weak current source or sink (approximate 20 k Ω impedance) to

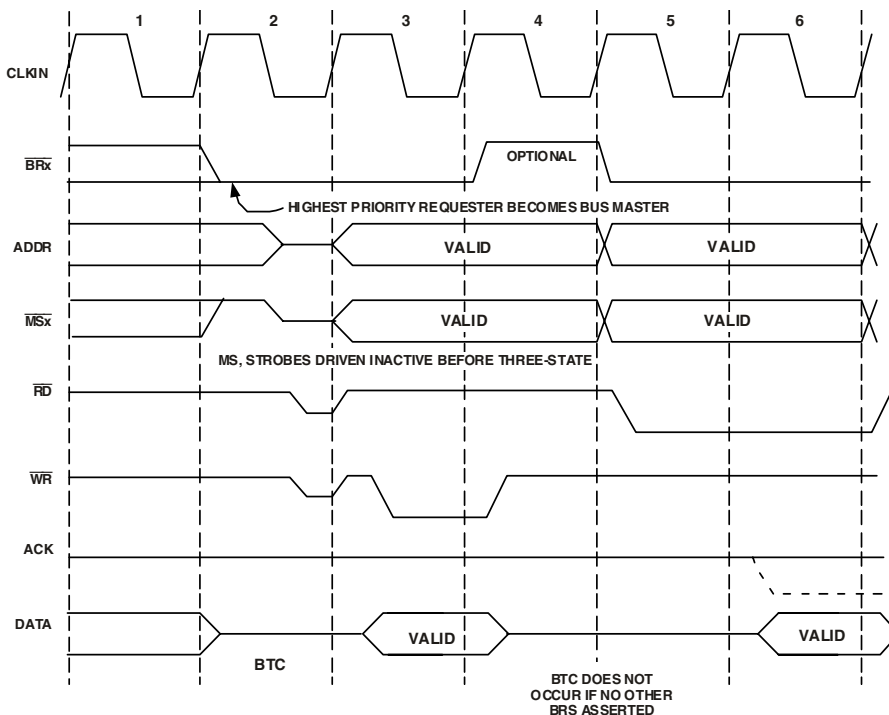


Figure 3-16. Bus Request and Read/Write Timing


keep these signals from drifting near input receiver thresholds when all drivers are three-stated. Note that single processor systems with $ID = 000$ also enable these pull-up devices.

When the bus master stops using the bus, its \overline{BRx} line is deasserted, allowing other processors to arbitrate for mastership if they need it. If no other processors are asserting their \overline{BRx} line when the master deasserts its \overline{BRx} , the master retains control of the bus and continues to drive the memory control signals until: 1) it needs to use the bus again, or 2) another processor asserts its \overline{BRx} line.

Bus Arbitration Priority (RPBA)

To resolve competing bus requests, there are two available priority schemes—fixed and rotating. The RPBA pin selects the scheme. When RPBA is high, rotating priority bus arbitration is selected, and when RPBA is low, fixed priority is selected. The RPBA pin must be set to the same value on each processor in a multiprocessing system.

In the fixed priority scheme, the processor with the lowest ID number among the competing bus requests becomes the bus master. If, for example, the processor with ID = 010 and the processor with ID = 100 request the bus simultaneously, the processor with ID=010 becomes bus master in the following cycle.

 Each processor knows the ID of the other processors requesting the bus, because the ID corresponds to the $\overline{\text{BRx}}$ line being used for each processor.

The rotating priority scheme gives roughly equal priority to each processor. When rotating priority is selected, the priority of each processor is reassigned after every transfer of bus mastership. Highest priority is rotated from processor to processor as if they were arranged in a circle—the processor with the next highest ID setting from the current bus master is the one that receives highest priority. Table 3-31 shows an example of how rotating priority changes on a cycle-by-cycle basis.

Table 3-31. Rotating Priority Arbitration Example

Cycle Number	Hardwired Processor IDs and Priority ¹			
	ID1	ID2	ID3	ID4
1 ²	M	1	2- $\overline{\text{BR}}$	3
2	2	3- $\overline{\text{BR}}$	M- $\overline{\text{BR}}$	1
3	2	3- $\overline{\text{BR}}$	M	1

Table 3-31. Rotating Priority Arbitration Example (Cont'd)

Cycle Number	Hardwired Processor IDs and Priority ¹			
	ID1	ID2	ID3	ID4
4	3- $\overline{\text{BR}}$	M	1	2- $\overline{\text{BR}}$
5 ³	1- $\overline{\text{BR}}$	2	3	M

- 1 The following symbols appear in these cells: 1-3 = assigned priority, M = bus mastership (in that cycle), $\overline{\text{BR}}$ = requesting bus mastership with $\overline{\text{BRx}}$
- 2 Initial priority assignments
- 3 Final priority assignments

Bus Mastership Time-out

In either the fixed or rotating priority scheme, systems may need to limit how long a bus master can retain the bus. This is accomplished by forcing the bus master to deassert its $\overline{\text{BRx}}$ line after a specified number of CLKIN cycles and giving the other processors a chance to acquire bus mastership.

To set up a bus master time-out, a program must load the bus time-out maximum (BMAX register, address = 0x180D, [Figure 3-17](#)) with the maximum number of CLKIN cycles (minus 2) that allows the processor to retain bus mastership. This equation is shown below.

$$\text{BMAX} = (\text{maximum number of bus mastership } \text{CLKIN} \text{ cycles}) - 2$$

The minimum value for BMAX is 2, which lets the processor retain bus mastership for four CLKIN cycles. Setting $\text{BMAX}=1$ is not allowed. To disable the bus master time-out function, set $\text{BMAX}=0$.

Each time a processor acquires bus mastership, its bus time-out counter (BCNT register, address = 0x180E) is loaded with the value in BMAX . The BCNT is then decremented in every CLKIN cycle in which the master performs a read or write over the bus and any other (slave) processors are requesting the bus. Any time the bus master deasserts its $\overline{\text{BRx}}$ line, BCNT is reloaded from BMAX .

Shared Memory Interface

BMAX (0x180D)

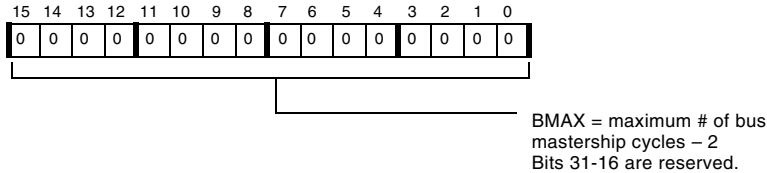


Figure 3-17. BMAX Register

When $\overline{\text{BCNT}}$ decrements to zero, the bus master first completes its off-chip read/write and then deasserts its own $\overline{\text{BRX}}$ (any new off-chip accesses are delayed), which allows transfer of bus mastership. If the $\overline{\text{ACK}}$ signal is holding off an access when $\overline{\text{BCNT}}$ reaches zero, bus mastership is not relinquished until the access can complete.

If $\overline{\text{BCNT}}$ reaches zero while bus lock is active, the bus master does not deassert its $\overline{\text{BRX}}$ line until bus lock is removed. Bus lock is enabled by the $\overline{\text{BUSLK}}$ bit (bit 29 of $\overline{\text{SYSCTL}}$ register). For more information, see [“Bus Lock and Semaphores” on page 3-92](#).

Bus Synchronization After Reset

When a shared memory system comes out of reset (after $\overline{\text{RESET}}$ is asserted), the bus arbitration logic on each processor must synchronize, ensuring that only one processor drives the external bus. One processor must become the bus master, and all other processors must recognize it before actively arbitrating for the bus. The bus synchronization scheme also lets the system safely bring individual processors into and out of reset.

One of the processors in the system must be assigned $\text{ID} = 001$ in order for the bus synchronization scheme to function properly. This processor also holds the external bus control lines stable during reset.



Bus arbitration and synchronization are disabled if the processor is in a single processor system ($\text{ID} = 000$).

To synchronize their bus arbitration logic and define the bus master after a system reset, the multiple processors obey the following rules:

- All processors except the one with $ID = 001$ deassert their \overline{BRX} line during reset. They keep their \overline{BRX} deasserted for at least two cycles after reset and until their bus arbitration logic is synchronized.
- After reset, a processor considers itself synchronized when it detects a cycle in which only one \overline{BRX} line is asserted. The processor identifies the bus master by recognizing which \overline{BRX} is asserted and updates its internal record to indicate the current master.
- The processor with $ID = 001$ asserts its \overline{BRX} during reset and for at least two cycles after reset. If no other \overline{BRX} lines are asserted during these cycles, the processor with $ID = 001$ drives the memory control signals to prevent glitches. Although the processor with $ID = 001$ is asserting its \overline{BRX} and driving the memory control signals during these cycles, this processor does not perform reads or writes over the bus.
- While in reset, the processor with $ID = 001$ attempts to gain control of the bus by asserting \overline{BRI} .
- While in reset, the processor with $ID = 001$ drives the \overline{RD} , \overline{WR} , and $\overline{MS3-0}$ signals only if it determines that it has control of the bus. For the processor to decide it has control of the bus: 1) its \overline{BRI} signal must be asserted and 2) in the previous cycle, no other processor's \overline{BRX} signals were asserted.

The processor with $ID = 001$ continues to drive the \overline{RD} , \overline{WR} , and $\overline{MS3-0}$ signals for two cycles after reset, as long as other \overline{BRX} lines are asserted.

If the processor with $ID = 001$ is synchronized by the end of the two cycles following reset, it becomes the bus master. If it is not synchronized at this time, it deasserts its \overline{BRX} and stops driving the memory control signals and does not arbitrate for the bus until it becomes synchronized. When a processor has synchronized itself, it sets the $BSYN$ bit in the $SYSTAT$ register.

Shared Memory Interface

If one processor comes out of reset after the others have synchronized and started program execution, that processor may not be able to synchronize immediately (for example, if it detects more than one $\overline{\text{BRX}}$ line asserted). If the non-synchronized processor tries to execute an instruction with an off-chip read or write, it cannot assert its $\overline{\text{BRX}}$ line to request the bus and execution is delayed until it can synchronize and correctly arbitrate for the bus.

During reset, the ACK line is pulled high internally by the processor bus master with a 20 k Ω equivalent resistor.

In a system where multiple ADSP-21368 processors share a bank of SDRAM, the master processor's SDRAM controller powers up the SDRAM. The master processor then periodically performs an auto-refresh command as expected. When another ADSP-21368 arbitrates for and receives bus mastership, it assumes the responsibility for performing a pre-charge all command, followed by the auto-refresh command.

When a processor (other than the one responsible for power-up) receives bus mastership for the first time, the auto-refresh command is not performed. This can cause a delay of up to four times the value programmed between the execution of auto-refresh commands. Further, this delay can occur for each processor (other than the master) in the system.

To compensate for this delay, use the following procedure.

1. The processors, other than the one responsible for power-up, should wait for SDRAM power-up to complete. Flags or NOP-loops may be used to accomplish this.
2. After detecting that power-up is complete, and before performing any external data accesses, the processors (other than the power-up processor) should execute the following set of instructions to ensure that an auto-refresh command is executed.

```
r0 = dm(sDRAM_addr);    /* dummy access to grab the bus */  
if not BM jump(pc,0);    /* wait for bus mastership */
```

```

/* Force an auto-refresh */
ustat1 = dm(SDCTL);
bit set ustat1 F_AR;
dm(SDCTL) = ustat1;

```

After this code is executed, the processors in the system can start normal external accesses. These steps must be repeated when a processor is reset.

Bus Synchronization Notes

1. During normal operation, do not reset the current bus master (external hard reset) as this causes system synchronization problems. A few key signals (for example `SDCLK`) are three stated or undriven at reset.
2. The `SDCLK` and `CLKIN` signals are used in the arbitration logic for the shared external bus. This logic requires that these clocks are rising edge aligned to function properly. Therefore, not all clock ratios are allowed in shared memory systems. The values of `PLLM` (PLL multiplier) and `PLLD` (PLL divider), which are set in the power management control register (`PMCTL`), need to be programmed such that the ratio ($PLLM/2PLLD$) is an integer. [For more information, see “Power Management Control Register \(`PMCTL`\)” on page A-170.](#)
3. If the clock ratio from the PLL is changed by programming values of `PLLM` and/or `PLLD` in the `PMCTL` register, the `FSYNC` bit (bit 28 of `SYSCTL` register) should be set to re synchronize the shared memory system. This bit should be cleared after meeting the PLL settling time and the clock locks to the new ratio. If only the value of `PLLD` is programmed, in addition to the above, enter dummy bypass mode by setting and clearing the `PLLBP` bit (bit 15 in the `PMCTL` register) to ensure the `SDCLK` and `CLKIN` are rising edge aligned.



Only SDRAM clock ($SDCLK$) ratios of two and four are supported in shared memory systems.

Bus Lock and Semaphores

To allow the processors in shared memory systems to share resources such as memory or I/O, semaphores can be used. A semaphore is a flag that can be read and written by any of the processors sharing the resource. The value of the semaphore tells the processor when it can access the resource. Semaphores are also useful to synchronize tasks being performed by different processors in a system.

With the use of its bus lock feature, the processor has the ability to read and modify a semaphore—a key requirement of shared memory systems.

Read-modify-write operations on semaphores can be performed if all of the processors obey two simple rules:

1. A processor must not write to a semaphore unless it is the bus master.
2. When attempting a read-modify-write operation on a semaphore, the processor must have bus mastership for the duration of the operation.

Both of these rules apply when a processor uses its bus lock feature, which retains its mastership of the bus and prevents other processors from simultaneously accessing the semaphore.

Bus lock is requested by setting the $BUSLK$ bit in the $SYSCTL$ register. When this happens, the processor initiates the bus arbitration process by asserting its \overline{BRx} line. When the processor becomes bus master, it locks the bus by keeping its \overline{BRx} line asserted, even when it is not performing an external read or write. When the $BUSLK$ bit is cleared, the processor gives up the bus by deasserting its \overline{BRx} line.

While the `BUSLK` bit is set, the processor can determine if it has acquired bus mastership by executing a conditional instruction with the bus master (`BM`) or not bus master (`Not BM`) condition codes, for example:

```
IF NOT BM JUMP(PC,0); /* Wait for bus mastership */
```

If the processor becomes the bus master, it can proceed with the external read or write. If not, it can clear its `BUSLK` bit and try again later.

A read-modify-write operation is accomplished with the following steps:

1. Request bus lock by setting the `BUSLK` bit in `SYSCTL`.
2. Wait for bus mastership to be acquired.
3. Read the semaphore, test it, then write to it.

Locking the bus prevents other processors from writing to the semaphore while the read-modify-write operation is occurring.

Shared Memory Interface Status

The system status (`SYSTAT`) register provides status information for host and multiprocessor systems. [Table 3-32](#) shows the status bits in this register.

Table 3-32. `SYSTAT` Register

Bit(s)	Name	Definition
1	BSYN	Bus Synchronized. This bit indicates whether the processor's bus arbitration logic is synchronized (if set, =1) or is not synchronized (if cleared, =0, reset value).
6-4	CRBM	Current Bus Master. These bits indicate the ID of the processor that is currently the bus master in a multiprocessor system.
10-8	IDC	ID Code. These bits indicate the state of the ID pins on the processor.

Shared Memory and the SDRAM Controller

In a shared memory environment, the SDRAM is shared among two or more ADSP-21368 processors. SDRAM input signals (including clock) are always driven by the bus master. The current bus master continues to hold the bus for $t_{RASmin} - 1$ cycles before giving up the bus to the new bus master.

The clock is three-stated on releasing the bus, and command lines are driven for one extra cycle with a NOP instruction. The new bus master also drives a NOP on the command lines immediately after acquiring the bus mastership. This prevents latching of invalid commands due to glitches on the clock, (if any) during bus mastership changeover.

The following should be noted when using the SDRAM controller in a shared memory system.

1. Processors do not track commands on the bus.
2. The master processor issues a refresh command immediately after getting bus-mastership and clearing its refresh counter. This simplifies the design and avoids maintaining the refresh counters in sync on all processors using an overhead of a few clock cycles on each mastership changeover.
3. For shared SDRAM timing, all processors must have the same SDCLK frequency, and the same core clock (CCLK) to SDRAM clock (SDCLK) ratio. This implies that all processors must use the same settings in their respective control (SDCTL) and refresh rate (SDRRC) registers.

Shared Memory Booting

The ADSP-21368 processor allows booting for multiple processors from a single EPROM/FLASH. [For more information, see “Shared Memory Booting” on page 14-40.](#)

4 DIGITAL AUDIO/DIGITAL PERIPHERAL INTERFACES

The digital audio interface (DAI) and the digital peripheral interface (DPI) are comprised of a groups of peripherals and their respective signal routing units (SRU1 and SRU2). The inputs and outputs of the peripherals are not directly connected to external pins. Rather, the SRUs connect the peripherals to a set of pins and to each other, based on a set of configuration registers. This allows the peripherals to be interconnected to suit a wide variety of systems. It also allows the ADSP-21367/8/9 and ADSP-2137x SHARC processors to include an arbitrary number and variety of peripherals while retaining high levels of compatibility without increasing pin count. [Table 4-1](#) shows the peripherals that are assigned to each SRU.



The peripherals listed in [Table 4-1](#) represent the maximum in any single processor. For a listing

Table 4-1. Signal Routing Unit Peripheral Assignments

DAI SRU1	DPI SRU2 ¹
Serial Ports (up to 8) ²	Timers (3)
S/PDIF Transmitter/Receiver	Serial Peripheral Interface (2)
Interrupts (10)	Two Wire Interface
Precision Clock Generators (4)	Universal Asynchronous Receiver/Transmitter (2)
Asynchronous Sample Rate Converter (8 channels)	General-Purpose I/O (9)

Structure of the Interfaces

Table 4-1. Signal Routing Unit Peripheral Assignments (Cont'd)

DAI SRU1	DPI SRU2 ¹
Input Data Port	Flags (12)
General-Purpose I/O (20 channels)	

- 1 The precision clock generator (PCG) units C and D can also be routed through the DPI.
- 2 SPORT6 and SPORT7 receive clocks from other sources but cannot send their own clocks to other SPORTS or other peripherals internally through SRU1. If needed, they have to be connected externally through pins.

The following sections describe the general operating theory of both the DAI and DPI as well as their SRUs.


Structure of the Interfaces

Both the DAI and DPI incorporate a specific set of peripherals and a very flexible routing (connection) system permitting a large combination of signal flows. A set of DAI/DPI-specific registers make such design, connectivity, and functionality variations possible. All routing related to peripheral states for the DAI/DPI interfaces are specified using registers related to each interface. For more information on pin states, refer to [Figure 4-6 on page 4-10](#).

The DAI/DPI may be used to connect any combination of inputs to any combination of outputs. This function is performed by the SRUs through memory-mapped registers.

This *virtual connectivity* design offers a number of distinct advantages:

- Flexibility
- Increased numbers and kinds of configurations
- Connections can be made through software—no hard wiring is required

 Inputs may only be connected to outputs.

DAI/DPI System Design

[Figure 4-1](#) and [Figure 4-2](#) show how the DAI pin buffers are connected through SRU1 and [Figure 4-3](#) and [Figure 4-4](#) show how the DPI pin buffers are connected through SRU2. The SRUs allow for very flexible data routing. In its design, the DAI/DPI makes use of several types of data from the sources, shown in [Table 4-1](#).

For a sample of a DAI system configuration, refer to “[Using the SRU\(\) Macro to Configure the DAI](#)” on [page 4-76](#).

DAI/DPI System Design

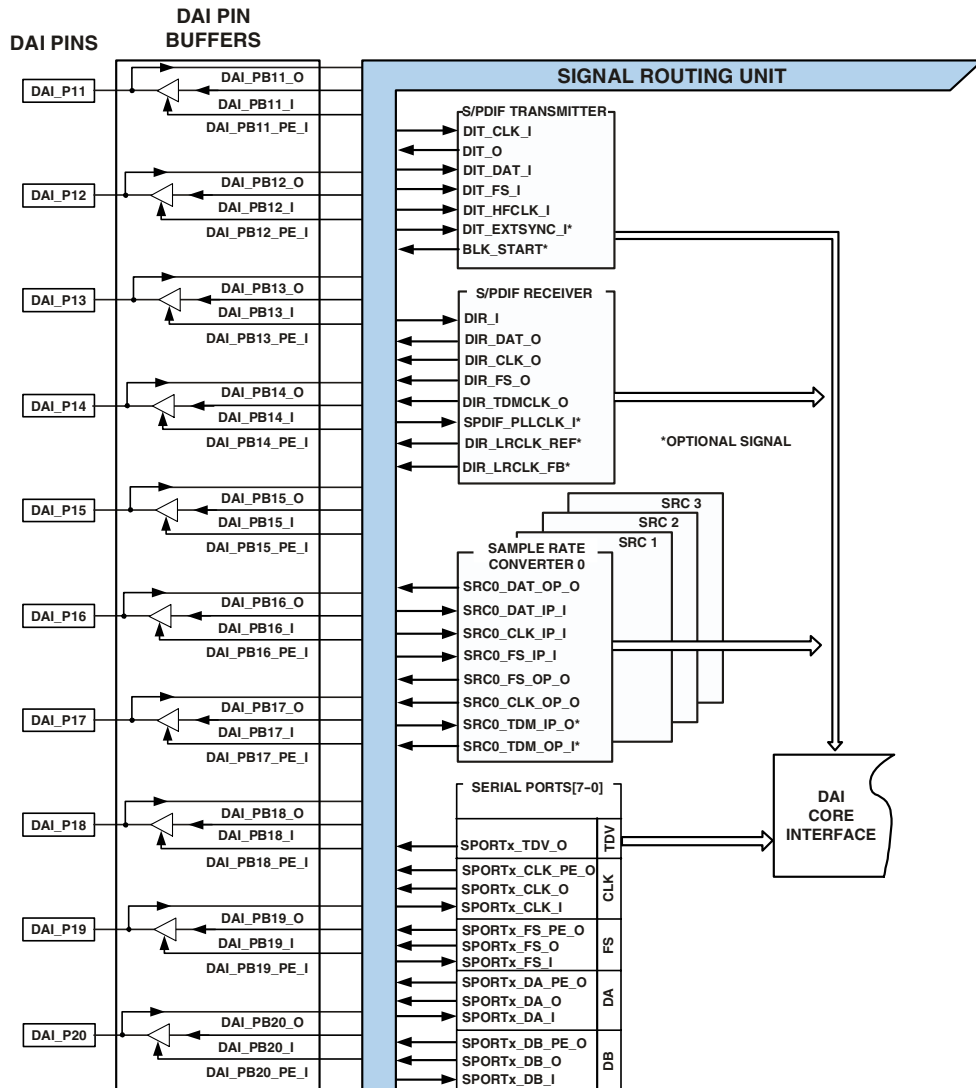


Figure 4-1. DAI System Design

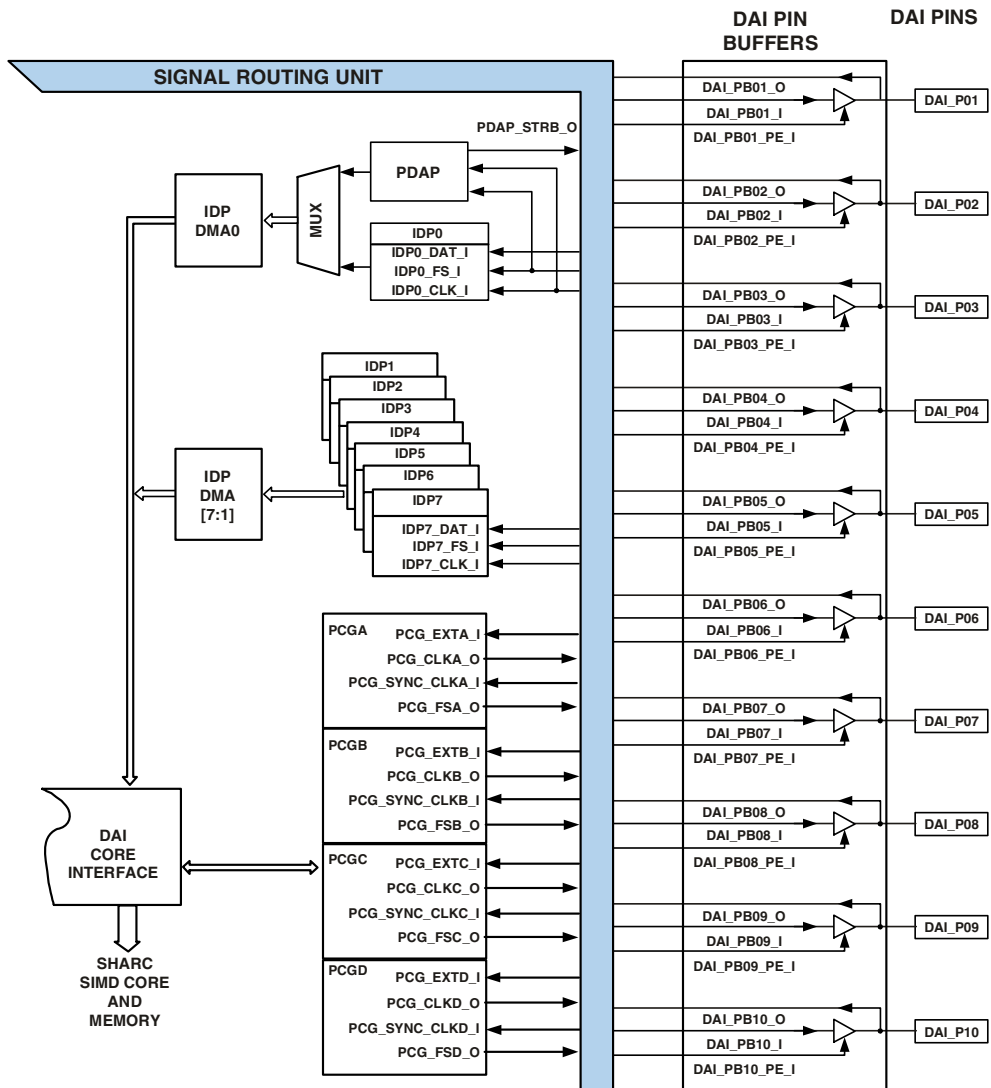


Figure 4-2. DAI System Design (continued)

DAI/DPI System Design

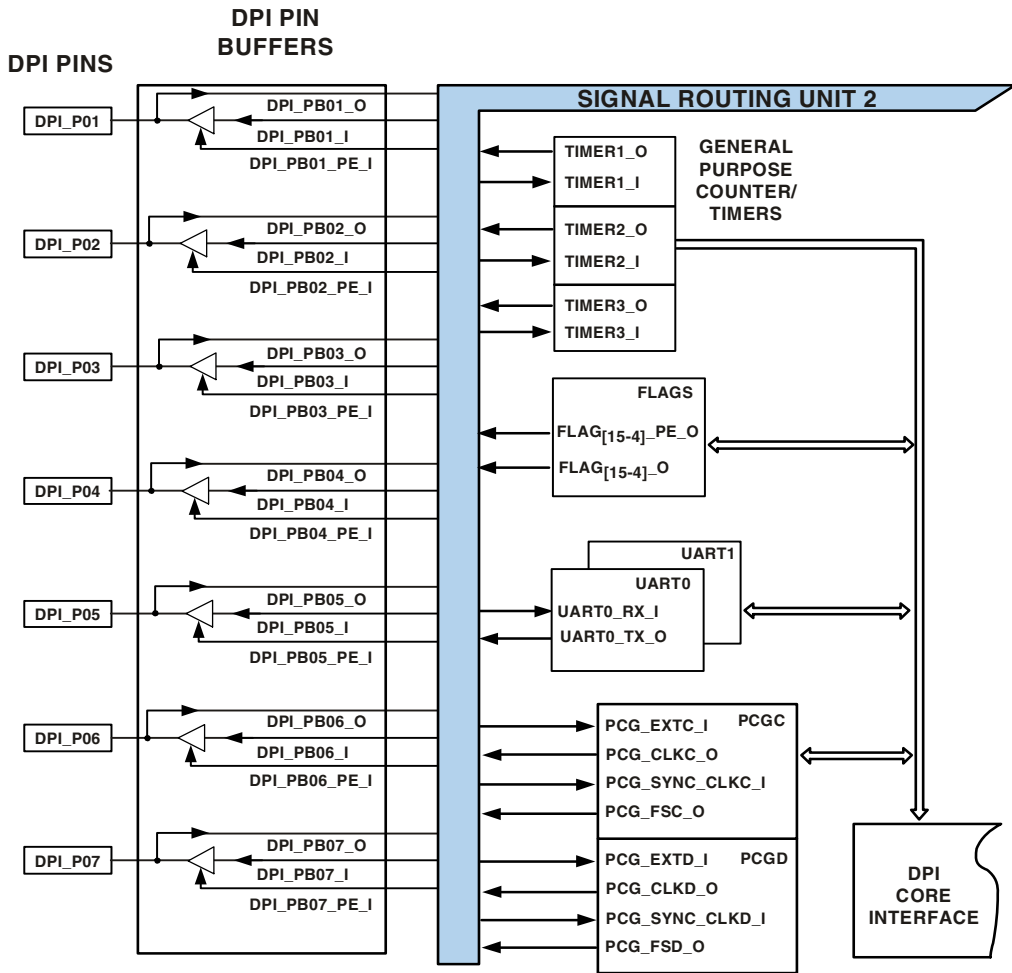


Figure 4-3. DPI System Design

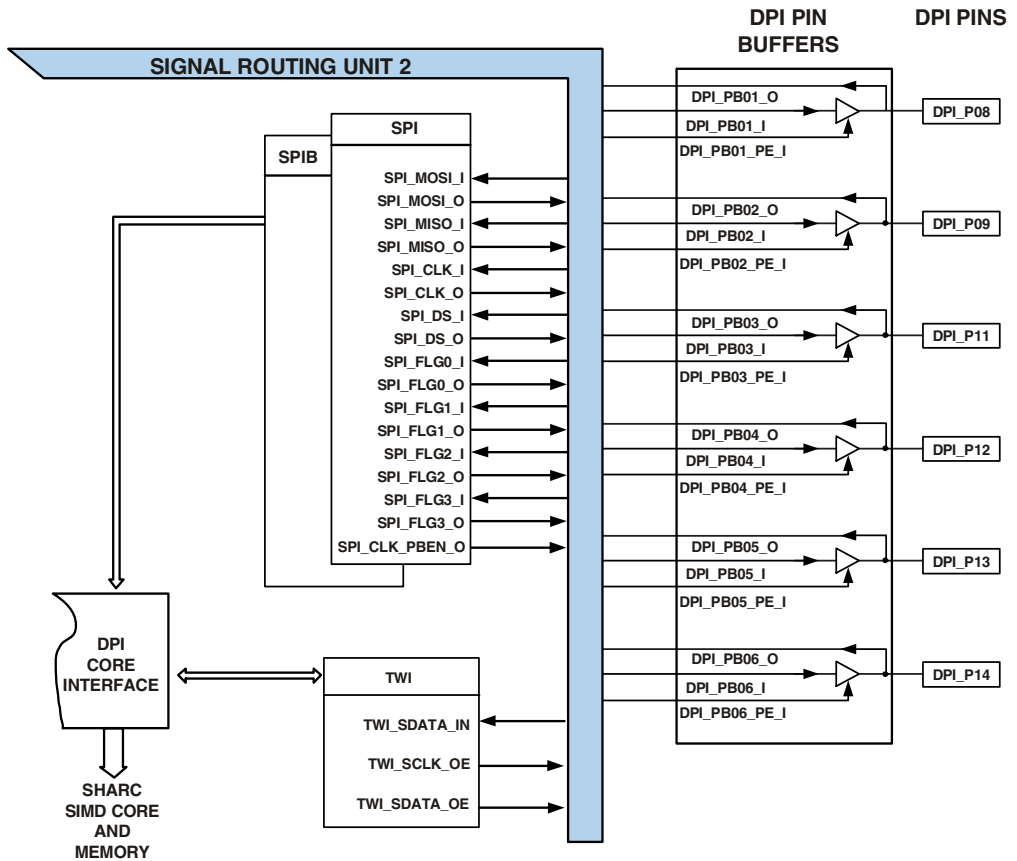


Figure 4-4. DPI System Design (continued)

Signal Routing Units

This section describes how to use the signal routing units (SRU1 and SRU2) to connect inputs to outputs.

Connecting Peripherals

The SRUs can be likened to a set of patch bays, which contains a bank of inputs and a bank of outputs. For each input, there is a set of permissible output options. Outputs can feed to any number of inputs in parallel, but every input must be patched to exactly one valid output source. Together, the set of inputs and outputs are called a group. The signal's inputs and outputs that comprise each group all serve similar purposes. They are compatible such that almost any output-to-input patch makes functional sense.

Table 4-2 lists the SRU signal groups that are named sequentially A through F for SRU1 and A through C for SRU2. Each group routes a unique set of signals with a specific purpose. For example, SRU1 Group A routes clock signals, while SRU2 group A routes the external 14 pins and other sources like timer outputs to the inputs of the various peripherals. Unlike SRU1 in the DAI module, all types of functionality like serial clock and data are clubbed into the same group in the DPI/SRU2.

Table 4-2. Pin Signal Group Assignments

Signal Group	DAI (SRU1)	DPI (SRU2)
Group A	Clock routing control	Input routing control, includes serial clock and data
Group B	Serial data routing control	Pin signal assignments
Group C	Frame sync routing control	Pin enable assignment, routes enables of various peripherals to the drive buffer of each of the 14 pins
Group D	Pin signal assignments	N/A

Table 4-2. Pin Signal Group Assignments (Cont'd)

Signal Group	DAI (SRU1)	DPI (SRU2)
Group E	Interrupts and miscellaneous signals	N/A
Group F	Pin enable signals used to specify whether each DAI pin is used as an output or an input.	N/A

Each input and output in each group is given a unique mnemonic. In the few cases where a signal appears in more than one group, the mnemonic is slightly different to distinguish between the connections. The convention is to begin the name with an identifier for the peripheral that the signal is coming to/from followed by the signal's function as shown in [Figure 4-5](#). A number is included if the DAI contains more than one peripheral type (for example, serial ports) or if the peripheral has more than one signal that performs this function (for example, IDP channels). The mnemonic always ends with `_I` if the signal is an input, or with `_O` if the signal is an output.

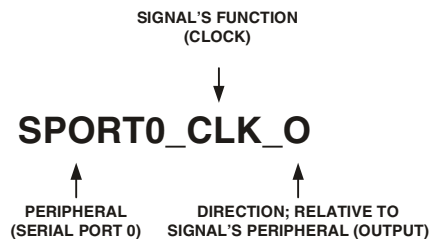


Figure 4-5. Example SRU Mnemonic

Note that it is not possible to connect a signal in one group directly to a signal in a different group (analogous to wiring from one patch bay to another). However, group D (in SRU1) is largely devoted to routing in this vein.

Pin Interface

Within the context of the SRUs, physical connections to the DAI/DPI pins are replaced by a logical interface known as a *pin buffer*. This is a three-terminal active device capable of sourcing/sinking output current when its driver is enabled, and passing external input signals when disabled. Each pin has an input, output, and enable pin as shown in [Figure 4-6](#). The inputs and the outputs are defined with respect to the pin, similar to a peripheral device. This is consistent with the SRU naming convention.

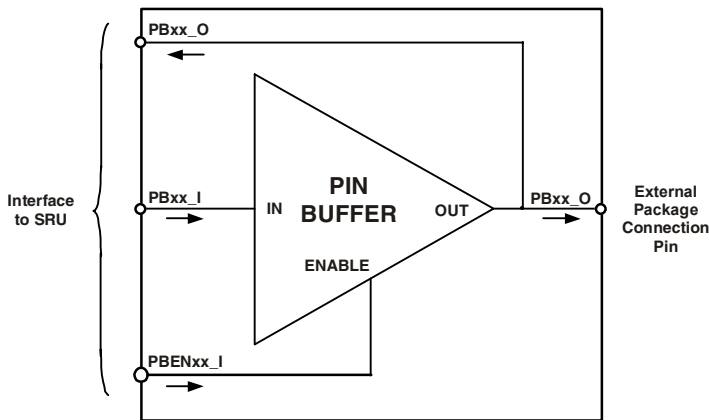


Figure 4-6. Pin Buffer Example

The notation for pin input and output connections can be quite confusing at first because, in a typical system, a pin is simply a wire that connects to a device. The manner in which the pins are connected within the SRU requires additional nomenclature. The pin interface's input may be thought of as the input to a buffer amplifier that can drive a load on the physical external lead. The pin interface enable is the input signal that enables the output of the buffer by turning it on when its value is logic high, and turning it off when its value is logic low.

When the pin enable is asserted, the pin output is logically equal to pin input, and the pin is driven. When the pin enable is deasserted, the output of the buffer amplifier becomes high impedance. In this situation, an external device may drive a level onto the line, and the pin is used as an input to the ADSP-21367/8/9 and ADSP-2137x processors.

While the pin is at high impedance and another device is driving a logic level onto the external pin, this logic level value is sent to the SRUs as the pin interface output. Even though the signal is an input to the SHARC processor, it is an output from the pin interface (as a three-terminal device) and may be patched to the signal inputs of peripherals within the SRUs. Pin output is equal to pin input when the pin enable is asserted, but pin output is equal to the external (input) signal when the pin enable is deasserted.



If a DAI/DPI pin is not being used, the pin enable (for example `DAI_PBxx_I`) for its pin buffer should be connected to `LOW` and its associated bit in the `DAI_PIN_PULLUP` register should be set (= 1) to enable a 22.5 k Ω pull-up resistor for that pin.

Pin Buffers as Signal Output Pins

In a typical embedded system, most pins are designated as either inputs or outputs when the circuit is designed, even though they may have the ability to be used in either direction. Each of the DAI/DPI pins can be used as either an input or an output. Although the direction of a DAI/DPI pin is set simply by writing to a memory-mapped register, most often the pin's direction is dictated by the designated use of that pin. For example, if a DAI pin is hard wired to only the input of another interconnected circuit, it would not make sense for the corresponding pin buffer to be configured as an input. Input pins are commonly tied to logic high or logic low to set the input to a fixed value. Similarly, setting the direction of a DAI pin at system startup by tying the pin buffer enable to a fixed value (either logic high or logic low) is often the simplest and cleanest way to configure the SRUs.

Signal Routing Units

When the DAI/DPI pin is used only as an output, connect the corresponding pin buffer enable to logic high as shown in Figure 4-7. This enables the buffer amplifier to operate as a current source and to drive the value present at the pin buffer input onto the DAI/DPI pin and off chip. When the pin buffer enable (in this example $PBEN_{xx_I}$) is set ($= 1$), the pin buffer output (PB_{xx_O}) is the same signal as the pin buffer input (PB_{xx_I}), and this signal is driven as an output.

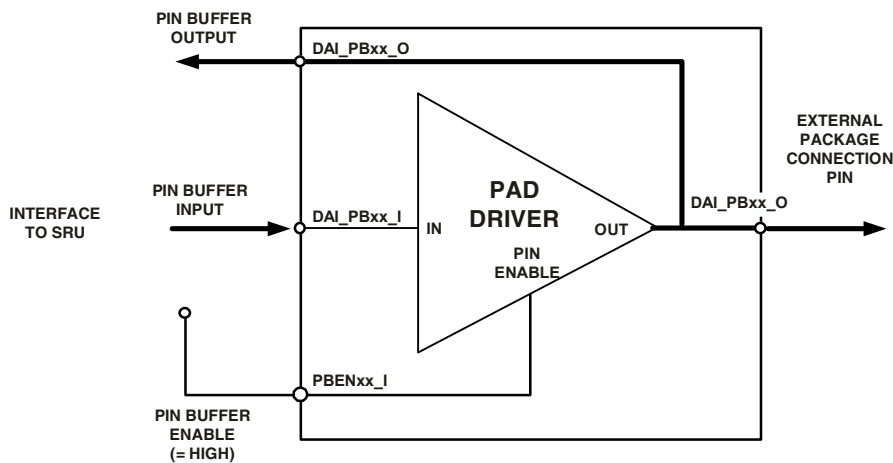


Figure 4-7. Pin Buffer as Output

Pin Buffers as Signal Input Pins

When the DAI/DPI pin is used only as an input, connect the corresponding pin buffer enable to logic low as shown in Figure 4-8. This disables the buffer amplifier and allows an off-chip source to drive the value present on the DAI/DPI pin and at the pin buffer output. When the pin buffer enable ($PBEN_{xx_I}$) is cleared ($= 0$), the pin buffer output (PB_{xx_O}) is the signal driven onto the DAI pin by an external source, and the pin buffer input (PB_{xx_I}) is not used.

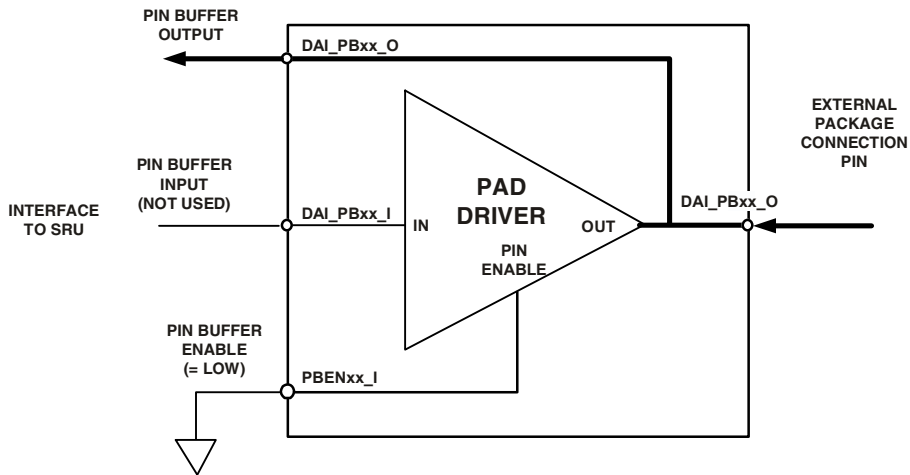


Figure 4-8. Pin Buffer as Input

Although not strictly necessary, it is recommended programming practice to tie the pin buffer input to logic low whenever the pin buffer enable is tied to logic low. By default, the pin buffer enables are connected to SPORT pin enable signals that may change value. Tying the pin buffer input low decouples the line from irrelevant signals and can make code easier to debug. It also ensures that no voltage is driven by the pin if a bug in your code accidentally asserts the pin enable.

Bidirectional Pin Buffers

All peripherals within the DAI and DPI that have bidirectional pins generate a corresponding pin enable signal. Typically, the settings within a peripheral's control registers determine if a bidirectional pin is an input or an output, and is then driven accordingly. Both the peripheral control registers and the configuration of the SRUs can effect the direction of signal flow in a pin buffer.

Signal Routing Units

For example, from an external perspective, when a serial port (SPORT) is completely routed off chip, it uses four pins—clock, frame sync, data channel A, and data channel B. Because all four of these pins comprise the interface that the serial port presents to SRU1, there are a total of 12 connections as shown in [Figure 4-9](#).

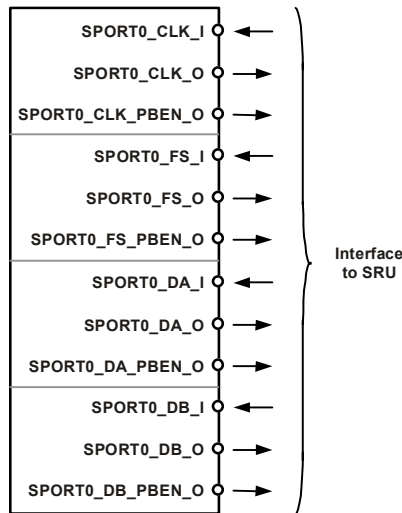


Figure 4-9. SRU1 Connections for SPORT_x

For each bidirectional line, the serial port provides three separate signals. For example, a SPORT clock has three separate SRU connections—an input clock to the SPORT (SPORT_x_CLK_I), an output clock from the SPORT (SPORT_x_CLK_O), and an output enable from the SPORT (SPORT_x_CLK_PBEN_O). Note that the input and output signal pair is never used simultaneously. The pin enable signal dictates which of the two SPORT lines appear at the DAI pin at any given time. By connecting all three signals through SRU1, the standard SPORT configuration registers behave as documented in [“SPORT Serial Control Registers \(SPCTL_x\)”](#) on

[page A-29](#). SRU1 then becomes transparent to the peripheral. [Figure 4-10](#) demonstrates SPORT0 properly routed to DAI pins one through four—although it can be equally well routed to any of the 20 DAI pins.

Though SPORT signals are capable of operating in this bidirectional manner, it is not required that they be connected to the pin buffer this way. As mentioned above, if the system design only uses a SPORT signal in one direction, it is easier and safer to connect the pin buffer enable directly high or low as appropriate. Furthermore, signals in the SRUs other than the pin buffer enable signal (which is generated by the peripheral) may be routed to the pin buffer enable input. For example, an outside source may be used to ‘gate’ a pin buffer output by controlling the corresponding pin buffer enable.

Making Connections in the SRUs

As described previously, the SRUs are similar to a set of patch bays. Each bay routes a distinct set of outputs to compatible inputs. These connections are implemented as a set of memory-mapped registers with a bit field for each input. The outputs are implemented as a set of bit encodings. Conceptually, a patch cord is used to connect an output to an input. In the SRUs, a bit pattern that is associated with a signal output (shown as item 1 in [Figure 4-11](#)) is written to a bit field corresponding to a signal input (shown as item 2 in [Figure 4-11](#)).

The memory-mapped SRU registers are arranged by groups, referred to as group A through group F in SRU1 and group A through group C in SRU2 and described in [“DAI/SRU1 Connection Groups” on page 4-18](#) and [“DPI/SRU2 Connection Groups” on page 4-51](#). Each group has a unique encoding for its associated output signals and a set of configuration registers. For example, in the DAI, group A is used to route clock signals. Five memory-mapped registers, `SRU_CLK4-0`, contain 5-bit wide fields corresponding to the clock inputs of various peripherals. The values written to these bit fields specify a signal source that is an output from

Making Connections in the SRUs

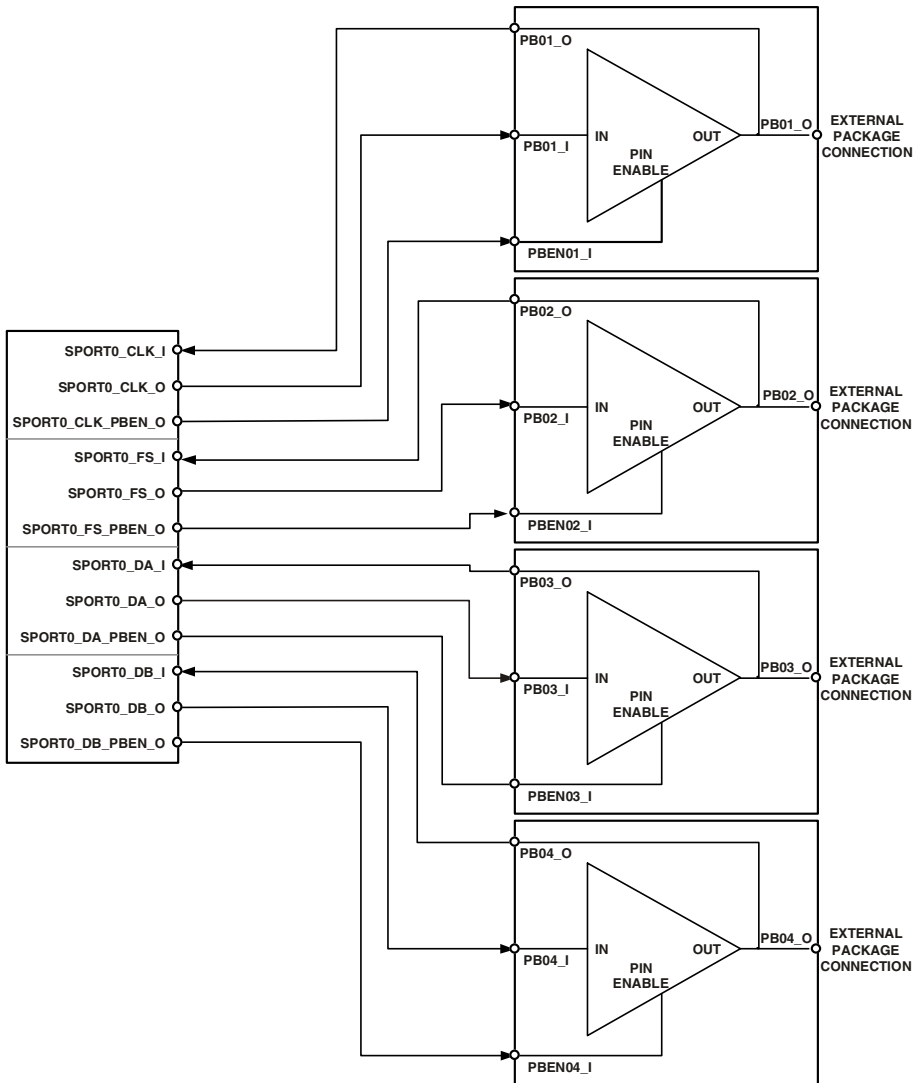


Figure 4-10. SRU1 Connection to Four Bidirectional SPORT Pins

another peripheral. All of the possible encodings represent sources that are clock signals (or at least could be clock signals in some systems).

[Figure 4-11](#) diagrams the input signals that are controlled by the DAI group A register, `SRU_CLK0`. All bit fields in the `SRU1` configuration registers correspond to inputs. The value written to the bit field specifies the signal source. This value is also an output from some other component within the `SRU`.

Note that the lower portion of the patch bay in [Figure 4-11](#) is shown with a large number of ports to reinforce the point that one output can be connected to many inputs. The same encoding can be written to any number of bit fields in the same group. It is not possible to run out of patch points for an output signal.

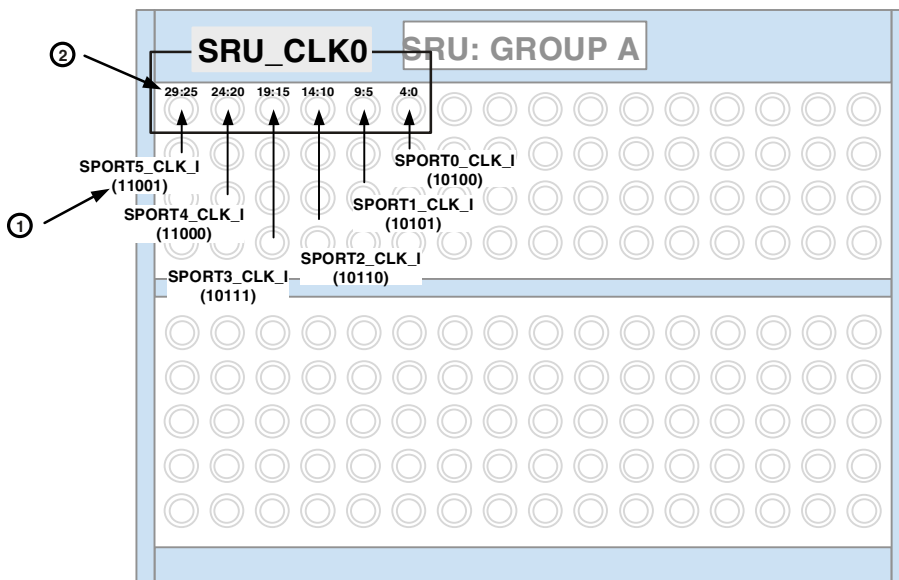


Figure 4-11. Patching to the Group A Register `SRU_CLK0`

Making Connections in the SRUs

Just as DAI group A routes clock signals, each of the other groups route a collection of compatible signals. Group B routes serial data streams while group C routes frame sync signals. Group D routes signals to pins so that they may be driven off chip. Note that all of the groups have an encoding that allows a signal to flow from a pin output to the input being specified by the bit field. However, group D is required to route a signal to the pin input. Group F routes signals to the pin enables, and the value of these signals determines if a DAI pin is used as an output or an input. These groups are described in more detail in the following sections.

DAI/SRU1 Connection Groups

The DAI/SRU1 has the default configuration shown in [Table 4-3](#).

Table 4-3. DAI/SRU1 Default Configuration


Pin Number	Signal Name	Pin Number	Signal Name
DAI_01	SPORT0_DA	DAI_11	SPORT3_DA
DAI_02	SPORT0_DB	DAI_12	SPORT3_DB
DAI_03	SPORT0_CLK	DAI_13	SPORT3_CLK
DAI_04	SPORT0_FS	DAI_14	SPORT3_FS
DAI_05	SPORT1_DA	DAI_15	SPORT4_DA
DAI_06	SPORT1_DB	DAI_16	SPORT4_DB
DAI_07	SPORT1_CLK	DAI_17	SPORT5_DA
DAI_08	SPORT1_FS	DAI_18	SPORT5_DB
DAI_09	SPORT2_DA	DAI_19	SPORT5_CLK
DAI_10	SPORT2_DB	DAI_20	SPORT5_FS

There are five separate groups of connections that are used in SRU1. The following sections summarize each.


Group A Connections—Clock Signals

Group A is used to route the following signals to clock inputs and are selected from the list of group A sources.

- SPORTs clock inputs (when the SPORTs are in clock slave mode)
- Clock inputs to the eight IDP (input data port) channels
- Four precision clock generator (PCG) external sources
- SRC clock inputs
- SPDIF transmitter clock inputs

 When channel 0 of the IDP is configured for PDAP input, the clock source set here is used as the parallel word latch instead of the serial bit clock.

Set all clock inputs that are not used to logic low. Any IDP channels that receive clock signals as set here send data to the FIFO. When a SPORT is used as a clock master, setting the unused SPORT clock input to logic low improves signal integrity. The registers are shown in [Figure 4-12](#) through [Figure 4-17](#). The input and output signals for group A are summarized in [Table 4-4 on page 4-23](#).

 The following notes apply to the group A connections

1. The `SRU_CLK4-0` registers are 30-bit registers. On reads, bits 30 and 31 always return zero.
2. SPORTs 6 and 7 receive their clocks from other sources but cannot send their own clocks to other SPORTs or other peripherals internally through SRU. If needed, they have to be connected externally through pins.

Making Connections in the SRUs

- Setting $\text{SRU_CLK4}[4:0] = 0x1C$ connects PCG_EXTA_I to logic low, not PCG_CLKA_0 .
Setting $\text{SRU_CLK4}[9:5] = 0x1D$ connects PCG_EXTB_I to logic low, not PCG_CLKB_0 .

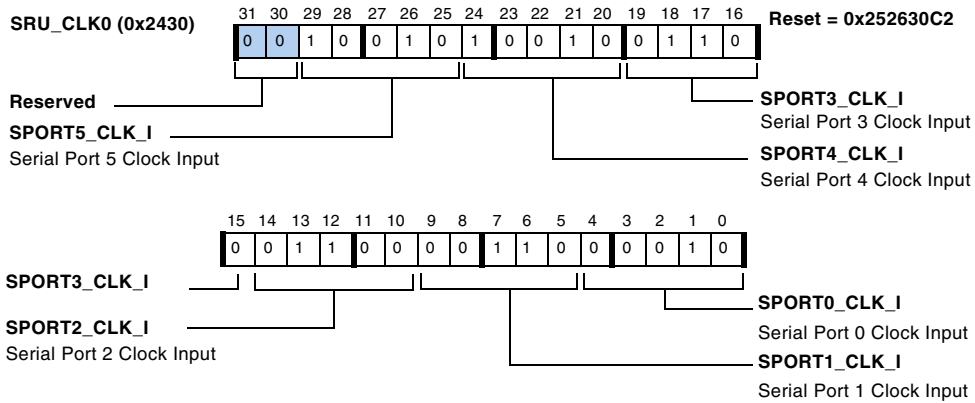


Figure 4-12. SRU_CLK0 Register

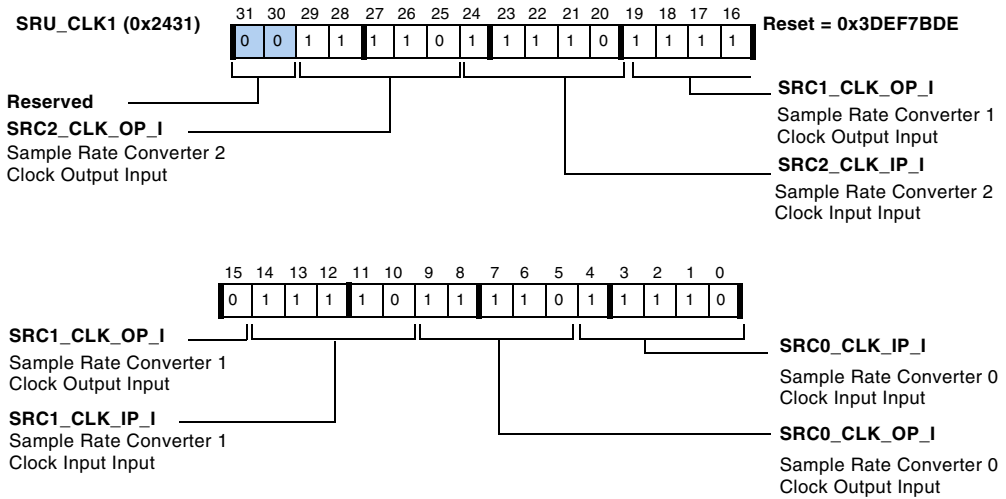


Figure 4-13. SRU_CLK1 Register

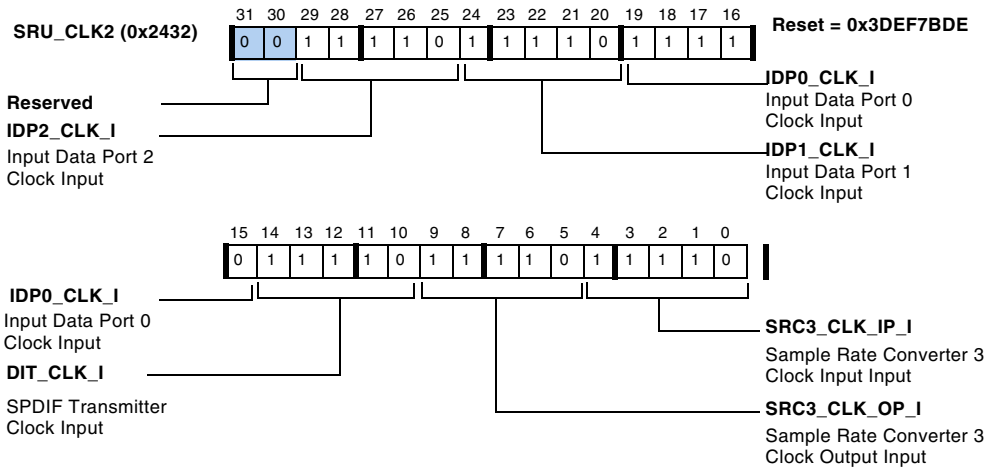


Figure 4-14. SRU_CLK2 Register

Making Connections in the SRUs

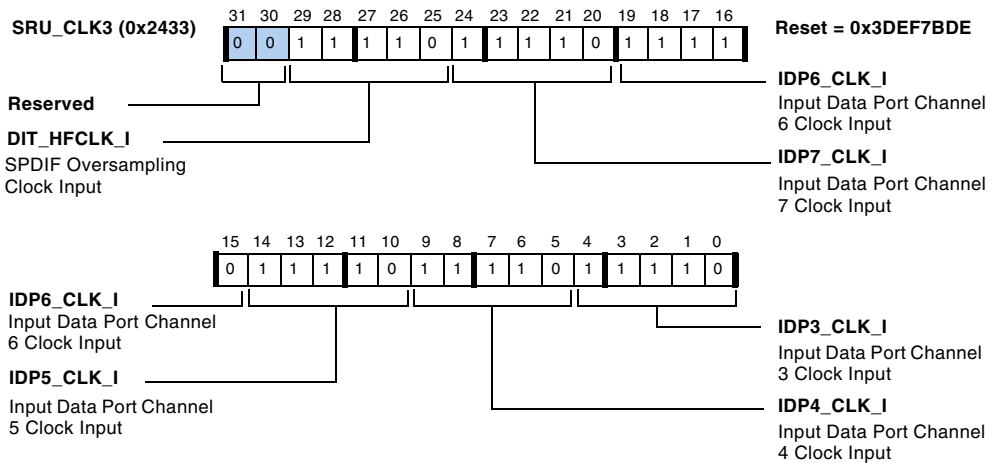
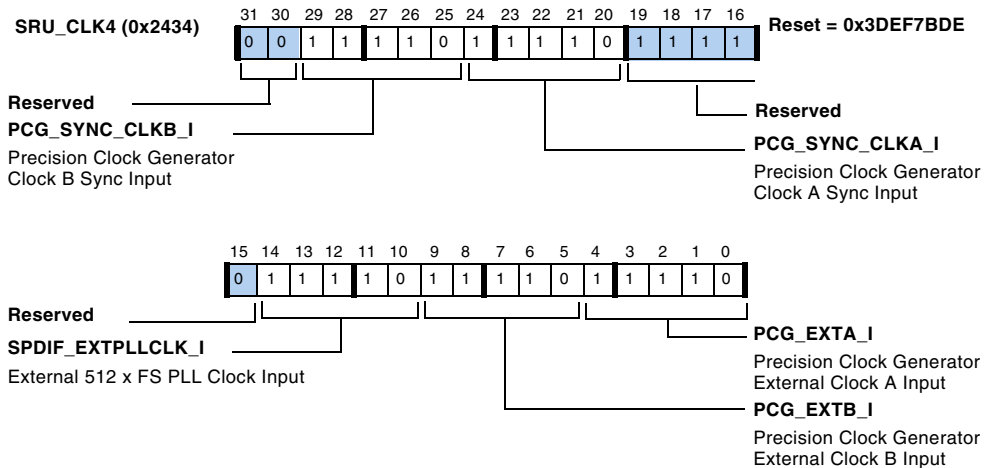


Figure 4-15. SRU_CLK3 Register



Setting SRU_CLK4 4-0 = 28 connects PCG_EXTB_I to logic low, not to PCG_CLKA_O.
 Setting SRU_CLK4 9-5 = 29 connects PCG_EXTB_I to logic low, not to PCG_CLKB_O.

Figure 4-16. SRU_CLK4 Register

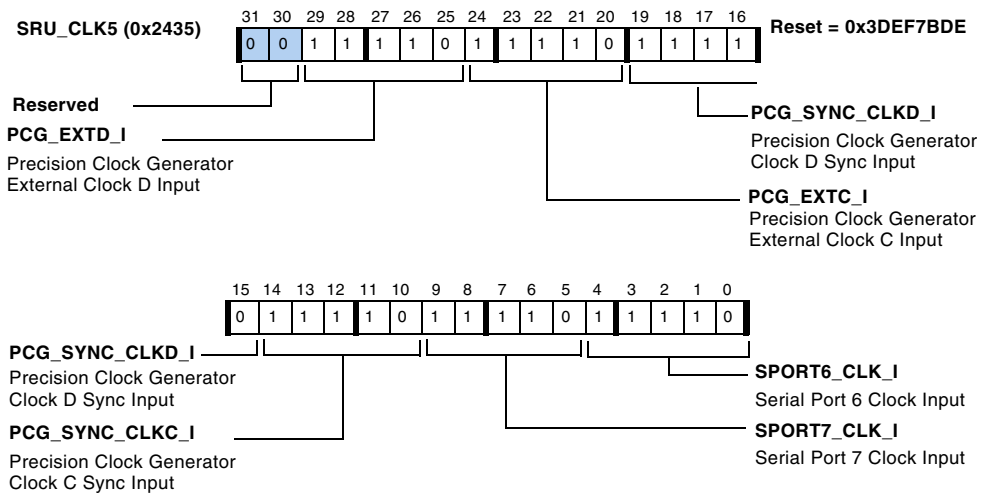


Figure 4-17. SRU_CLK5 Register

Table 4-4. Group A Sources—Serial Clock

Selection Code	Source Signal	Description (Source)
00000 (0x0)	DAI_PB01_O	Select DAI pin buffer 1
00001 (0x1)	DAI_PB02_O	Select DAI pin buffer 2
00010 (0x2)	DAI_PB03_O	Select DAI pin buffer 3
00011 (0x3)	DAI_PB04_O	Select DAI pin buffer 4
00100 (0x4)	DAI_PB05_O	Select DAI pin buffer 5
00101 (0x5)	DAI_PB06_O	Select DAI pin buffer 6
00110 (0x6)	DAI_PB07_O	Select DAI pin buffer 7
00111 (0x7)	DAI_PB08_O	Select DAI pin buffer 8
01000 (0x8)	DAI_PB09_O	Select DAI pin buffer 9
01001 (0x9)	DAI_PB10_O	Select DAI pin buffer 10
01010 (0xA)	DAI_PB11_O	Select DAI pin buffer 11

Making Connections in the SRUs

Table 4-4. Group A Sources—Serial Clock (Cont'd)

Selection Code	Source Signal	Description (Source)
01011 (0xB)	DAI_PB12_O	Select DAI pin buffer 12
01100 (0xC)	DAI_PB13_O	Select DAI pin buffer 13
01101 (0xD)	DAI_PB14_O	Select DAI pin buffer 14
01110 (0xE)	DAI_PB15_O	Select DAI pin buffer 15
01111 (0xF)	DAI_PB16_O	Select DAI pin buffer 16
10000 (0x10)	DAI_PB17_O	Select DAI pin buffer 17
10001 (0x11)	DAI_PB18_O	Select DAI pin buffer 18
10010 (0x12)	DAI_PB19_O	Select DAI pin buffer 19
10011 (0x13)	DAI_PB20_O	Select DAI pin buffer 20
10100 (0x14)	SPORT0_CLK_O	Select SPORT 0 Clock
10101 (0x15)	SPORT1_CLK_O	Select SPORT 1 Clock
10110 (0x16)	SPORT2_CLK_O	Select SPORT 2 Clock
10111 (0x17)	SPORT3_CLK_O	Select SPORT 3 Clock
11000 (0x18)	SPORT4_CLK_O	Select SPORT 4 Clock
11001 (0x19)	SPORT5_CLK_O	Select SPORT 5 Clock
11010 (0x1A)	DIR_CLK_O	Select SPDIF receive clock output
11011 (0x1B)	DIR_TDMCLK_O	Select SPDIF receive TDM clock output
11100 (0x1C)	PCG_CLKA_O	Select precision clock A output
11101 (0x1D)	PCG_CLKB_O	Select precision clock B output
11110 (0x1E)	LOW	Select logic level low (0)
11111 (0x1F)	HIGH	Select logic level high (1)

Group B Connections—Data Signals

Group B connections are used to route signals to serial data inputs. This includes serial data inputs to both the A and B channels of the SPORTs and to each of the eight IDP channels. The SRCs and SPDIF transmitter are also selected from the list of group B sources and set in the group B registers. When a SPORT is configured to transmit, the data source set here is ignored. Likewise, when channel 0 of the IDP is used for the PDAP, the serial data source set here is ignored. The registers are shown in [Figure 4-18](#) through [Figure 4-24](#). The input and output signals for group B are summarized in [Table 4-5 on page 4-29](#).



The following notes apply to group B connections.

1. SRU_DAT0, SRU_DAT1, SRU_DAT2, SRU_DAT3, SRU_DAT4, and SRU_DAT5 are 30-bit registers. On reads, bits 30 and 31 always return zero.
2. SRU_DAT6 is a 24-bit register. On reads, bits 31 through 24 always return zero.

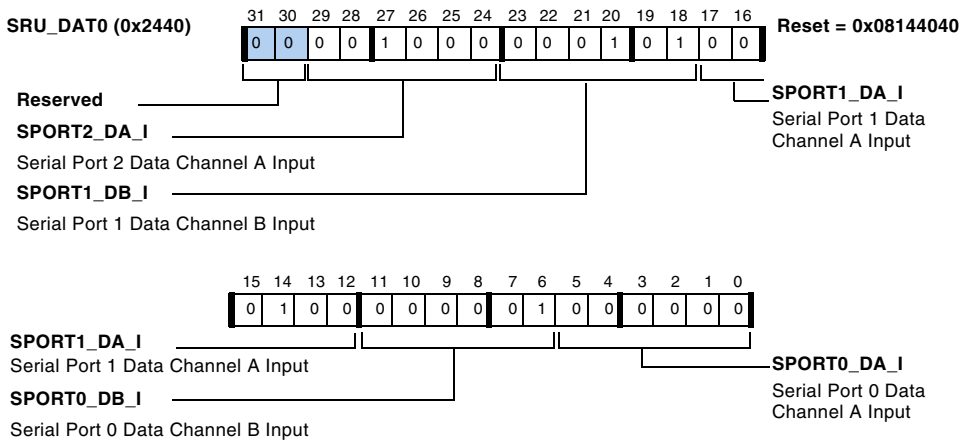


Figure 4-18. SRU_DAT0 Register

Making Connections in the SRUs

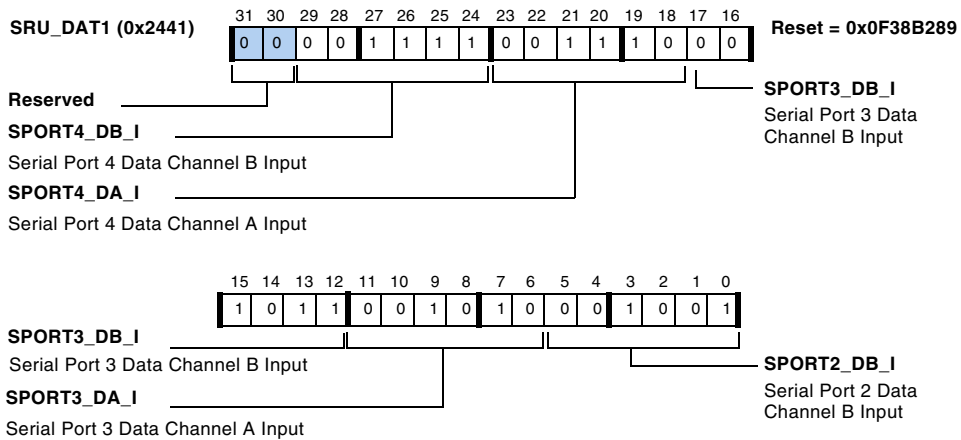


Figure 4-19. SRU_DAT1 Register

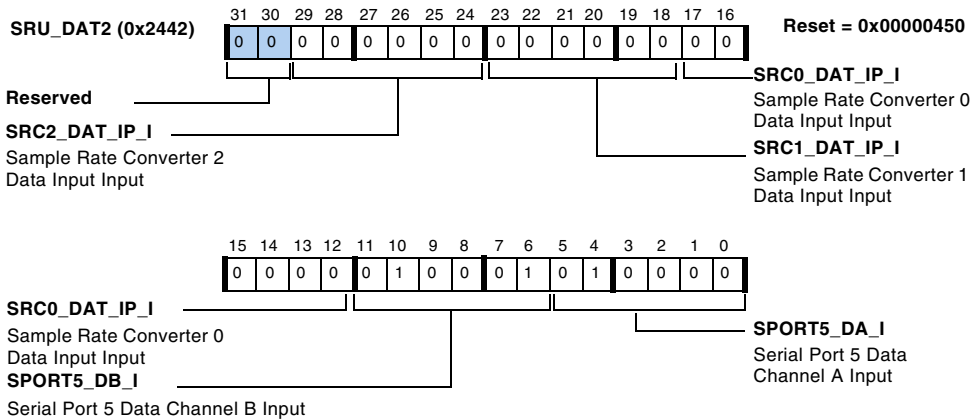


Figure 4-20. SRU_DAT2 Register

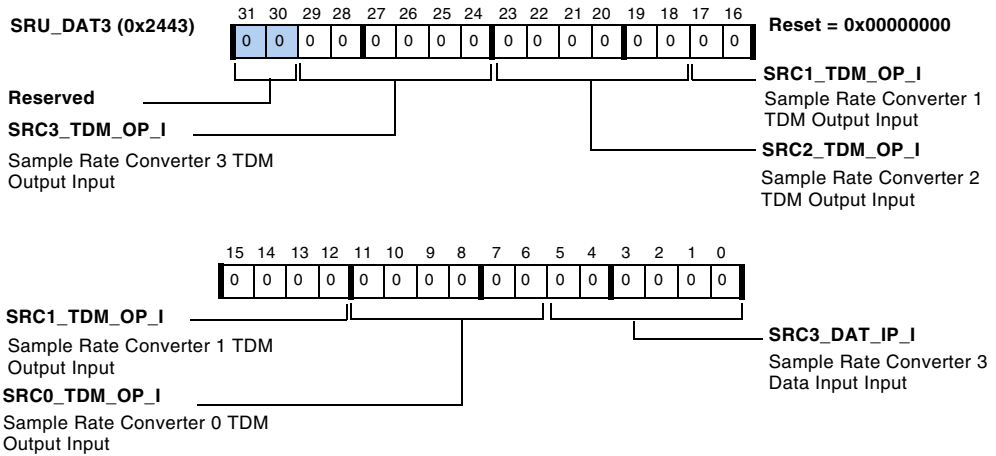


Figure 4-21. SRU_DAT3 Register

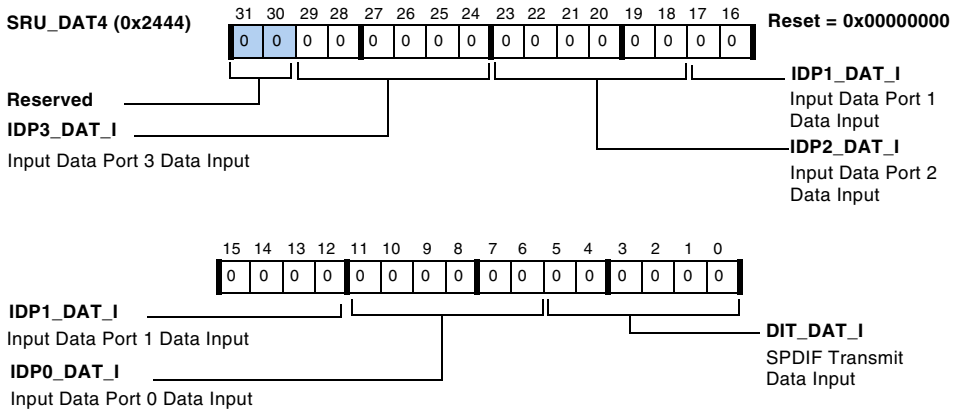


Figure 4-22. SRU_DAT4 Register

Making Connections in the SRUs

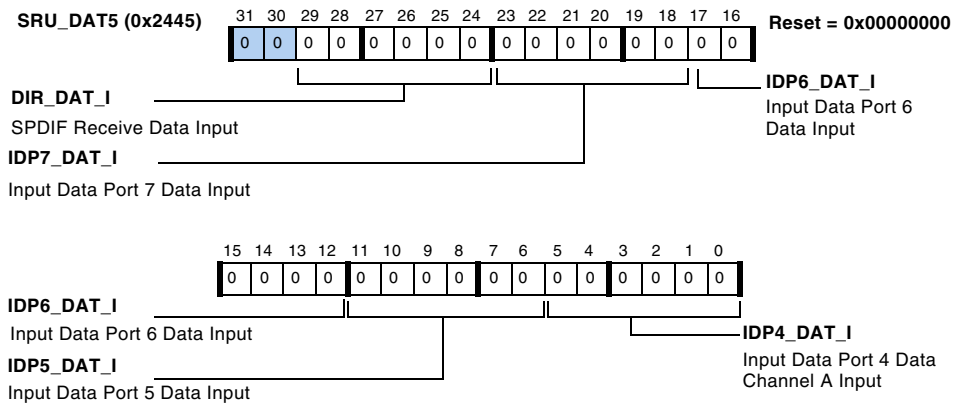


Figure 4-23. SRU_DAT5 Register

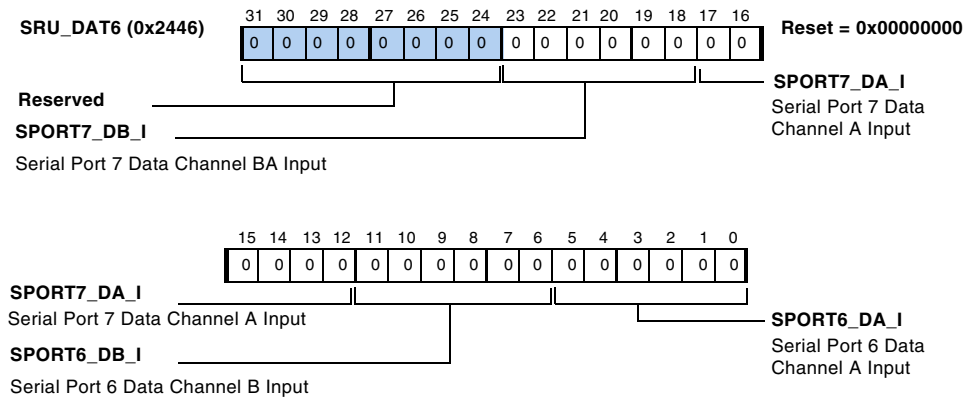


Figure 4-24. SRU_DAT6 Register

Table 4-5. Group B Sources—Serial Data

Selection Code	Source Signal	Description (Source)
000000 (0x0)	DAI_PB01_O	Select DAI pin buffer 1
000001 (0x1)	DAI_PB02_O	Select DAI pin buffer 2
000010 (0x2)	DAI_PB03_O	Select DAI pin buffer 3
000011 (0x3)	DAI_PB04_O	Select DAI pin buffer 4
000100 (0x4)	DAI_PB05_O	Select DAI pin buffer 5
000101 (0x5)	DAI_PB06_O	Select DAI pin buffer 6
000110 (0x6)	DAI_PB07_O	Select DAI pin buffer 7
000111 (0x7)	DAI_PB08_O	Select DAI pin buffer 8
001000 (0x8)	DAI_PB09_O	Select DAI pin buffer 9
001001 (0x9)	DAI_PB10_O	Select DAI pin buffer 10
001010 (0xA)	DAI_PB11_O	Select DAI pin buffer 11
001011 (0xB)	DAI_PB12_O	Select DAI pin buffer 12
001100 (0xC)	DAI_PB13_O	Select DAI pin buffer 13
001101 (0xD)	DAI_PB14_O	Select DAI pin buffer 14
001110 (0xE)	DAI_PB15_O	Select DAI pin buffer 15
001111 (0xF)	DAI_PB16_O	Select DAI pin buffer 16
010000 (0x10)	DAI_PB17_O	Select DAI pin buffer 17
010001 (0x11)	DAI_PB18_O	Select DAI pin buffer 18
010010 (0x12)	DAI_PB19_O	Select DAI pin buffer 19
010011 (0x13)	DAI_PB20_O	Select DAI pin buffer 20
010100 (0x14)	SPORT0_DA_O	Select SPORT 0A data
010101 (0x15)	SPORT0_DB_O	Select SPORT 0B data
010110 (0x16)	SPORT1_DA_O	Select SPORT 1A data
010111 (0x17)	SPORT1_DB_O	Select SPORT 1B data

Making Connections in the SRUs

Table 4-5. Group B Sources—Serial Data (Cont'd)

Selection Code	Source Signal	Description (Source)
011000 (0x18)	SPORT2_DA_O	Select SPORT 2A data
011001 (0x19)	SPORT2_DB_O	Select SPORT 2B data
011010 (0x1A)	SPORT3_DA_O	Select SPORT 3A data
011011 (0x1B)	SPORT3_DB_O	Select SPORT 3B data
011100 (0x1C)	SPORT4_DA_O	Select SPORT 4A data
011101 (0x1D)	SPORT4_DB_O	Select SPORT 4B data
011110 (0x1E)	SPORT5_DA_O	Select SPORT 5A data
011111 (0x1F)	SPORT5_DB_O	Select SPORT 5B data
100000 (0x20)	SRC0_DAT_OP_O	SRC0 data out (stereo)
100001 (0x21)	SRC1_DAT_OP_O	SRC1 data out (stereo)
100010 (0x22)	SRC2_DAT_OP_O	SRC2 data out (stereo)
100011 (0x23)	SRC3_DAT_OP_O	SRC3 data out (stereo)
100100 (0x24)	SRC0_TDM_IP_O	SRC0 data out
100101 (0x25)	SRC1_TDM_IP_O	SRC1 data out
100110 (0x26)	SRC2_TDM_IP_O	SRC2 data out
100111 (0x27)	SRC3_TDM_IP_O	SRC3 data out
101000 (0x28)	DIR_DAT_O	SPDIF_RX serial data out
101001 (0x29)	LOW	Select logic level low (0)
101010 (0x2A)	LOW	Select logic level low (0)
101011 (0x2B)	HIGH	Select logic level high (1)
101100 (0x2C)	SPORT6_DA_O	Select SPORT 6A data
101101 (0x2D)	SPORT6_DB_O	Select SPORT 6B data
101110 (0x2E)	SPORT7_DA_O	Select SPORT 7A data
101111 (0x2F)	SPORT7_DB_O	Select SPORT 7B data

Table 4-5. Group B Sources—Serial Data (Cont'd)

Selection Code	Source Signal	Description (Source)
110000 (0x30)	DIT_O	Select SPDIF biphas encoded output
110001 (0x31) –111101 (0x3D)		Reserved
111110 (0x3E)	LOW	Select logic level low (0)
111111 (0x3F)	HIGH	Select logic level high (1)

Group C Connections—Frame Sync Signals

Group C connections are used to route signals to frame sync inputs. These are the SPORT frame sync inputs (when the SPORT is in slave mode) and the frame sync inputs to the eight IDP channels. The SRCs and SPDIF transmitter and receiver are also selected from the list of group C sources and set in the group C registers.

Each of the frame sync inputs specified is connected to a frame sync source based on the 5-bit values described in the group C frame sync sources. Thirty-two possible frame sync sources can be connected using the SRU_FS0-4 registers. The registers are shown in [Figure 4-25](#) through [Figure 4-29](#). The input and output signals for group C are summarized in [Table 4-6 on page 4-35](#).



The following notes apply to group C connections.

1. SRU_FS0, SRU_FS1, and SRU_FS2 are 30-bit registers. On reads, bits 30 and 31 always return zero.
2. SRU_FS3 is a 24-bit register. On reads, bits 31 through 24 always return zero.
3. SRU_FS4 is a 10-bit register. On reads, bits 31 through 10 always return zero.

Making Connections in the SRUs

- SPORTs 6 and 7 receive frame syncs from other sources but cannot send their own frame syncs to other SPORTs or other peripherals internally through the SRU. If needed, they have to be connected externally through pins.

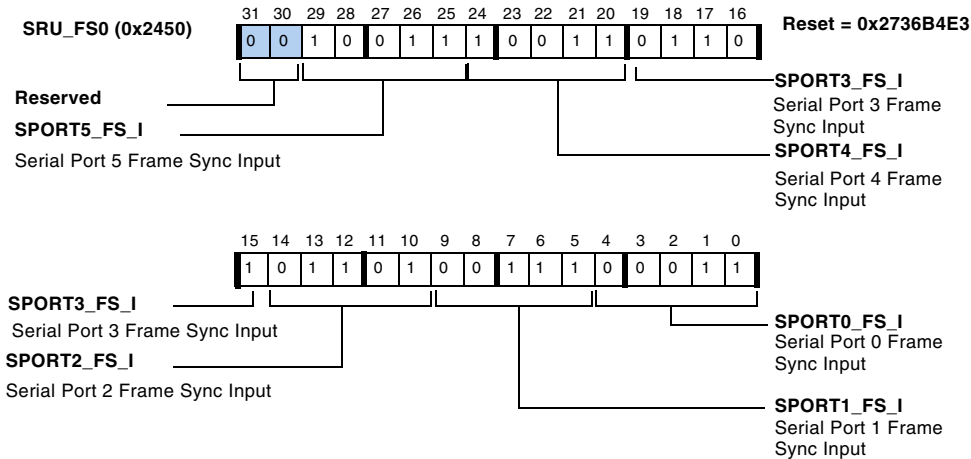


Figure 4-25. SRU_FS0 Register

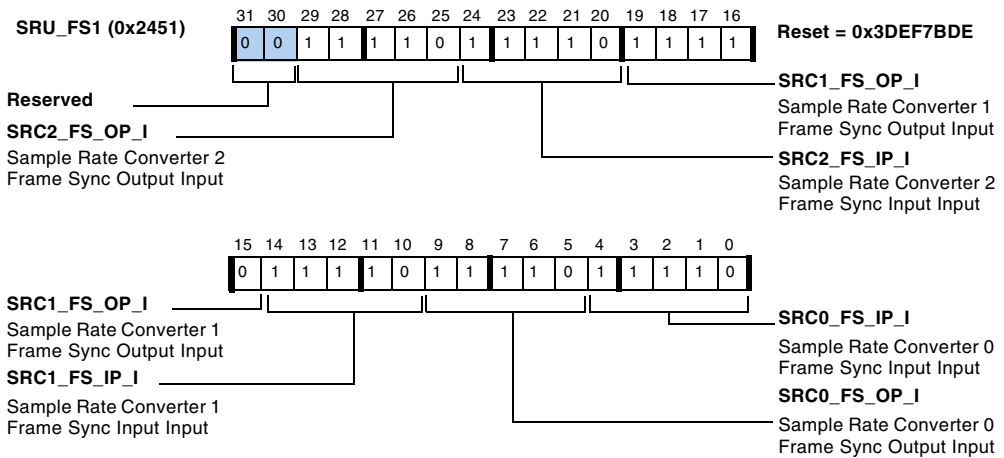


Figure 4-26. SRU_FS1 Register

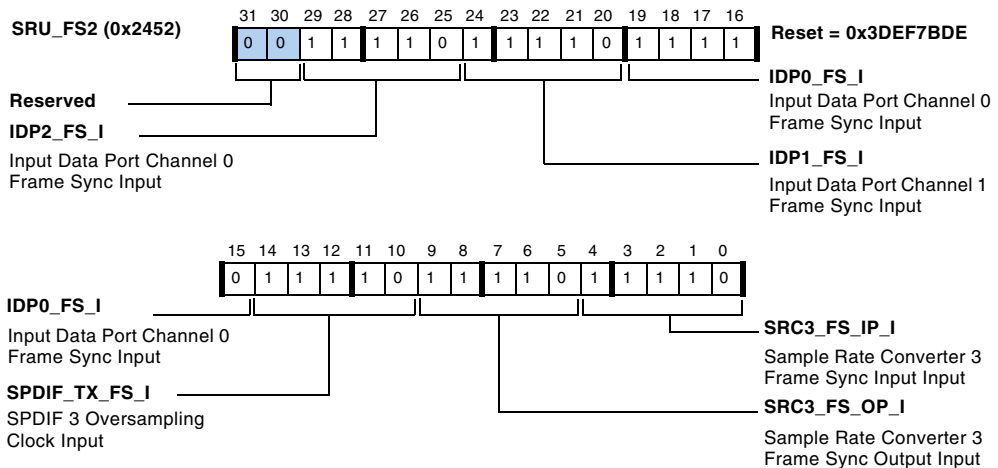


Figure 4-27. SRU_FS2 Register

Making Connections in the SRUs

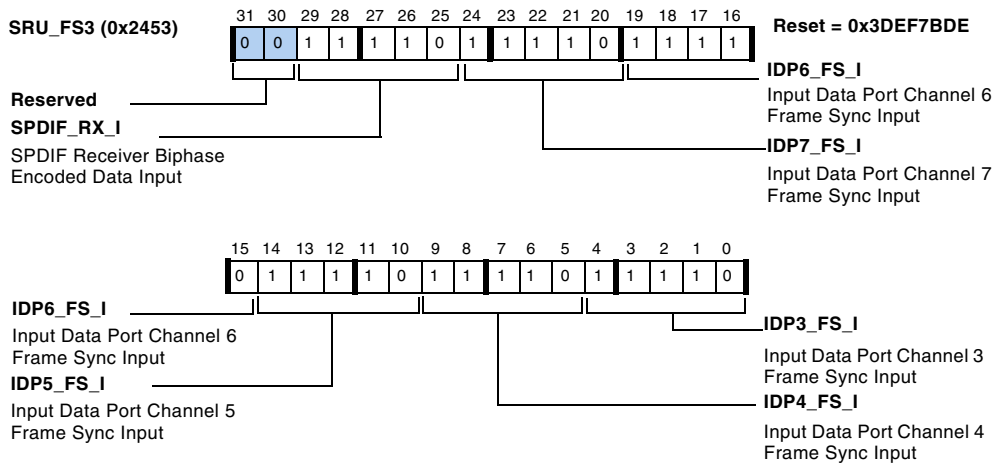


Figure 4-28. SRU_FS3 Register

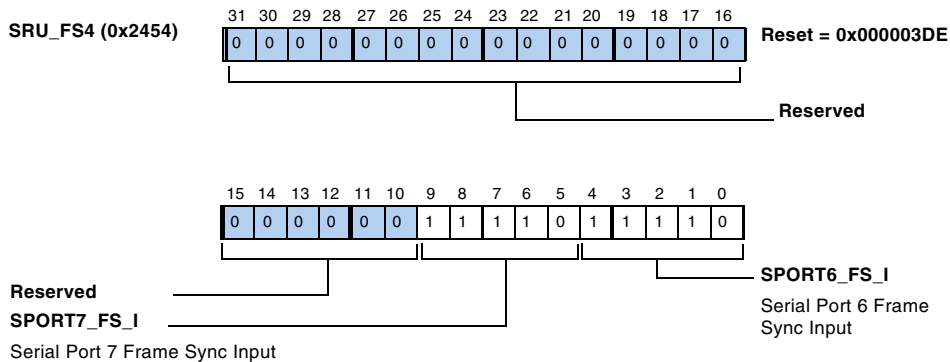


Figure 4-29. SRU_FS4 Register

Table 4-6. Group C Sources—Frame Sync

Selection Code	Source Signal	Description (Source)
00000 (0x0)	DAI_PB01_O	Select DAI pin buffer 1
00001 (0x1)	DAI_PB02_O	Select DAI pin buffer 2
00010 (0x2)	DAI_PB03_O	Select DAI pin buffer 3
00011 (0x3)	DAI_PB04_O	Select DAI pin buffer 4
00100 (0x4)	DAI_PB05_O	Select DAI pin buffer 5
00101 (0x5)	DAI_PB06_O	Select DAI pin buffer 6
00110 (0x6)	DAI_PB07_O	Select DAI pin buffer 7
00111 (0x7)	DAI_PB08_O	Select DAI pin buffer 8
01000 (0x8)	DAI_PB09_O	Select DAI pin buffer 9
01001 (0x9)	DAI_PB10_O	Select DAI pin buffer 10
01010 (0xA)	DAI_PB11_O	Select DAI pin buffer 11
01011 (0xB)	DAI_PB12_O	Select DAI pin buffer 12
01100 (0xC)	DAI_PB13_O	Select DAI pin buffer 13
01101 (0xD)	DAI_PB14_O	Select DAI pin buffer 14
01110 (0xE)	DAI_PB15_O	Select DAI pin buffer 15
01111 (0xF)	DAI_PB16_O	Select DAI pin buffer 16
10000 (0x10)	DAI_PB17_O	Select DAI pin buffer 17
10001 (0x11)	DAI_PB18_O	Select DAI pin buffer 18
10010 (0x12)	DAI_PB19_O	Select DAI pin buffer 19
10011 (0x13)	DAI_PB20_O	Select DAI pin buffer 20
10100 (0x14)	SPORT0_FS_O	Select SPORT 0 frame sync
10101 (0x15)	SPORT1_FS_O	Select SPORT 1 frame sync
10110 (0x16)	SPORT2_FS_O	Select SPORT 2 frame sync
10111 (0x17)	SPORT3_FS_O	Select SPORT 3 frame sync

Making Connections in the SRUs

Table 4-6. Group C Sources—Frame Sync (Cont'd)

Selection Code	Source Signal	Description (Source)
11000 (0x18)	SPORT4_FS_O	Select SPORT 4 frame sync
11001 (0x19)	SPORT5_FS_O	Select SPORT 5 frame sync
11010 (0x1A)	DIR_FS_O	SPDIF_RX frame sync output
11011 (0x1B)	Reserved	
11100 (0x1C)	PCG_FSA_O	Select precision frame sync A output
11101 (0x1D)	PCG_FSB_O	Select precision frame sync B output
11110 (0x1E)	LOW	Select logic level low (0)
11111 (0x1F)	HIGH	Select logic level high (1)

Group D Connections—Pin Signal Assignments

Group D is used to specify any signals that are driven off chip by the pin buffers. A pin buffer input (DAI_PBxx_I) is driven as an output from the processor when the pin buffer enable is set (= 1).

Each physical pin (connected to a bonded pad) may be routed via the SRU to any of the outputs of the DAI audio peripherals, based on the 7-bit values listed in [Table 4-7 on page 4-39](#). The SRU may also be used to route signals that control the pins in other ways. These signals may be configured for use as general-purpose I/O, precision clock generators, or miscellaneous control signals. The registers are shown in [Figure 4-30](#) through [Figure 4-34](#).



The following notes apply to the group D sources.

1. Setting SRU_PIN4[28] to high inverts the level of DAI_P19_I and setting SRU_PIN4[29] inverts the level of DAI_P20_I.
2. If SRU_PIN4[20:14] = 0x12 then setting SRU_PIN4[28] to high does not invert the output.

3. If $SRU_PIN4[27:21] = 0x13$, then setting $SRU_PIN4[29]$ to high does not invert the output.
4. SRU_PIN0 , SRU_PIN1 , SRU_PIN2 , and SRU_PIN3 are 28-bit registers. Reads on bits 28 through 31 always return zero.
5. SRU_PIN4 is a 30-bit register. Reads on bits 31 and 30 always return zero.

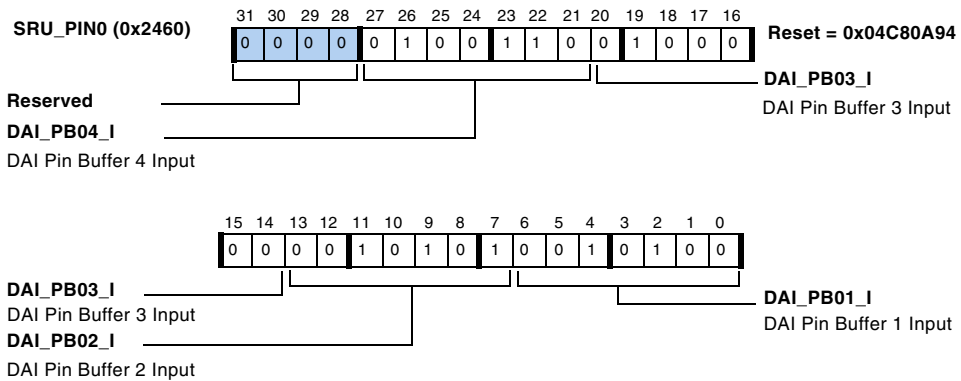


Figure 4-30. SRU_PIN0 Register

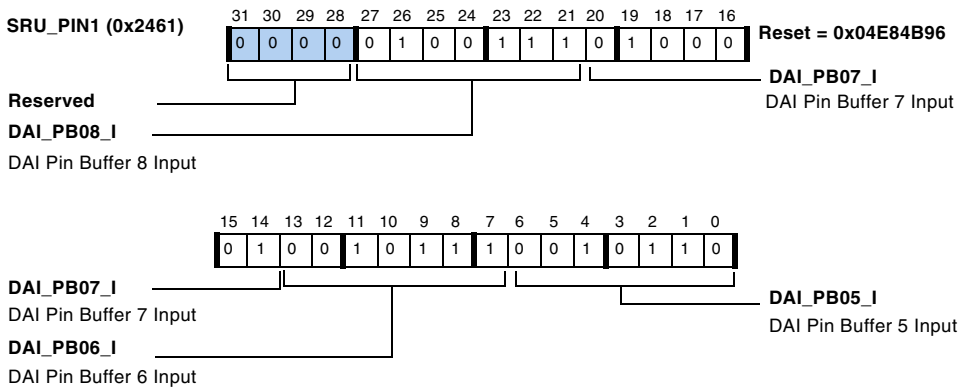


Figure 4-31. SRU_PIN1 Register

Making Connections in the SRUs

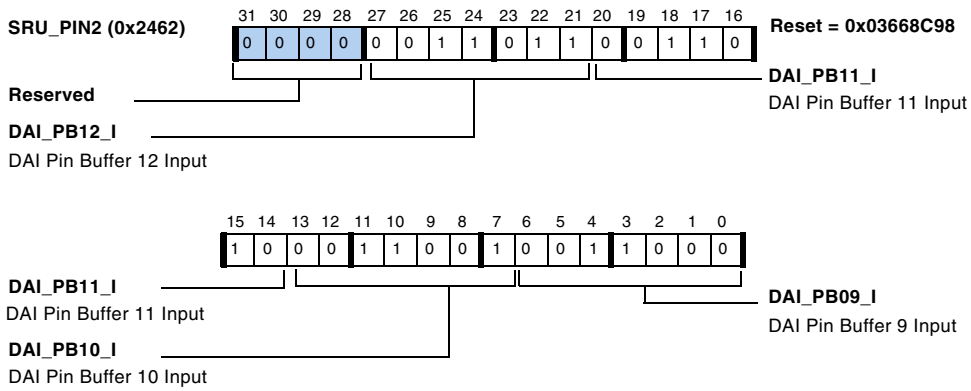


Figure 4-32. SRU_PIN2 Register

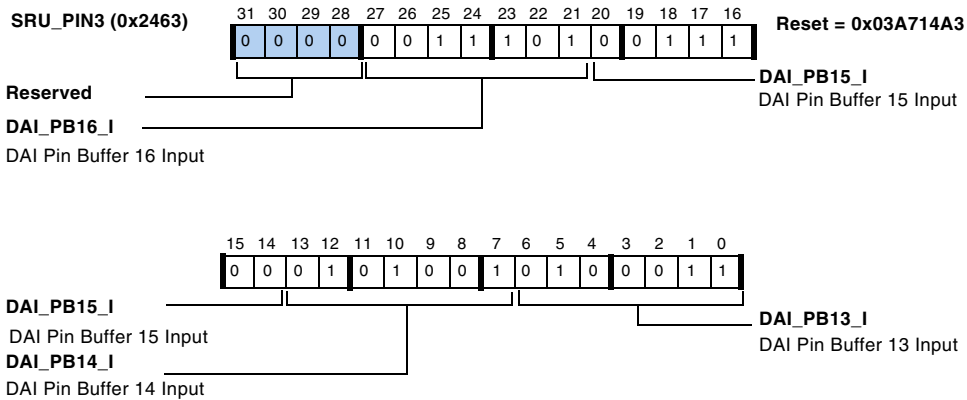


Figure 4-33. SRU_PIN3 Register

Digital Audio/Digital Peripheral Interfaces

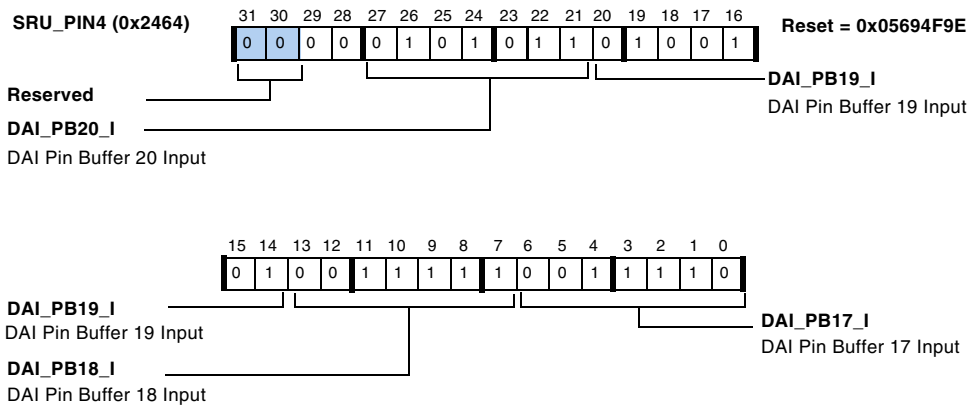


Figure 4-34. SRU_PIN4 Register

Table 4-7. Group D Sources—Pin Signal Assignments

Selection Code	Source Signal	Description (Source)
0000000 (0x0)	DAI_PB01_O	Select DAI pin buffer 1
0000001 (0x1)	DAI_PB02_O	Select DAI pin buffer 2
0000010 (0x2)	DAI_PB03_O	Select DAI pin buffer 3
0000011 (0x3)	DAI_PB04_O	Select DAI pin buffer 4
0000100 (0x4)	DAI_PB05_O	Select DAI pin buffer 5
0000101 (0x5)	DAI_PB06_O	Select DAI pin buffer 6
0000110 (0x6)	DAI_PB07_O	Select DAI pin buffer 7
0000111 (0x7)	DAI_PB08_O	Select DAI pin buffer 8
0001000 (0x8)	DAI_PB09_O	Select DAI pin buffer 9
0001001 (0x9)	DAI_PB10_O	Select DAI pin buffer 10
0001010 (0xA)	DAI_PB11_O	Select DAI pin buffer 11
0001011 (0xB)	DAI_PB12_O	Select DAI pin buffer 12
0001100 (0xC)	DAI_PB13_O	Select DAI pin buffer 13

Making Connections in the SRUs

Table 4-7. Group D Sources—Pin Signal Assignments (Cont'd)

Selection Code	Source Signal	Description (Source)
0001101 (0xD)	DAI_PB14_O	Select DAI pin buffer 14
0001110 (0xE)	DAI_PB15_O	Select DAI pin buffer 15
0001111 (0xF)	DAI_PB16_O	Select DAI pin buffer 16
0010000 (0x10)	DAI_PB17_O	Select DAI pin buffer 17
0010001 (0x11)	DAI_PB18_O	Select DAI pin buffer 18
0010010 (0x12)	DAI_PB19_O	Select DAI pin buffer 19
0010011 (0x13)	DAI_PB20_O	Select DAI pin buffer 20
0010100 (0x14)	SPORT0_DA_O	Select SPORT 0A data
0010101 (0x15)	SPORT0_DB_O	Select SPORT 0B data
0010110 (0x16)	SPORT1_DA_O	Select SPORT 1A data
0010111 (0x17)	SPORT1_DB_O	Select SPORT 1B data
0011000 (0x18)	SPORT2_DA_O	Select SPORT 2A data
0011001 (0x19)	SPORT2_DB_O	Select SPORT 2B data
0011010 (0x1A)	SPORT3_DA_O	Select SPORT 3A data
0011011 (0x1B)	SPORT3_DB_O	Select SPORT 3B data
0011100 (0x1C)	SPORT4_DA_O	Select SPORT 4A data
0011101 (0x1D)	SPORT4_DB_O	Select SPORT 4B data
0011110 (0x1E)	SPORT5_DA_O	Select SPORT 5A data
0011111 (0x1F)	SPORT5_DB_O	Select SPORT 5B data
0100000 (0x20)	SPORT0_CLK_O	Select SPORT 0 clock
0100001 (0x21)	SPORT1_CLK_O	Select SPORT 1 clock
0100010 (0x22)	SPORT2_CLK_O	Select SPORT 2 clock
0100011 (0x23)	SPORT3_CLK_O	Select SPORT 3 clock
0100100 (0x24)	SPORT4_CLK_O	Select SPORT 4 clock

Table 4-7. Group D Sources—Pin Signal Assignments (Cont'd)

Selection Code	Source Signal	Description (Source)
0100101 (0x25)	SPORT5_CLK_O	Select SPORT 5 clock
0100110 (0x26)	SPORT0_FS_O	Select SPORT 0 frame sync
0100111 (0x27)	SPORT1_FS_O	Select SPORT 1 frame sync
0101000 (0x28)	SPORT2_FS_O	Select SPORT 2 frame sync
0101001 (0x29)	SPORT3_FS_O	Select SPORT 3 frame sync
0101010 (0x2A)	SPORT4_FS_O	Select SPORT 4 frame sync
0101011 (0x2B)	SPORT5_FS_O	Select SPORT 5 frame sync
0101100 (0x2C)	SPORT6_DA_O	Select SPORT 6A data
0101101 (0x2D)	SPORT6_DB_O	Select SPORT 6B data
0101110 (0x2E)	SPORT7_DA_O	Select SPORT 7A data
0101111 (0x2F)	SPORT7_DB_O	Select SPORT 7B data
0110000 (0x30)	PDAP_STRB_O	Select PDAP data transfer request strobe
0110001–0110011	Reserved	
0110100 (0x34)	SPORT6_CLK_O	Select SPORT 6 clock
0110101 (0x35)	SPORT7_CLK_O	Select SPORT 7 clock
0110110 (0x36)	SPORT6_FS_O	Select SPORT 6 frame sync
0110111 (0x37)	SPORT7_FS_O	Select SPORT 7 frame sync
0111000 (0x38)	PCG_CLKA_O	Select precision clock A
0111001 (0x39)	PCG_CLKB_O	Select precision clock B
0111010 (0x3A)	PCG_FSA_O	Select precision frame sync A
0111011 (0x3B)	PCG_FSB_O	Select precision frame sync B
0111100 (0x3C)	Reserved	
0111101 (0x3D)	SRC0_DAT_OP_O	SRC0 data output
0111110 (0x3E)	SRC1_DAT_OP_O	SRC1 data output

Making Connections in the SRUs

Table 4-7. Group D Sources—Pin Signal Assignments (Cont'd)

Selection Code	Source Signal	Description (Source)
0111111 (0x3F)	SRC2_DAT_OP_O	SRC2 data output
1000000 (0x40)	SRC3_DAT_OP_O	SRC3 data output
1000001 (0x41)	DIR_DAT_O	SPDIF_RX data output
1000010 (0x42)	DIR_FS_O	SPDIF_RX frame sync output
1000011 (0x43)	DIR_CLK_O	SPDIF_RX clock output
1000100 (0x44)	DIR_TDMCLK_O	SPDIF_RX TDM clock output
1000101 (0x45)	DIT_DAT_O	SPDIF_TX data output
1000110 (0x46)	SPORT0_TDV_O	SPORT0 transmit data valid output
1000111 (0x47)	SPORT1_TDV_O	SPORT0 transmit data valid output
1001000 (0x48)	SPORT2_TDV_O	SPORT0 transmit data valid output
1001001 (0x49)	SPORT3_TDV_O	SPORT0 transmit data valid output
1001010 (0x4A)	SPORT4_TDV_O	SPORT0 transmit data valid output
1001011 (0x4B)	SPORT5_TDV_O	SPORT0 transmit data valid output
1001100 (0x4C)	SPORT6_TDV_O	SPORT0 transmit data valid output
1001101 (0x4D)	SPORT7_TDV_O	SPORT0 transmit data valid output
1001110 (0x4E)	DIR_LRCLK_FB	External PLL – feedback point connection
1001111 (0x4F)	DIR_LRCLK_REF	External PLL – reference point connection
1010000 (0x50)	PCG_CLKC_O	Select precision clock C
1011001 (0x51)	PCG_CLKD_O	Select precision clock D
1011010 (0x52)	PCG_FSC_O	Select precision frame sync C
1010011 (0x53)	PCG_FSD_O	Select precision frame sync D
1010100 – 1111101	Reserved	
1111110 (0x7E)	LOGIC_LEVEL_LOW	Logic level low (0)
1111111 (0x7F)	LOGIC_LEVEL_HIGH	Logic level high (1)

Group E Connections—Interrupts and Miscellaneous Signals

Group E connections, shown in [Table 4-8 on page 4-45](#), are slightly different from the others in that the inputs and outputs being routed vary considerably in function. This group routes control signals (interrupts and miscellaneous signals) and provides a means of connecting signals between groups. Signals with names such as `MISCxy` appear as inputs in group E, but do not directly feed any peripheral. Rather, they reappear as outputs in group D and group F. The registers for this group are shown in [Figure 4-35](#) and [Figure 4-36](#).

Additional connections among groups D, E, and F provide a surprising amount of utility. Since group D routes signals off chip and group F dictates pin direction, these few signal paths enable an enormous number of possible uses and connections for DAI pins. A few examples include:

- One pin's input can be patched to another pin's output, allowing board-level routing under software control.
- A pin input can be patched to another pin's enable, allowing an off-chip signal to gate an output from the processor.
- Any of the DAI pins can be used as interrupt sources or general-purpose I/O (GPIO) signals.
- Many types of bidirectional signaling may be created by routing an output of the PCG to a pin enable.

The SRUs enable many possible functional changes, both internally and externally. Used creatively, the SRUs allow system designers to radically change functionality at run time, and to potentially reuse circuit boards across many products.

Making Connections in the SRUs



The following notes apply to group E connections.

1. SRU_EXT_MISCB is a 30-bit register. On reads, bits 30 and 31 always return zero.
2. A detailed description of the DAI interrupt register and its usage is provided in [“DAI Interrupt Controller Registers” on page A-112](#).
3. Setting SRU_MISCA[30] to high inverts the level of MISCA4_I, and setting SRU_MISCA[31] to high inverts the level of MISCA5_I.

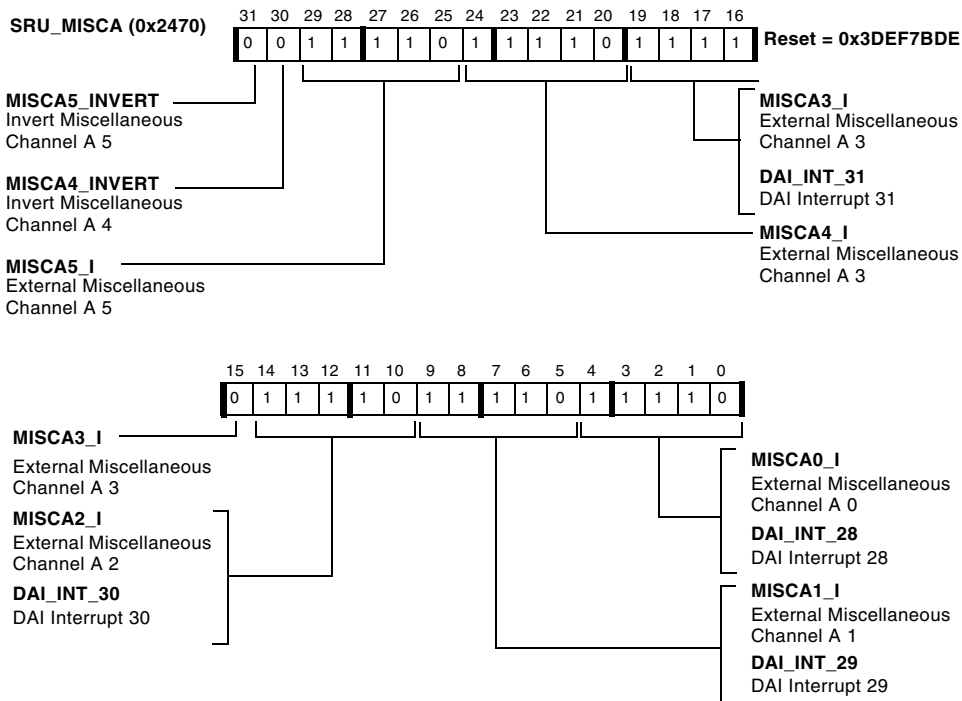


Figure 4-35. SRU_MISCA Register

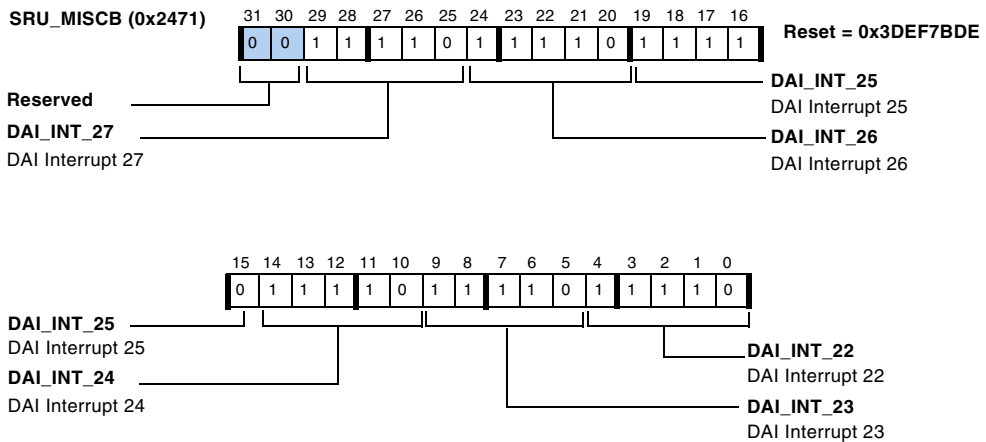


Figure 4-36. SRU_MISC B Register

Table 4-8. Group E Sources—Miscellaneous Signals

Selection Code	Source Signal	Description (Output Source)
00000 (0x0)	DAI_PB01_O	Select DAI pin buffer 1 output
00001 (0x1)	DAI_PB02_O	Select DAI pin buffer 2 output
00010 (0x2)	DAI_PB03_O	Select DAI pin buffer 3 output
00011 (0x3)	DAI_PB04_O	Select DAI pin buffer 4 output
00100 (0x4)	DAI_PB05_O	Select DAI pin buffer 5 output
00101 (0x5)	DAI_PB06_O	Select DAI pin buffer 6 output
00110 (0x6)	DAI_PB07_O	Select DAI pin buffer 7 output
00111 (0x7)	DAI_PB08_O	Select DAI pin buffer 8 output
01000 (0x8)	DAI_PB09_O	Select DAI pin buffer 9 output
01001 (0x9)	DAI_PB10_O	Select DAI pin buffer 10 output
01010 (0xA)	DAI_PB11_O	Select DAI pin buffer 11 output
01011 (0xB)	DAI_PB12_O	Select DAI pin buffer 12 output

Making Connections in the SRUs

Table 4-8. Group E Sources—Miscellaneous Signals (Cont'd)

Selection Code	Source Signal	Description (Output Source)
01100 (0xC)	DAI_PB13_O	Select DAI pin buffer 13 output
01101 (0xD)	DAI_PB14_O	Select DAI pin buffer 14 output
01110 (0xE)	DAI_PB15_O	Select DAI pin buffer 15 output
01111 (0xF)	DAI_PB16_O	Select DAI pin buffer 16 output
10000 (0x10)	DAI_PB17_O	Select DAI pin buffer 17 output
10001 (0x11)	DAI_PB18_O	Select DAI pin buffer 18 output
10010 (0x12)	DAI_PB19_O	Select DAI pin buffer 19 output
10011 (0x13)	DAI_PB20_O	Select DAI pin buffer 20 output
10100 (0x14)	SPORT0_F0_O	Select serial port0 frame sync
10101 (0x15)	SPORT1_FS_O	Select serial port1 frame sync
10110 (0x16)	SPORT2_FS_O	Select serial port2 frame sync
10111 (0x17)	SPORT3_FS_O	Select serial port3 frame sync
11000 (0x18)	SPORT4_FS_O	Select serial port4 frame sync
11001 (0x19)	SPORT5_FS_O	Select serial port5 frame sync
11010 (0x1A)	PCG_CLKA_O	Select precision clock A
11011 (0x1B)	PCG_FSA_O	Select precision frame sync A
11100 (0x1C)	PCG_CLKB_O	Select precision clock B
11101 (0x1D)	PCG_FSB_O	Select precision frame sync B
11110 (0x1E)	LOW	Select logic level low (0) as a source
11111 (0x1F)	HIGH	Select logic level high (1) as a source

Group F—Pin Enable Signals

Group F signals, shown in Figure 4-37 through Figure 4-40 and described in Table 4-9, are used to specify whether each DAI pin is used as an output or an input by setting the source for the pin buffer enables. When a pin buffer enable (DAI_PBEN $\times\times$ _I) is set (= 1) the signal present at the corresponding pin buffer input (DAI_PB $\times\times$ _I) is driven off chip as an output. When a pin buffer enable is cleared (= 0) the signal present at the corresponding pin buffer input is ignored.

The pin enable control registers activate the drive buffer for each of the 20 DAI pins. When the pins are not enabled (driven), they can be used as inputs.



The following note applies to the group F connections.

SRU_PBEN0, SRU_PBEN1, SRU_PBEN2 and SRU_PBEN3 are 30 bit registers. On reads, bits 30 and 31 always return zero.

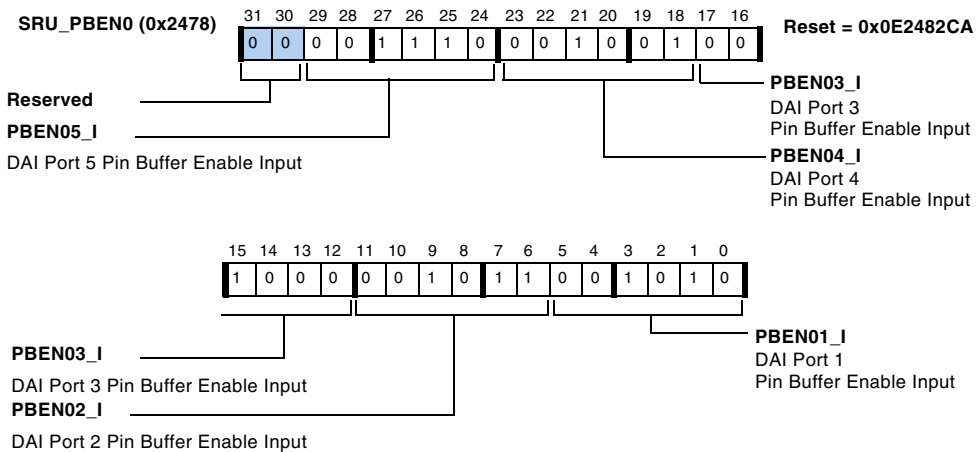


Figure 4-37. SRU_PBEN0

Making Connections in the SRUs

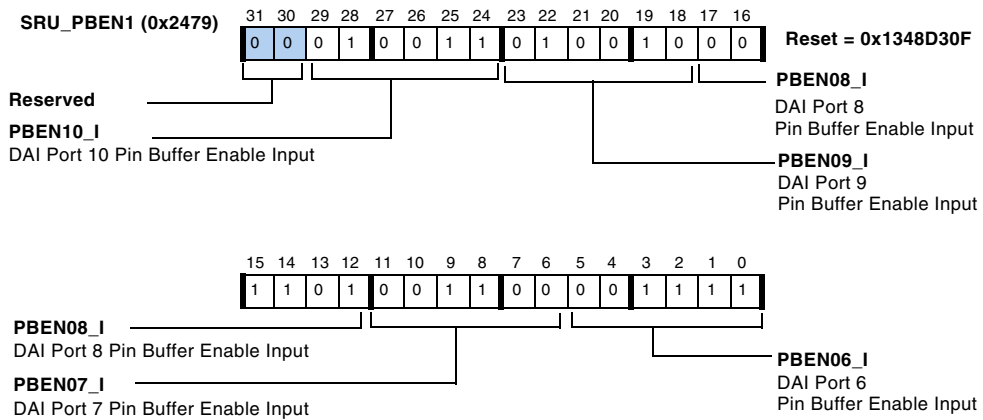


Figure 4-38. SRU_PBEN1

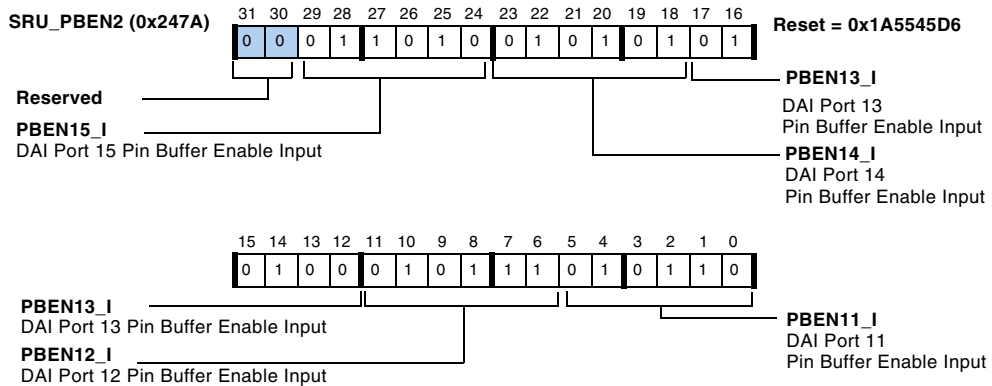


Figure 4-39. SRU_PBEN2

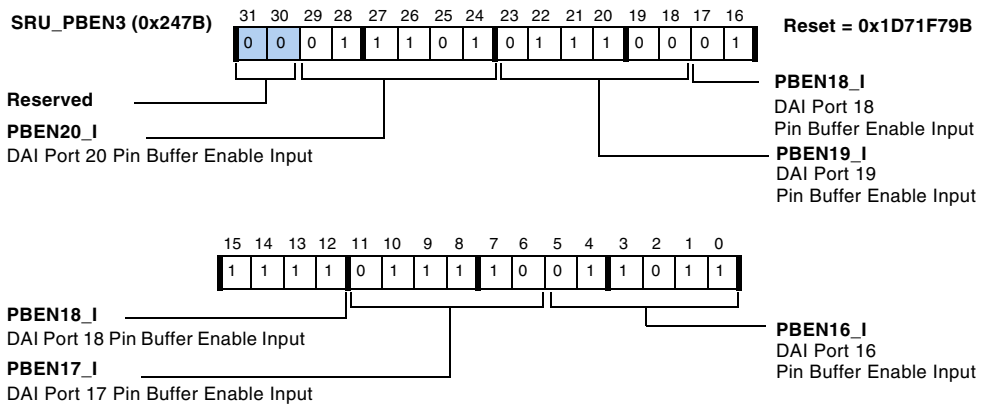


Figure 4-40. SRU_PBEN3

Table 4-9. Group F Sources—Pin Output Enable

Selection Code	Source Signal	Description (Output Source)
000000 (0x0)	LOW	Select logic level low (0)
000001 (0x1)	HIGH	Select logic level high (1)
000010 (0x2)	MISCA0_O	Assign miscellaneous control A0 output to a pin
000011 (0x3)	MISCA1_O	Assign miscellaneous control A1 output to a pin
000100 (0x4)	MISCA2_O	Assign miscellaneous control A2 output to a pin
000101 (0x5)	MISCA3_O	Assign miscellaneous control A3 output to a pin
000110 (0x6)	MISCA4_O	Assign miscellaneous control A4 output to a pin
000111 (0x7)	MISCA5_O	Assign miscellaneous control A5 output to a pin
001000 (0x8)	SPORT0_CLK_PBEN_O	Select serial port 0 clock output enable
001001 (0x9)	SPORT0_FS_PBEN_O	Select serial port 0 frame sync output enable
001010 (0xA)	SPORT0_DA_PBEN_O	Select serial port 0 data channel A output enable
001011 (0xB)	SPORT0_DB_PBEN_O	Select serial port 0 data channel B output enable
001100 (0xC)	SPORT1_CLK_PBEN_O	Select serial port 1 clock output enable

Making Connections in the SRUs

Table 4-9. Group F Sources—Pin Output Enable (Cont'd)

Selection Code	Source Signal	Description (Output Source)
001101 (0xD)	SPORT1_FS_PBEN_O	Select serial port 1 frame sync output enable
001110 (0xE)	SPORT1_DA_PBEN_O	Select serial port 1 data channel A output enable
001111 (0xF)	SPORT1_DB_PBEN_O	Select serial port 1 data channel B output enable
010000 (0x10)	SPORT2_CLK_PBEN_O	Select serial port 2 clock output enable
010001 (0x11)	SPORT2_FS_PBEN_O	Select serial port 2 frame sync output enable
010010 (0x12)	SPORT2_DA_PBEN_O	Select serial port 2 data channel A output enable
010011 (0x13)	SPORT2_DB_PBEN_O	Select serial port 2 data channel B output enable
010100 (0x14)	SPORT3_CLK_PBEN_O	Select serial port 3 clock output enable
010101 (0x15)	SPORT3_FS_PBEN_O	Select serial port 3 frame sync output enable
010110 (0x16)	SPORT3_DA_PBEN_O	Select serial port 3 data channel A output enable
010111 (0x17)	SPORT3_DB_PBEN_O	Select serial port 3 data channel B output enable
011000 (0x18)	SPORT4_CLK_PBEN_O	Select serial port 4 clock output enable
011001 (0x19)	SPORT4_FS_PBEN_O	Select serial port 4 frame sync output enable
011010 (0x1A)	SPORT4_DA_PBEN_O	Select serial port 4 data channel A output enable
011011 (0x1B)	SPORT4_DB_PBEN_O	Select serial port 4 data channel B output enable
011100 (0x1C)	SPORT5_CLK_PBEN_O	Select serial port 5 clock output enable
011101 (0x1D)	SPORT5_FS_PBEN_O	Select serial port 5 frame sync output enable
011110 (0x1E)	SPORT5_DA_PBEN_O	Select serial port 5 data channel A output enable
011111 (0x1F)	SPORT5_DB_PBEN_O	Select serial port 5 data channel B output enable
100000 (0x20)	SPORT6_CLK_PBEN_O	Select serial port 6 clock output enable
100001 (0x21)	SPORT6_FS_PBEN_O	Select serial port 6 frame sync output enable
100010 (0x22)	SPORT6_DA_PBEN_O	Select serial port 6 data channel A output enable
100011 (0x23)	SPORT6_DB_PBEN_O	Select serial port 6 data channel B output enable
100100 (0x24)	SPORT7_CLK_PBEN_O	Select serial port 7 clock output enable

Table 4-9. Group F Sources—Pin Output Enable (Cont'd)

Selection Code	Source Signal	Description (Output Source)
100101 (0x25)	SPORT7_FS_PBEN_O	Select serial port 7 frame sync output enable
100110 (0x26)	SPORT7_DA_PBEN_O	Select serial port 7 data channel A output enable
100111 (0x27)	SPORT7_DB_PBEN_O	Select serial port 7 data channel B output enable
101000 (0x28)	SPORT0_TDV_PBEN_O	Select serial port 0 transmit data valid output
101001 (0x29)	SPORT1_TDV_PBEN_O	Select serial port 1 transmit data valid output
101010 (0x2A)	SPORT2_TDV_PBEN_O	Select serial port 2 transmit data valid output
101011 (0x2B)	SPORT3_TDV_PBEN_O	Select serial port 3 transmit data valid output
101100 (0x2C)	SPORT4_TDV_PBEN_O	Select serial port 4 transmit data valid output
101101 (0x2D)	SPORT5_TDV_PBEN_O	Select serial port 5 transmit data valid output
100111 (0x2E)	SPORT6_TDV_PBEN_O	Select serial port 6 transmit data valid output
101110 (0x2F)	SPORT7_TDV_PBEN_O	Select serial port 7 transmit data valid output
110000 (0x30)–111111 (0x3F)		Reserved

DPI/SRU2 Connection Groups

The DPI/SRU2 has the following default configurations where seven bits are for SPI1 booting, two bits are for UART1, two bits are for the TWI, and the remaining three bits are for the timers.

Table 4-10. DPI/SRU2 Default Configuration

Pin Number	Signal	Pin Number	Signal
DPI_01	SPIMOSI	DPI_08	SPIFLG3
DPI_02	SPIMISO	DPI_09	UART0_TX
DPI_03	SPICLK	DPI_10	UART0_RX
DPI_04	SPIDS	DPI_11	TWI_SDATA
DPI_05	SPIFLG0	DPI_12	TWI_SCLK

Making Connections in the SRUs

Table 4-10. DPI/SRU2 Default Configuration (Cont'd)

Pin Number	Signal	Pin Number	Signal
DPI_06	SPIFLG1	DPI_13	TIMER0_O
DPI_07	SPIFLG2	DPI_14	TIMER1_O

There are three separate groups of connections that are used in SRU2. The following sections summarize each.

Group A Connections—Input Routing Signals

Group A is used to route the 14 external pin signals, timer, and UART outputs to the inputs of the other peripherals. Unlike the DAI SRU, all functions, such as serial clock and data, are combined into the same group A connections.

All clock inputs that are not used should be set to logic low. The registers and input signals for group A are summarized in [Figure 4-41](#) through [Figure 4-46](#) and [Table 4-11](#).

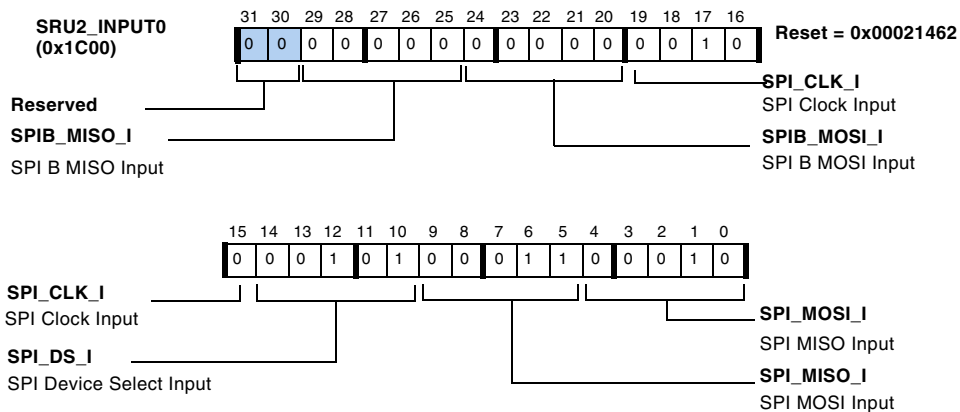


Figure 4-41. SRU2_INPUT0 Register

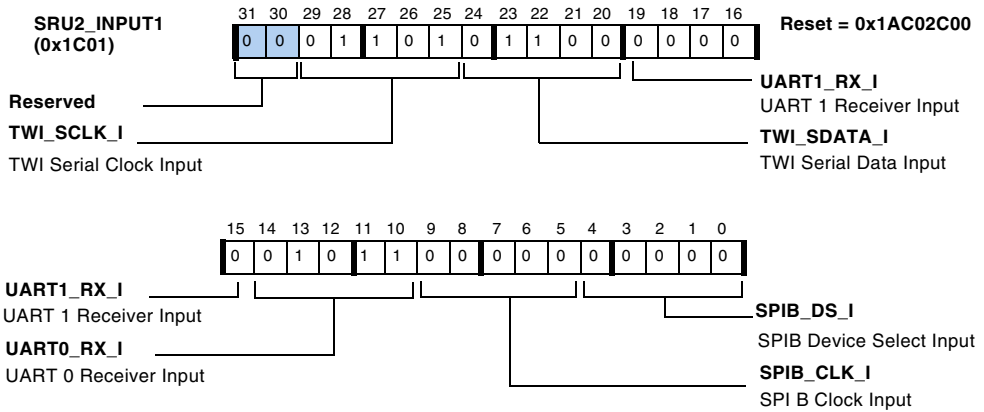


Figure 4-42. SRU2_INPUT1 Register

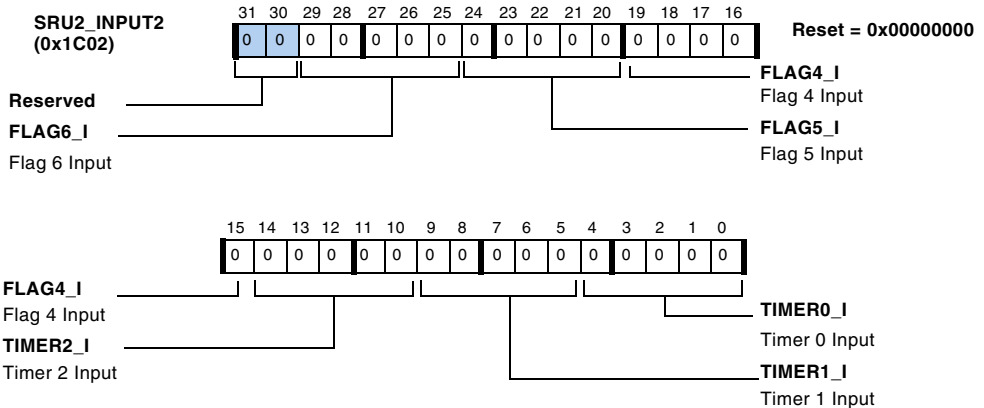


Figure 4-43. SRU2_INPUT2 Register

Making Connections in the SRUs

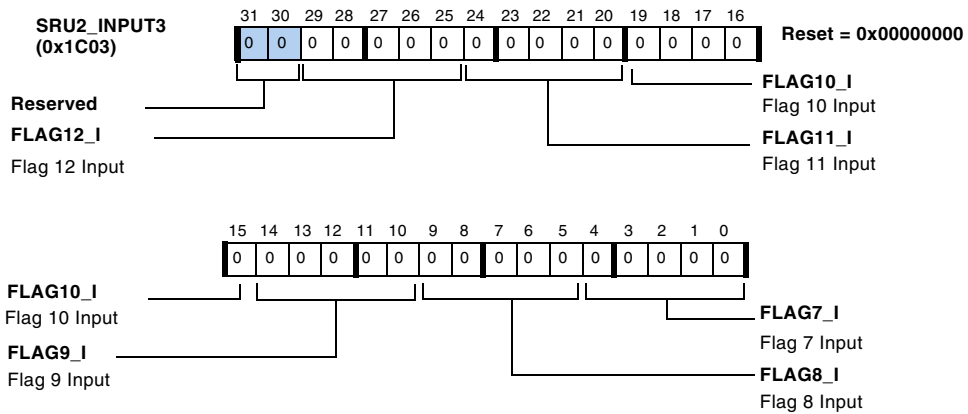


Figure 4-44. SRU2_INPUT3 Register

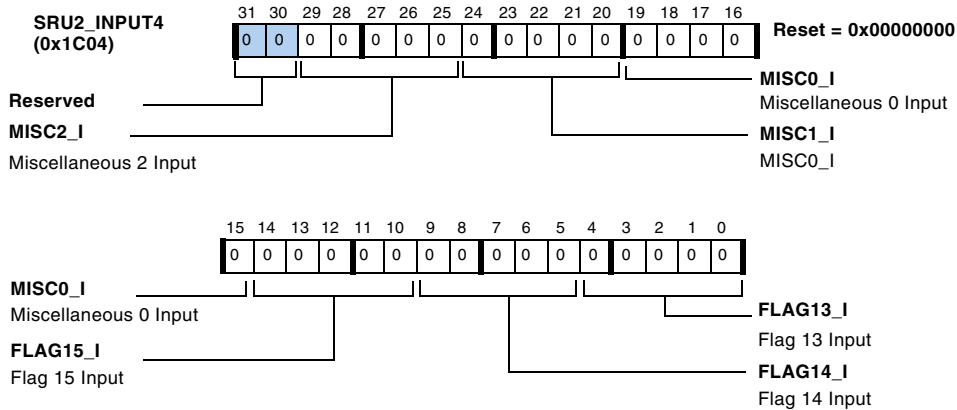


Figure 4-45. SRU2_INPUT4 Register

Digital Audio/Digital Peripheral Interfaces

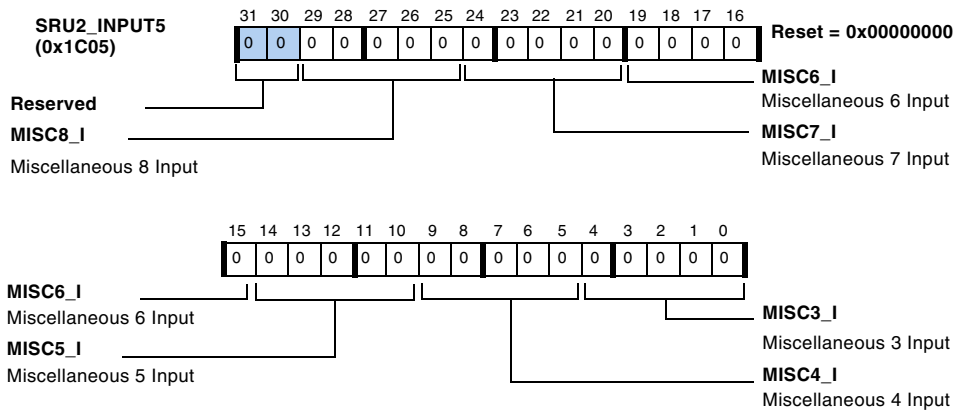


Figure 4-46. SRU2_INPUT5 Register

Table 4-11. Group A Connections

Binary	Decimal	Signal	Description
00000	00	LOGIC_LEVEL_LOW	Logic level low (0)
00001	01	LOGIC_LEVEL_HIGH	logic level high (1)
00010	02	DPI_P01_O	External pin 1
00011	03	DPI_P02_O	External pin 2
00100	04	DPI_P03_O	External pin 3
00101	05	DPI_P04_O	External pin 4
00110	06	DPI_P05_O	External pin 5
00111	07	DPI_P06_O	External pin 6
01000	08	DPI_P07_O	External pin 7
01001	09	DPI_P08_O	External pin 8
01010	10	DPI_P09_O	External pin 9
01011	11	DPI_P10_O	External pin 10
01100	12	DPI_P11_O	External pin 11

Making Connections in the SRUs

Table 4-11. Group A Connections (Cont'd)

Binary	Decimal	Signal	Description
01101	13	DPI_P12_O	External pin 12
01110	14	DPI_P13_O	External pin 13
01111	15	DPI_P14_O	External pin 14
10000	16	TIMER0_O	Timer0 output
10001	17	TIMER1_O	Timer1 output
10010	18	TIMER2_O	Timer2 output
10011	19	UART0_TX_O	UART0 transmitter output
10100	20	UART1_TX_O	UART1 transmitter output
10101-11111	21-31	RESERVED	RESERVED

Group B Connections—Pin Assignment Signals

Group B connections, shown in [Figure 4-47](#) through [Figure 4-49](#) and [Table 4-12](#), are used to route output signals to the 14 DPI pins.

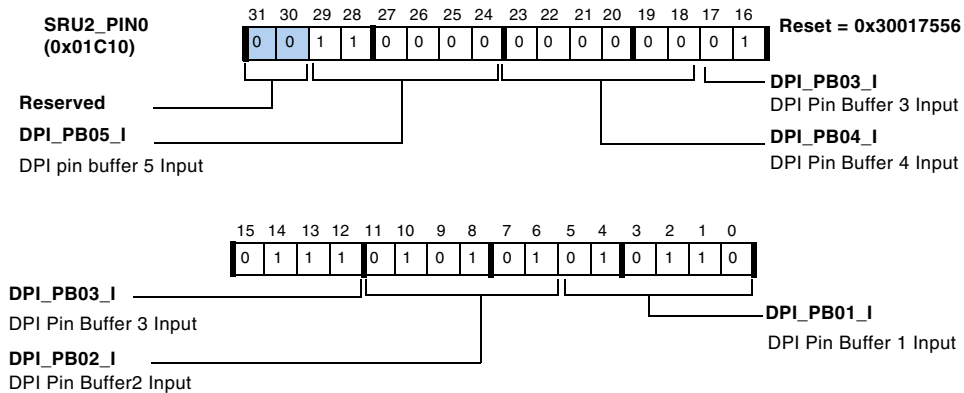


Figure 4-47. SRU2_PIN0 Register

Digital Audio/Digital Peripheral Interfaces

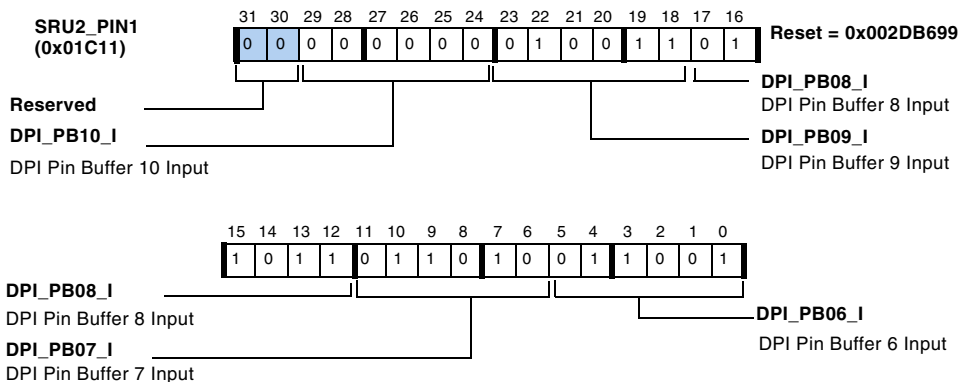


Figure 4-48. SRU2_PIN1 Register

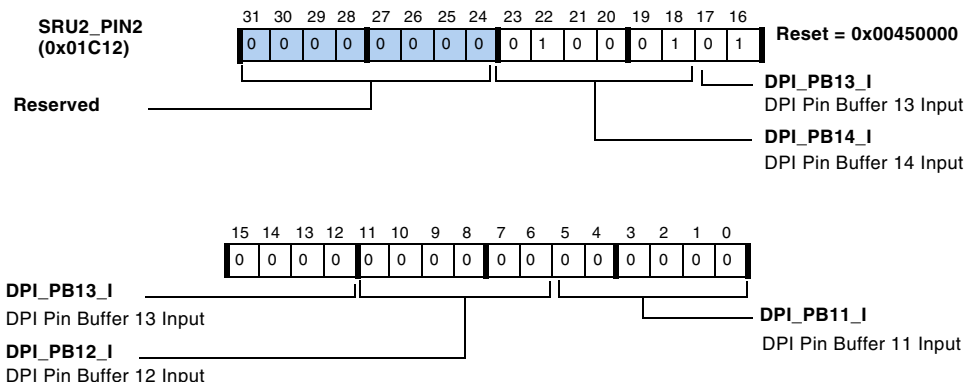


Figure 4-49. SRU2_PIN2 Register

Table 4-12. Group B Signals

Binary	Decimal	Signal	Description
000000	00	LOGIC_LEVEL_LOW	Logic level low (0)
000001	01	LOGIC_LEVEL_HIGH	Logic level high (1)
000010	02	DPI_P01_O	External pin 1

Making Connections in the SRUs

Table 4-12. Group B Signals (Cont'd)

Binary	Decimal	Signal	Description
000011	03	DPI_P02_O	External pin 2
000100	04	DPI_P03_O	External pin 3
000101	05	DPI_P04_O	External pin 4
000110	06	DPI_P05_O	External pin 5
000111	07	DPI_P06_O	External pin 6
001000	08	DPI_P07_O	External pin 7
001001	09	DPI_P08_O	External pin 8
001010	10	DPI_P09_O	External pin 9
001011	11	DPI_P10_O	External pin 10
001100	12	DPI_P11_O	External pin 11
001101	13	DPI_P12_O	External pin 12
001110	14	DPI_P13_O	External pin 13
001111	15	DPI_P14_O	External pin 14
010000	16	TIMER0_O	Timer0 output
010001	17	TIMER1_O	Timer1 output
010010	18	TIMER2_O	Timer2 output
010011	19	UART0_TX_O	UART0 transmitter output
010100	20	UART1_TX_O	UART1 transmitter output
010101	21	SPI_MISO_O	MISO from SPI
010110	22	SP_IM0SI_O	MOSI from SPI
010111	23	SPI_CLK_O	Clock output from SPI
011000	24	SPI_FLG0_O	Slave select 0 from SPI
011001	25	SPI_FLG1_O	Slave select 1 from SPI
011010	26	SPI_FLG2_O	Slave select 2 from SPI

Table 4-12. Group B Signals (Cont'd)

Binary	Decimal	Signal	Description
011011	27	SPI_FLG3_O	Slave select 3 from SPI
011100	28	SPIB_MISO_O	MISO from SPIB
011101	29	SPIB_M0SI_O	MOSI from SPIB
011110	30	SPIB_CLK_O	Clock output from SPIB
011111	31	SPIB_FLG0_O	Slave select 0 from SPIB
100000	32	SPIB_FLG1_O	Slave select 1 from SPIB
100001	33	SPIB_FLG2_O	Slave select 2 from SPIB
100010	34	SPIB_FLG3_O	Slave select 3 from SPIB
100011	35	FLAG4_O	Flag 4 output
100100	36	FLAG5_O	Flag 5 output
100101	37	FLAG6_O	Flag 6 output
100110	38	FLAG7_O	Flag 7 output
100111	39	FLAG8_O	Flag 8 output
101000	40	FLAG9_O	Flag 9 output
101001	41	FLAG10_O	Flag 10 output
101010	42	FLAG11_O	Flag 11 output
101011	43	FLAG12_O	Flag 12 output
101100	44	FLAG13_O	Flag 13 output
101101	45	FLAG14_O	Flag 14 output
101110	46	FLAG15_O	Flag 15 output
101111	47	PCG_CLKC_O	Precision clock generator clock C out
110000	48	PCG_CLKD_O	Precision clock generator clock D out
110001	49	PCG_FSC_O	Precision clock generator frame sync C out

Making Connections in the SRUs

Table 4-12. Group B Signals (Cont'd)

Binary	Decimal	Signal	Description
110010	50	PCG_FSD_O	Precision clock generator frame sync D out
110011– 111111	51-63	RESERVED	

Group C Connections—Pin Enable Signals

Group C signals, shown in [Table 4-13 on page 4-62](#), are used to specify whether each DPI pin is used as an output or an input by setting the source for the pin buffer enable. When a pin buffer enable (DPI_PBen_{xx}_I) is set (= 1), the signal present at the corresponding pin buffer input (DPI_PB_{xx}_I) is driven off chip as an output. When a pin buffer enable is cleared (= 0), the signal present at the corresponding pin buffer input is ignored.

The registers that control group C settings are shown in [Figure 4-50](#) through [Figure 4-52](#).



The TWI output is an open-drain output so the pins used for TWI data and clock should be connected to logic level 0.

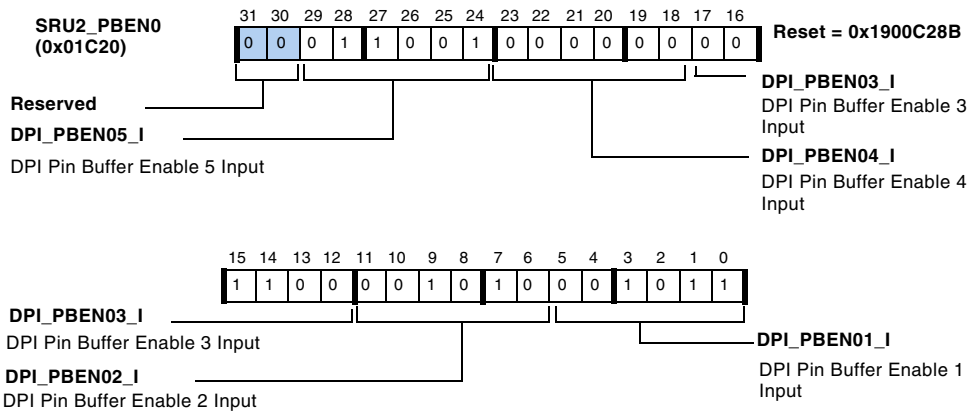


Figure 4-50. SRU2_PBEN0 Register

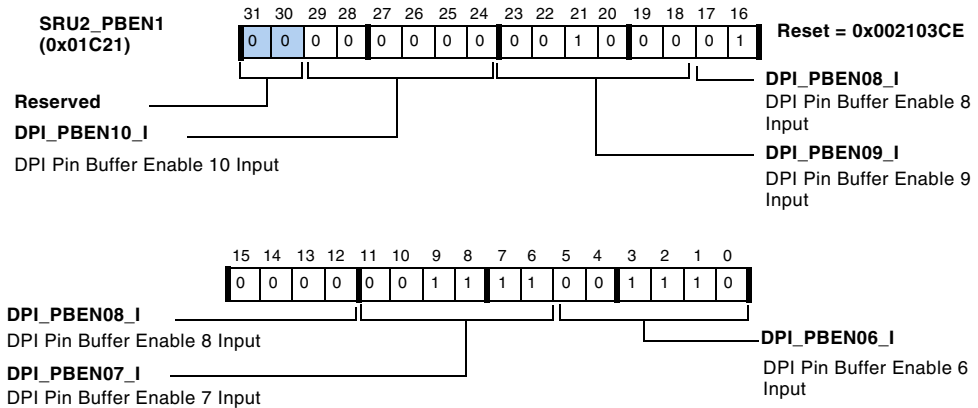


Figure 4-51. SRU2_PBEN1 Register

Making Connections in the SRUs

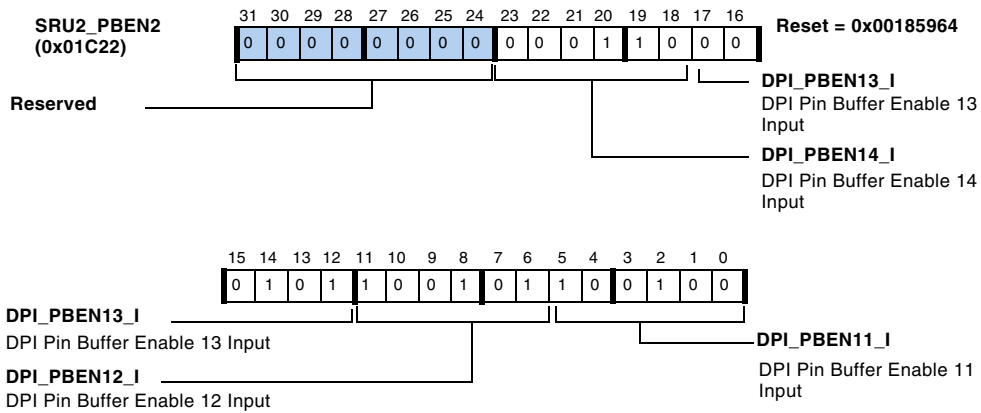


Figure 4-52. SRU2_PBEN2 Register

Table 4-13. Group C Signals

Binary	Decimal	Signal	Description
000000	00	LOGIC_LEVEL_LOW	Logic level low (0)
000001	01	LOGIC_LEVEL_HIGH	Logic level high (1)
000010	02	EXT_MISC_0	Miscellaneous control 0
000011	03	EXT_MISC_1	Miscellaneous control 1
000100	04	EXT_MISC_2	Miscellaneous control 2
000101	05	TIMER0_PE_O	Enable for timer 0 output
000110	06	TIMER1_PE_O	Enable for timer 1 output
000111	07	TIMER2_PE_O	Enable for timer 2 output
001000	08	UART0_TX_PE_0	Pin enable for UART 0 transmitter
001001	09	UART1_TX_PE_0	Pin enable for UART 1 transmitter
001010	10	SPIMISO_PE_O	Pin enable for MISO from SPI
001011	11	SPIMOSI_PE_O	Pin enable for MOSI from SPI
001100	12	SPICLK_PE_O	Pin enable for CLK from SPI

Table 4-13. Group C Signals (Cont'd)

Binary	Decimal	Signal	Description
001101	13	SPIFLG0_PE_O	Pin enable for slave select 0 from SPI
001110	14	SPIFLG1_PE_O	Pin enable for slave select 1 from SPI
001111	15	SPIFLG2_PE_O	Pin enable for slave select 2 from SPI
010000	16	SPIFLG3_PE_O	Pin enable for slave select 3 from SPI
010001	17	SPIBMISO_PE_O	Pin enable for MISO from SPIB
010010	18	SPIBMOSI_PE_O	Pin enable for MOSI from SPIB
010011	19	SPIBCLK_PE_O	Pin enable for CLK from SPIB
010100	20	SPIBFLG0_PE_O	Pin enable for slave select 0 from SPIB
010101	21	SPIBFLG1_PE_O	Pin enable for slave select 1 from SPIB
010110	22	SPIBFLG2_PE_O	Pin enable for slave select 2 from SPIB
010111	23	SPIBFLG3_PE_O	Pin enable for slave select 3 from SPIB
011000	24	FLAG4_PE_O	Pin enable for Flag 4 output
011001	25	FLAG5_PE_O	Pin enable for Flag 5 output
011010	26	FLAG6_PE_O	Pin enable for Flag 6 output
011011	27	FLAG7_PE_O	Pin enable for Flag 7 output
011100	28	FLAG8_PE_O	Pin enable for Flag 8 output
011101	29	FLAG9_PE_O	Pin enable for Flag 9 output
011110	30	FLAG10_PE_O	Pin enable for Flag 10 output
011111	31	FLAG11_PE_O	Pin enable for Flag 11 output
100000	32	FLAG12_PE_O	Pin enable for Flag 12 output
100001	33	FLAG13_PE_O	Pin enable for Flag 13 output
100010	34	FLAG14_PE_O	Pin enable for Flag 14 output
100011	35	FLAG15_PE_O	Pin enable for Flag 15 output
100100	36	TWI_SDATA_OE	Serial data output enable from TWI

General-Purpose I/O (GPIO) and Flags

Table 4-13. Group C Signals (Cont'd)

Binary	Decimal	Signal	Description
100101	37	TWI_SCLK_OE	Serial clock output enable from TWI
100110	38	EXT_MISC_3	Miscellaneous control 3
100111	39	EXT_MISC_4	Miscellaneous control 4
101000	40	EXT_MISC_5	Miscellaneous control 5
101001	41	EXT_MISC_6	Miscellaneous control 6
101010	42	EXT_MISC_7	Miscellaneous control 7
101011	43	EXT_MISC_8	Miscellaneous control 8
101100– 111111	44-63	Reserved	

The pin enable control registers activate the drive buffer for each of the 14 DPI pins. When the pins are not enabled (driven), they can be used as inputs.

General-Purpose I/O (GPIO) and Flags

Each interface is capable of using its pins for general-purpose I/O and flags. The sections that follow describe this use for each interface.

DAI GPIO and Flags

In the DAI, any of the pins may also be considered general-purpose input/output (GPIO) pins. Each of the DAI pins can also be set to drive a high or low logic level signal to assert signals. They can also be connected to miscellaneous signals and used as interrupt sources or as control inputs to other blocks.

DPI GPIO and Flags

In the DPI, the `EXT_MISC` signals are used for general-purpose I/O. An interrupt controller processes these signals to generate an interrupt, for example `DPI_INT`. The `EXT_MISC` signals can also be used to control pin enables. Flags 4-15 can be routed using the DPI pins.

Miscellaneous Signals

In the ADSP-21367/8/9 and ADSP-2137x processors, exception applications can also configure signals to:

- Act as interrupt sources
- Act as inverted signals (forcing a signal to active low)
- Connect one pin to another
- Act as pin enables

DAI/DPI Interrupt Controller

The DAI contains a dedicated interrupt controller that signals the core when DAI peripheral events occur.

Relationship to the Core

Generally, interrupts are classified as catastrophic or normal. Catastrophic events include any hardware interrupts (for example, resets) and emulation interrupts (under the control of the PC), math exceptions, and “reads” of memory that do not exist. Catastrophic events are treated as high priority events. In comparison, normal interrupts are “deterministic”—specific events emanating from a source (the causes), the result of which is the generation of an interrupt. The expiration of a timer can

DAI/DPI Interrupt Controller

generate an interrupt, a signal that a serial port has received data that must be processed, a signal that an SPI has either transmitted or received data, and other software interrupts like the insertion of a trap that causes a breakpoint—all are conditions, which identify to the core that an event has occurred.

Since DAI specific events generally occur infrequently, the DAI IC classifies such interrupts as either high or low priority interrupts. Within these broad categories, programs can indicate which interrupts are high and which are classified as low.

Any interrupt causes a four-cycle stall, since it forces the core to stop instruction execution while in process, then vector to the interrupt service routine (ISR), (which is basically an interrupt vector table (IVT) lookup), then proceed to implement the instruction referenced in the IVT. For more information, see [Appendix B, Interrupts](#).


When an interrupt from the DAI needs servicing, one of the two core interrupt service routines (ISR) must query the DAI's interrupt controller to determine the source(s). Sources can be any one or more of the interrupt controller's 32 configurable channels (`DAI_INT[31:0]`). [For more information, see “DAI Interrupt Controller Registers” on page A-112.](#)

DAI events trigger two interrupts in the primary IVT—one each for low or high priority.

DAI Interrupts

There are several registers in the DAI interrupt controller that can be configured to control how the DAI interrupts are reported to and serviced by the core's interrupt controller. Among other options, each DAI interrupt can be mapped either as a high or low priority interrupt in the primary interrupt controller. Certain DAI interrupts can be triggered on either the rising or the falling edge of the signals, and each DAI interrupt can also be independently masked.

Just as the core has its own interrupt latch registers (IRPTL and LIRPTL), the DAI has its own latch registers (DAI_IRPTL_L and DAI_IRPTL_H). When a DAI interrupt is configured to be high priority, it is latched in the DAI_IRPTL_H register. When any bit in the DAI_IRPTL_H register is set (= 1), bit 11 in the IRPTL register is also set and the core services that interrupt with high priority. When a DAI interrupt is configured to be low priority, it is latched in the DAI_IRPTL_L register. Similarly, when any bit in the DAI_IRPTL_L register is set (= 1), bit 6 in the LIRPTL register is also set and the core services that interrupt with low priority. Regardless of the priority, when a DAI interrupt is latched and promoted to the core interrupt latch, the ISR must query the DAI's interrupt controller to determine the source(s). Sources can be any one or more of the interrupt controller's 32 configurable channels (DAI_INT31-0). [For more information, see “DAI Interrupt Controller Registers” on page A-112.](#)


 Reading the DAI's interrupt latches clears them. Therefore, the ISR must service all the interrupt sources it discovers. That is, if multiple interrupts are latched in one of the DAI_IRPTL_x registers, all of them must be serviced before executing an RTI instruction.

The IDP_FIFO_GTN_INT interrupt is not cleared when the DAI_IRPTL_H/L registers are read. This interrupt is cleared automatically when the situation that caused the interrupt goes away.

DPI Interrupts

The DPI also has an interrupt controller, similar to that in the DAI. There are 14 interrupts—4 from the UARTs, 1 from the two wire interface (TWI) and 9 general-purpose I/O interrupts. All of these interrupts are combined into a single interrupt, namely DPI_INT. The DPI_IRPTL register contains the status on individual interrupts. Apart from DPI_IRPTL, there are two additional registers, DPI_IRPTL_RE and DPI_IRPTL_FE which are used for interrupt latching. Setting a corresponding bit in the DPI_IRPTL_RE register enables interrupt latching at the rising edge of the corresponding signal. Similarly, setting a corresponding bit in the

DPI_IRPTL_FE register enables interrupt latching at the falling edge of that signal. A particular interrupt can be latched at the rising or falling edge independently. Keeping corresponding bits reset in both these registers (DPI_IRPTL_RE and DPI_IRPTL_FE) disables the corresponding interrupt.

 The DPI_INT interrupt is automatically cleared when the DPI_IRPTL register is read for the EXT_MISC, UART_TX, and UART_RX (for DMA mode only) interrupts. Furthermore, the UART_RX interrupt for I/O mode must be cleared in the ISR. For information on using interrupts in the UART, see [“UARTxIER Register” on page 11-7](#), and [“UARTxIIR Register” on page 11-9](#). For information on using interrupts in the TWI, see [“Interrupt Source Register” on page 12-7](#) and [“Interrupt Enable Register” on page 12-8](#).

The UART and TWI interrupts are only used as rising edge interrupts. The corresponding bits in the DPI_IRPTL_FE register are reserved. By default, at reset, all interrupts are disabled (in other words, both the DPI_IRPTL_RE and DPI_IRPTL_FE registers have zero values).

Interrupt service routines (ISR) must read the DPI_IRPTL register to discover all of the interrupts that are currently latched. A shadow register, DPI_IRPTL_SH, is provided for the primary register DPI_IRPTL. Reads of this register returns data in the DPI_IRPTL register without clearing the contents of the register.

Note that the ISR must make sure that the current interrupt condition is removed before exiting to the main program. Further note that for the interrupts corresponding to the UART and the TWI, the bits in the DPI_IRPTL_RE register are used as enable bits and corresponding bits from the DPI_IRPTL_FE register are not used.

High and Low Priority Latches

In the ADSP-21367/8/9 and ADSP-2137x processors, a pair of registers (`DAI_IRPTL_H` and `DAI_IRPTL_L`) replace functions normally performed by the `IRPTL` register. A single register (`DAI_IRPTL_PRI`) specifies the latch to which each of these interrupts are mapped.

Two registers (`DAI_IRPTL_RE` and `DAI_IRPTL_FE`) replace the DAI peripheral's version of the `IMASK` register. As with the `IMASK` register, these DAI registers provide a way to specify which interrupts to notice and handle, and which interrupts to ignore. These dual registers function like `IMASK` does, but with a higher degree of granularity.

Signals from the SRU can be used to generate interrupts. For example, when `SRU_EXTMISCA2_INT` (bit 30) of `DAI_IRPTL_H` is set to one, any signals from the external miscellaneous channel 2 generate an interrupt. If set to one, DAI interrupts trigger an interrupt in the core and the interrupt latch is set. A read of this bit does not reset it to zero. The bit is only set to zero when the cause of the interrupt is cleared. A DAI interrupt indicates the source (in this case, external miscellaneous A, channel 2), and checks the IVT for an instruction (next operation) to perform.

The 32 interrupt signals within the interrupt controller are mapped to 2 interrupt signals in the primary interrupt controller of the SHARC processor core. The `DAI_IRPTL_PRI` register specifies if the interrupt controller interrupt is mapped to the high or low core interrupt (1 = high priority and 0 = low priority).

The `DAI_IRPTL_H` register is a read-only register with bits set for every DAI interrupt latched for the high priority core interrupts. The `DAI_IRPTL_L` register is a read-only register with bits set for every DAI interrupt latched for the low priority core interrupts. When a DAI interrupt occurs, the low or high priority core ISR interrogates its corresponding register to determine which of the 32 interrupt sources to service. When the `DAI_IRPTL_H`

register is read, the high priority latched interrupts are all cleared. When the `DAI_IRPTL_L` register is read, the low priority latched interrupts are all cleared.

Rising and Falling Edge Masks

For interrupt sources that correspond to waveforms (as opposed to DAI event signals such as DMA complete or buffer full), the edge of a waveform may be used as an interrupt source as well. Just as interrupts can be generated by a source, interrupts can also be generated and latched on the rising (or falling) edges of a signal. This concept does not exist in the main interrupt controller—only in the DAI interrupt controller.

When a signal comes in, the system needs to determine what kind of signal it is and what kind of protocol, as a result, to service. The preamble indicates the signal type. When the protocol changes, output (signal) type is noted.

For audio applications, the ADSP-21367/8/9 and ADSP-2137x processors need information about interrupt sources that correspond to waveforms (not event signals). As a result, the falling edge of the waveform may be used as an interrupt source as well. Programs may select any of these four conditions:

- Latch on the rising edge
- Latch on the falling edge
- Latch on *both* the rising and falling edge

The DAI interrupt controller may be configured using three registers. Each of the 32 interrupt lines can be independently configured to trigger in response to the incoming signal's rising edge, its falling edge, or both the rising edge and the falling edge. Setting a bit in either the

DAI_IRPTL_RE or DAI_IRPTL_FE registers enables the interrupt level on the rising and falling edges, respectively. For more information on these registers, see [“DAI Interrupt Controller Registers” on page A-112](#).

Programs can manage responses to signals by configuring registers. In a sample audio application, for example, upon detection of a change of protocol, the output can be muted. This change of output and the resulting behavior (causing the sound to be muted) results in an alert signal (an interrupt) being introduced in response (if the detection of a protocol change is a high priority interrupt).



The DAI_IRPTL_FE register can only be used for latching interrupts on the falling edge.

Use of the DAI_IRPTL_RE or DAI_IRPTL_FE registers allows programs to notice and respond to rising edges, falling edges, both rising and falling edges, or neither rising nor falling edges so they can be masked separately.

Responses to changes in conditions of signals (including changes in DMA state, introduction of error conditions, and so on) can only be enabled using the DAI_IRPTL_RE register.

Configuring Peripherals Using SRU1

The following sections describe how the various peripherals associated with SRU1 are configured.

Configuring the SPORTs

The serial port chapter provides the signal sensitivity information of the SPORT signals which needs to be considered while configuring the serial ports using DAI pins. [For more information, see “Serial Port Signal Sensitivity” on page 5-9.](#)

Configuring the PCGs

[“Precision Clock Generators” on page 13-1](#) provides extensive information on programming using the DAI. [“Clock and Frame Sync Divisors PCG Channel B” on page 13-20](#) is an example program that uses PCG channel B to output a clock on DAI pin 1 and a frame sync on DAI pin 2.

Configuring Peripherals Using SRU2

The following sections describe how the various peripherals associated with SRU2 are configured.

Configuring the SPI

All SPI signals are routed through the SRU2 and are routed as described in [“Signal Routing Units” on page 4-8](#), [Figure 4-3 on page 4-6](#) and [Figure 4-4 on page 4-7](#). Normally, it is acceptable to enable (SRU2) output buffers to permanently ground the pin enable signals. However, this does not work for the open-drain mode, because the SRU2 buffer always actively drives the output pin. Where open-drain mode is used, the pin enable signals must be connected to the pin enable associated with the SPI SRU2 buffers and connected with the `SPIB_MISO_0`, `SPIB_MOSI_0`, `SPIB_CLK_0`, and `SPIB_FLGx_0` pins. For more information on open-drain mode, see [“Open Drain Mode \(OPD\)” on page 6-9](#).

Choosing the Pin Enable for the SPI Clock

When in SPI master mode, the `SPICLK` signal is sent from the processor. The enable signal for the DPI pin being used for the clock must be connected correctly depending on the SPI mode being used (based on the setting of `CPHASE` and `CLKPL` bits in the `SPICTL` register). [Table 4-14](#) shows the correct pin enable to use by SPI mode.

Table 4-14. SPI Pin Enable Selections

Mode	CLKPL	CPHASE	Use Pin Enable...
0	0	0	HIGH
1	0	1	HIGH
2	1	0	SPI_CLK_O
3	1	1	SPI_CLK_O

As an example, the output of the clock to DPI pin 3 would be configured in SPI mode 3 as follows.

```
SRU(SPI_CLK_O,DPI_PB03_I);
SRU(SPI_CLK_O,DPI_PBEN03_I);
```

Configuring the Two Wire Interface

Each TWI device, whether master or slave, responds to the other devices with an acknowledge bit when data is received. For details, see [Chapter 12, Two Wire Interface Controller](#). For proper operation, the SRU must be set up so that data communication is possible in both directions on the data pin.

The examples in [Listing 4-1](#) through [Listing 4-4](#) show the SRU settings for different TWI modes.

Listing 4-1. TWI Master Transmit Mode

```
SRU(LOW,DPI_PB11_I)          /* Since TWI output is an open-drain
                               output, the TWI pin is connected
                               to logic level low */

SRU(TWI_DATA_PBEN_0,DPI_PBEN11_I) /* TWI data output connected
                                     to DPI pin 11 input */
```

Configuring Peripherals Using SRU2

```
SRU(DPI_PB11_0, TWI_DATA_I);    /* DPI pin 11 output connected
                                to TWI data input */

SRU(LOW,DPI_PB12_I)             /* Since TWI output is an open-drain
                                output, the TWI pin is connected
                                to logic level low */

SRU2(TWI_CLK_PBEN_0,DPI_PBEN12_I) /* TWI clock connected to DPI
                                pin 12 */
```

Listing 4-2. TWI Master Receive Mode

```
SRU(DPI_PB11_0,TWI_DATA_I)      /* DPI pin 11 output connected to
                                TWI data input */

SRU(LOW,DPI_PB11_I)             /* Since TWI output is an open-drain
                                output the TWI pin is connected to
                                logic level low */

SRU(TWI_DATA_PBEN_0,DPI_PBEN11_I) /* TWI data output
                                connected to DPI pin 11
                                input */

SRU(LOW,DPI_PB12_I) )          /* Since TWI output is an open-drain
                                output the TWI pin is connected to
                                logic level low */

SRU(TWI_CLK_PBEN_0,DPI_PBEN12_I) /* TWI Clock connected to DPI
                                pin 12
```

Listing 4-3. TWI Slave Transmit Mode

```
SRU(LOW,DPI_PB11_I)          /* Since TWI output is an open-drain
                               output the TWI pin is connected
                               to logic level low */

SRU(TWI_DATA_PBEN_0,DPI_PBEN11_I)  /* TWI data output connected
                                     to DPI pin 11 input */

SRU(DPI_PB11_0,TWI_DATA_I)        /* DPI pin 11 output connected to
                                     TWI data input */

SRU(DPI_PB12_0, TWI_CLK_I)        /* clock signal from DPI pin 12 is
                                     connected to the TWI slave
                                     clock input */

SRU(LOW,DPI_PBEN12_I)           /* Disables DPI pin 12 as input as
                                     the clock is not generated by
                                     the slave */
```

Listing 4-4. TWI Slave Receive Mode

```
SRU(DPI_PB11_0,TWI_DATA_I)        /* DPI pin 11 output connected to
                                     TWI data input */

SRU(LOW,DPI_PB11_I) )           /* Since TWI output is an open
                                     drain output the TWI pin is
                                     connected to logic level low */

SRU2(TWI_DATA_PBEN_0,DPI_PBEN11_I)  /* TWI data output
                                     connected to DPI pin
                                     11 input */
```

Using the SRU() Macro to Configure the DAI

```
SRU2(DPI_PB12_0, TWI_CLK_I)      /* Clock signal from DPI pin 12
                                   is connected to the TWI slave
                                   clock input */
SRU2(LOW,DPI_PBen12_I) )        /* Disables DPI pin 12 as input as
                                   the clock is not generated by
                                   the slave */
```

Using the SRU() Macro to Configure the DAI

As discussed in the previous sections, the signal routing unit (SRU) is controlled by writing values that correspond to signal sources into bit fields that further correspond to signal inputs. The SRU is arranged into functional groups such that registers that are made up of these bit fields accept a common set of source signal values.

In order to ease the coding process, the header file `sru.h` is included with the VisualDSP++ tools. This file implements a macro that automates most of the work of signal assignments and functions. The macro has identical syntax in C/C++ and assembly, and makes a single connection from an output to an input:

```
SRU(Output Signal, Input Signal);
```

The names passed to the macro are the names given in [Table 4-4](#) through [Table 4-9](#) and in the DAI registers section in “[DAI/SRU1 Connection Groups](#)” on [page 4-18](#). To use this macro, add the code shown in [Listing 4-5](#) to your source code.

Listing 4-5. DAI Macro Code

```
#include <sru.h>;
/* The following lines illustrate how the macro is used: */
/* Route SPORT 1 clock output to pin buffer 5 input */
    SRU(SPORT1_CLK_0,DAI_PB05_I);

/* Route pin buffer 14 out to IDP3 frame sync input */
    SRU(DAI_PB14_0,IDP3_FS_I);

/* Connect pin buffer enable 19 to logic low */
    SRU(LOW,DAI_PBEN19_I);
```

Additional example code is available on the Analog Devices Web site.



There is a macro that has been created to connect peripherals used in a DAI configuration. This code can be used in both assembly and C code. See the `INCLUDE` file `SRU.H`.

There is also a software plug-in called the Expert DAI that greatly simplifies the task of connecting the signals described in this chapter. This plug-in is described in Engineer-to-Engineer Note EE-243, “Using the Expert DAI for ADSP-2126x and ADSP-2136x SHARC Processors”. This EE note is also found on the Analog Devices Web site.

Using the SRU() Macro to Configure the DAI

5 SERIAL PORTS


The ADSP-21367/8/9 and ADSP-2137x processors have up to eight independent, synchronous serial ports (SPORTs) that provide an I/O interface to a wide variety of peripheral devices. They are called SPORT0, SPORT1, SPORT2, SPORT3, SPORT4, SPORT 5, SPORT6, and SPORT7. Each SPORT has its own set of control registers and data buffers. With a range of clock and frame synchronization options, the SPORTs allow a variety of serial communication protocols and provide a glueless hardware interface to many industry-standard data converters and CODECs.

Serial ports can operate at one-eighth the full clock rate of the processor, a maximum rate of 50M bit/s for $(CCLK) = 3$ ns. If channels A and B are active, each SPORT has 83.3M bit/s maximum total throughput. Bidirectional (transmit or receive) functions provide greater flexibility for serial communications. Serial port data can be automatically transferred to and from on-chip memory using DMA block transfers. In addition to standard synchronous serial mode, each SPORT offers a time division multiplexed (TDM) multichannel mode, left-justified sample pair mode, and I²S mode.

Features


Serial ports offer the following additional features and capabilities:

- Two additional SPORTs, each with their own DMA channels and interrupts, have been added to the ADSP-21367, ADSP-21368, ADSP-21369, and ADSP-21371 SHARC processors.
- Two bidirectional channels (A and B) per SPORT, configurable as either transmitters or receivers. Each SPORT can also be configured as two receivers or two transmitters, permitting two unidirectional streams into or out of the same SPORT. This bidirectional functionality provides greater flexibility for serial communications. Further, two SPORTs can be combined to enable full-duplex, dual-stream communications.
- All serial data signals have programmable receive and transmit functions and thus have one transmit and one receive data buffer register (double buffer). There is also a bidirectional shift register associated with each serial data signal. Double buffering provides additional time to service the SPORT.
- A serial clock and frame sync provide signals in a wide range of frequencies. Alternately, the SPORT can accept clock and frame sync input from an external source, as described in [Figure 5-10 on page 5-70](#).
- The processors allow interrupt-driven, single word transfers to and from on-chip memory controlled by the core, described in [“Single Word Transfers” on page 5-81](#).
- DMA transfers to and from on-chip memory. Each SPORT can automatically receive or transmit an entire block of data. Further the SPORTs on the processors offer chained DMA operations for multiple data blocks, see [“Chaining DMA Processes” on page 2-14](#).

-  New DMA channels are added for SPORT6 and 7. These new SPORTs also have their own interrupt lines.

Operation Modes

Three serial ports have four operation modes: standard DSP serial, left-justified sample pair, I²S, and multichannel. In standard DSP serial, left-justified sample pair and I²S modes, when both A and B channels are used, they transmit or receive data simultaneously, sending or receiving bit 0 on the same edge of the serial clock, bit 1 on the next edge of the serial clock, and so on. In multichannel mode, SPORTs can receive and transmit A and B channel data selectively from up to 128 channels of a TDM serial bit stream. See [“SPORT Operation Modes” on page 5-10](#).

-  For the ADSP-21367/8/9 and ADSP-2137x processors in multichannel mode, there are several enhancements from previous SHARC processors.
- In previous SHARC processors, SPORTs were forced into pairs when in multichannel mode. The SPORTs in the ADSP-21367/8/9 and ADSP-2137x processors now operate independently.
 - In addition, control registers are added to each SPORT. In previous versions, there was only one TDM control register for each SPORT pair.
 - New data valid pins are added to each SPORT. In previous versions, in paired mode, the frame sync of one of the SPORTs was used as a data valid pin so eight new pin definitions are added to the SRU.

Operation Modes

- In multichannel mode, support for packed I²S mode is new for the ADSP-21367/8/9 and ADSP-2137x processors. However, it should be noted that packed I²S mode is not identical to I²S mode in all cases. [For more information, see “Packed I2S Mode” on page 5-33.](#)
- Even though SPORTs can operated independently in TDM mode, compression and expansion logic is only available in SPORT0, 2, 4, and 6 and expansion logic is available only in SPORT1, 3, 5 and 7.
- The Frame sync error detection logic is added for the ADSP-21367/8/9 and ADSP-2137x processors. It detects frame syncs coming early where the frame sync arrives while transmission/reception of previous word is occurring. It does not detect errors such as frame syncs arriving late. One dedicated error interrupt is added which is shared by all sports.

The SPORTs are configurable for transferring data words between 3 and 32 bits in length, either most significant bit (MSB) first or least significant bit (LSB) first. Words must be between 8 and 32 bits in length for I²S and left-justified sample pair mode. Refer to [“Data Word Formats” on page 5-43](#) and the individual SPORTs operation mode sections for additional information.

Multichannel mode operation supports 128-channel TDM, described in [“Multichannel Operation” on page 5-25.](#)



Receive comparison and two-dimensional DMA are not supported in the ADSP-21367/8/9 and ADSP-2137x.

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

Serial Port Signals

Figure 5-1 shows the SPORT signals (note that not all models have eight SPORTs). Any 20 of these 32 signals can be mapped to digital audio interface (DAI_Px) pins through the signal routing unit (SRU1). For more information, see Chapter 4, Digital Audio/Digital Peripheral Interfaces.

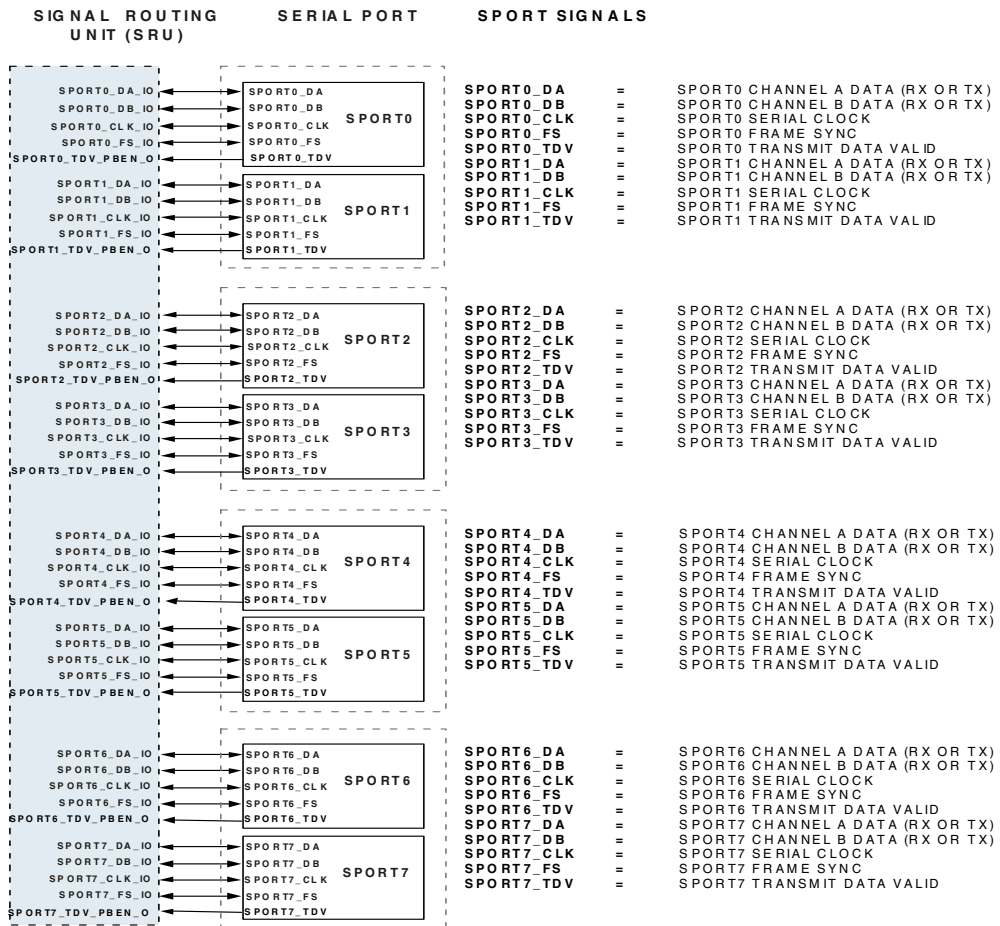


Figure 5-1. DSP Standard Serial Mode – Serial Port Signals



Pairings of SPORTs (0 and 1, 2 and 3, and 4 and 5, 6 and 7) are used only in loopback mode for testing.

A SPORT receives serial data on one of its bidirectional serial data signals configured as inputs, or transmits serial data on the bidirectional serial data signals configured as outputs. It can receive or transmit on both channels simultaneously and unidirectionally, where the pair of data signals can both be configured as either transmitters or receivers.

The `SPORTx_DA` and `SPORTx_DB` channel data signals on each SPORT cannot transmit and receive data simultaneously for full-duplex operation. Two SPORTs must be combined to achieve full-duplex operation. The `SPTRAN` bit in the `SPCTLx` register controls the direction for both the A and B channel signals. Therefore, the direction of channel A and channel B on a particular SPORT must be the same.

Serial communications are synchronized to a clock signal. Every data bit must be accompanied by a clock pulse. Each SPORT can generate or receive its own clock signal (`SPORTx_CLK`). Internally-generated serial clock frequencies are configured in the `DIVx` registers. The A and B channel data signals shift data based on the rate of `SPORTx_CLK`. See [Figure 5-10 on page 5-70](#) for more details.

In addition to the serial clock signal, data may be signaled by a frame synchronization signal. The framing signal can occur at the beginning of an individual word or at the beginning of a block of words. The configuration of frame sync signals depends upon the type of serial device connected to the processor. Each SPORT can generate or receive its own frame sync signal (`SPORTx_FS`) for transmitting or receiving data. Internally-generated frame sync frequencies are configured in the `DIVx` registers. Both the A and B channel data signals shift data based on their corresponding `SPORTx_FS` signal. See [Figure 5-10 on page 5-70](#) for more details.

Figure 5-2 shows a block diagram of a SPORT. Setting the `SPTRAN` bit enables the data buffer path, which, once activated, responds by shifting data in at the rate of `SPORTx_CLK`. An application program must use the correct SPORT data buffers, according to the value of `SPTRAN` bit. The `SPTRAN` bit enables either the transmit data buffers for the transmission of A and B channel data, or it enables the receive data buffers for the reception of A and B channel data. Inactive data buffers are not needed.

If the SPORT is configured as a transmitter, the data transmitted is written to the `TXSPxA/TXSPxB` buffer. The data is (optionally) companded in hardware on the primary A channel (SPORT0, 2, 4, and 6 only), then automatically transferred to the transmit shift register, because companding is not supported on the secondary B channels. The data in the shift register is then shifted out via the SPORT's `SPORTx_DA` or `SPORTx_DB` signal, synchronous to the `SPORTx_CLK` clock. If framing signals are used, the `SPORTx_FS` signal indicates the start of the serial word transmission. The `SPORTx_DA` or `SPORTx_DB` signal is always driven if the SPORT is enabled (`SPEN_A` or `SPEN_B` = 1 in the `SPCTLx` control register), unless it is in multi-channel mode and an inactive time slot occurs.

When the SPORT is configured as a transmitter (`SPTRAN` = 1), the `TXSPxA` and `TXSPxB` buffers, and the channel transmit shift registers respond to `SPORTx_CLK` and `SPORTx_FS` to transmit data. The receive `RXSPxA` and `RXSPxB` buffers, and the receive shift registers are inactive and do not respond to `SPORTx_CLK` and `SPORTx_FS` signals. Since these registers are inactive, using the core to read them causes a core hang. Furthermore, a DMA operation may read these registers erroneously without causing a core hang.



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

Serial Port Signals

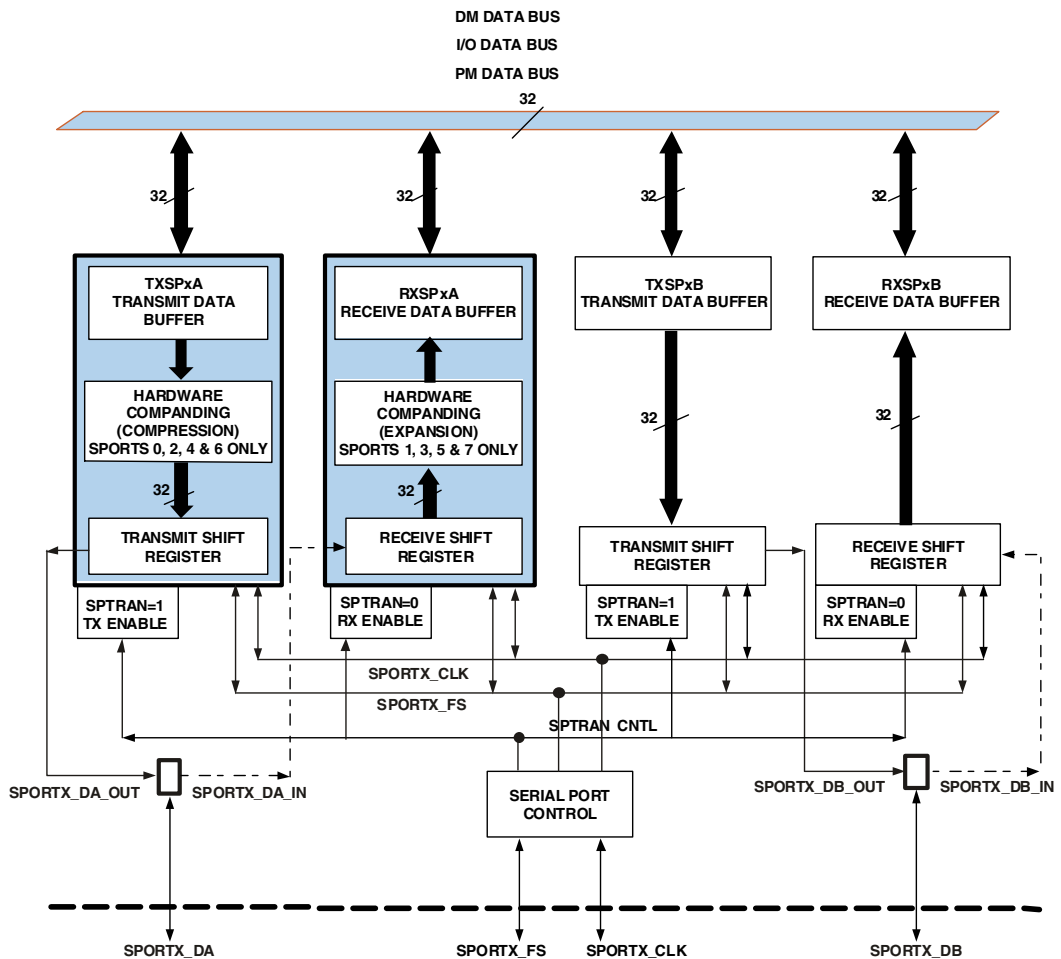




Figure 5-2. Serial Port Block Diagram

If the SPORT is configured as a serial receiver ($SPTRAN = 0$), the receive portion of the SPORT shifts in data from the $SPORTx_DA$ or $SPORTx_DB$ signal, synchronous to the $SPORTx_CLK$ receive clock. If framing signals are used, the $SPORTx_FS$ signal indicates the beginning of the serial word being received. When an entire word is shifted in on the primary

A channel, the data is (optionally) expanded (SPORT1, 3, and 5 only), then automatically transferred to the `RXSPxA` buffer. When an entire word is shifted in on the secondary channel, it is automatically transferred to the `RXSPxB` buffer.

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

 If the SPORTs are configured as receivers (`SPTRAN` bit = 0 in `SPCTLx`), programs should not write to the inactive `TXSPxA` and `TXSPxB` buffers. If the core keeps writing to the inactive buffer, the transmit buffer status becomes full. This causes the core to hang indefinitely since data is never transmitted out of the deactivated transmit data buffers.

 The processor's SPORTs are not UARTs and cannot communicate with an RS-232 device or any other asynchronous communications protocol. Instead, the ADSP-21367/8/9 and ADSP-2137x processors have two dedicated UART peripherals. [For more information, see Chapter 11, UART Port Controller.](#)

Serial Port Signal Sensitivity

There is some sensitivity to noise on the `SPORTx_CLK` and `SPORTx_FS` signals. In certain cases, by correctly programming the signal routing unit (SRU1) clock and frame sync registers, the reflection sensitivity in these signals can be avoided.

Consider this SPORT clock example. In its default routing, SRU1 maps the signal from the DAI pin (`DAI_PBxx_0`) back to the sport clock input (`SPORTx_CLK_I`), as well as routing the SPORT clock output

SPORT Operation Modes

(SPORTx_CLK_0) to the pin buffer input (PBxx_I). The connection of PBxx_0 to SPORTx_CLK_I opens a vulnerability to a glitch coming in even though the SPORT is driving the clock as an output. By programming SRU1 to remove this input path, programs can avoid this vulnerability.

This is done by leaving the routing of SPORTx_CLK_0 to PBxx_I as before, providing the SPORT clock off chip, but also routing it directly back to SPORTx_CLK_I to give the state machine its signal. This effectively closes off the external access to SPORTx_CLK_I. In the SRU programming code, the following code should be added: `SRU(SPORTx_CLK_0,SPORTx_CLK_I)`. [For more information, see Chapter 4, Digital Audio/Digital Peripheral Interfaces.](#)

SPORT Operation Modes

SPORTs operate in four modes:

- Standard DSP serial mode, described in [“Standard DSP Serial Mode” on page 5-12](#)
- Left-justified sample pair mode, described in [“Left-Justified Sample Pair Mode” on page 5-16](#)
- I²S mode, described in [“I2S Mode” on page 5-20](#)
- Multichannel mode, described in [“Multichannel Operation” on page 5-25](#)




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

The SPORT operating mode can be selected via the SPCTLx register. See [Table 5-1](#) for a summary of the control bits as they relate to the four operating modes.

The operating mode (OPMODE) bit of the SPCTLx register selects between I²S mode/left-justified sample pair mode, and non-I²S mode (DSP serial port/multichannel mode). In multichannel mode, the MCEA bit in the SPMCTLx register enables the A channels and the MCEB bit in the SPMCTLx register enables the B channels. The data direction bit (SPTRAN) selects whether the port is a transmitter or receiver in all the SPORT operation modes.

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

 Companding logic is only supported for multi-channel mode and packed I²S mode. Companding logic is available only for SPORT0, 2, 4 and 6 A channels. Expansion logic is now available only in SPORT1, 3, 5 and 7 A channels.

The SPCTLx register is unique in that the name and functionality of its bits changes depending on the operation mode selected. In each section that follows, the bit names associated with the operating modes are described. [Table 5-1](#) provides values for each of the bits in the SPORT control (SPCTLx) registers that must be set in order to configure each specific SPORT operation mode. An X in a field indicates that the bit is not supported for the specified operating mode.

Table 5-1. SPORT Operation Modes

Operating Modes	Bits					
	OPMODE	LAFS	FRFS	MCEA	MCEB	SLENx
Standard DSP Serial Mode	0	0, 1	X	0	0	3-32 ¹
I ² S (Tx/Rx on Left Channel First)	1	0	1	0	0	8-32
I ² S (Tx/Rx on Right Channel First)	1	0	0	0	0	8-32

SPORT Operation Modes

Table 5-1. SPORT Operation Modes (Cont'd)

Operating Modes	Bits					
	OPMODE	LAFS	FRFS	MCEA	MCEB	SLEN _x
Packed I ² S Mode A Channel	1	0	X	1	0	3-32
Packed I ² S Mode B Channel	1	0	X	0	1	3-32
Packed I ² S Mode A and B Channels	1	0	X	1	1	3-32
Left-justified Sample Pair Mode (Tx/Rx on FS Rising Edge)	1	1	0	0	0	8-32
Left-justified sample pair (Tx/Rx on FS Falling Edge)	1	1	1	0	0	8-32
Multichannel A Channels	0	0	X	1	0	3-32 ¹
Multichannel B Channels	0	0	X	0	1	3-32 ¹
Multichannel A and B Channels	0	0	X	1	1	3-32 ¹

- 1 Although SPORTs process word lengths of 3 to 32 bits, transmitting or receiving words smaller than 7 bits at core clock frequency/4 of the processor may cause incorrect operation when DMA chaining is enabled. Chaining disables the processor's internal I/O bus for several cycles while the new transfer control block (TCB) parameters are being loaded. Receive data may be lost (for example, overwritten) during this period.

Standard DSP Serial Mode

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

Standard DSP Serial Mode Control Bits

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

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

Clocking Options

In standard DSP serial mode, the SPORTs can either accept an external serial clock or generate it internally. The `ICLK` bit in the `SPCTL` register determines the selection of these options (see [“Clock Signal Options” on page 5-36](#) for more details). For internally-generated serial clocks, the `CLKDIV` bits in the `DIVx` register configure the serial clock rate (see [Figure 5-10 on page 5-70](#) for more details).

Finally, programs can select whether the serial clock edge is used for sampling or driving serial data and/or frame syncs. This selection is performed using the `CKRE` bit in the `SPCTL` register (see [Table A-8 on page A-37](#) for more details).

Frame Sync Options

A variety of framing options are available for the SPORTs. For detailed descriptions of framing options, see [“Frame Sync Options” on page 5-37](#). In this mode, these options are independent of clocking, data formatting, or other configurations. The frame sync signal (`SPORTx_FS`) is used as a framing signal for serial word transfers.

Framing is optional for serial communications. The `FSR` bit in the `SPCTL` register controls whether the frame sync signal is required for every serial word transfer or if it is used simply to start a block of serial word transfers. See [“Framed Versus Unframed Frame Syncs” on page 5-37](#) for more

SPORT Operation Modes

details on this option. Similar to the serial clock, the frame sync can be an external signal or generated internally. The `IFS` bit in the `SPCTL` register allows the selection between these options (see the internal frame sync select bit description in [Figure 5-10 on page 5-70](#) for more details). For internally-generated frame syncs, the `FSDIV` bits in the `DIVx` register configure the frame sync rate. For internally-generated frame syncs, it is also possible to configure whether the frame sync signal is activated based on the `FSDIV` setting and the transmit or receive buffer status, or by the `FSDIV` setting only.

All settings are configured through the `DIFS` bit of the `SPCTL` register. See [“Data-Independent Frame Syncs” on page 5-41](#) for more details. The frame sync can be configured to be active high or active low through the `LFS` bit in the `SPCTL` register. See [“Active Low Versus Active High Frame Syncs” on page 5-39](#) for more details). The timing between the frame sync signal and the first bit of data that are either transmitted or received is also selected through the `LAFS` bit in the `SPCTL` register. See [“Early Versus Late Frame Syncs” on page 5-40](#) for more details.

Data Formatting

Several data formatting options are available for the SPORTs in DSP standard serial mode. Each SPORT has an A and B channel available. Both can be configured for transmitting or receiving. The `SPTRAN` bit controls the configuration of transmit versus receive operations. Serial ports can transmit or receive a selectable word length, which is programmed by the `SLEN` bits in the `SPCTL` register (see [“Setting Word Length \(SLEN\)” on page 5-17](#) for more details).

Serial ports also include companding hardware built into the A channels that allows sign extension or zero-filling of upper bits of the serial data word. These configurations are selected by the `DTYPE` bits in the `SPCTL` register. See [“Data Type” on page 5-46](#) and [“Companding” on page 5-47](#) for more information.

The endian format (LSB versus MSB first) is selectable by the `LSBF` bit of the `SPCTL` register (see [“Endian Format” on page 5-45](#) for more details). Data packing of two serial words into a 32-bit word is also selectable. The `PACK` bit in the `SPCTL` register controls this option. See [“Data Packing and Unpacking” on page 5-45](#) for more details.

Data Transfers

Serial port data is transferred using two different methods:

- DMA transfers
- Core-driven single word transfers

DMA transfers can be set up to transfer a configurable number of serial words between the SPORT buffers (`TXSPxA`, `TXSPxB`, `RXSPxA`, and `RXSPxB`) and internal memory automatically. For more information on Sport DMA operations, see DMA Block transfers section on [“DMA Block Transfers” on page 5-73](#). Core-driven transfers use SPORT interrupts to signal the processor core to perform single word transfers to/from the SPORT buffers (`TXSPxA`, `TXSPxB`, `RXSPxA`, and `RXSPxB`). See [“SPORT Interrupts” on page 5-72](#) for more details.

Status Information

Serial ports provide status information about data buffers through the `DXS_A` and `DXS_B` status bits and error status through the `ROVF` or `TUVF` bits in the `SPCTLx` register. See [“Serial Port Control Registers \(SPCTLx\)” on page 5-59](#) for more details.

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


Left-Justified Sample Pair Mode

In left-justified sample pair mode, each frame sync cycle receives or transmits two samples of data—one sample on the high segment of the frame sync, the other on the low segment of the frame sync. Prior to development of the I²S standard, many manufacturers used a variety of non-standard stereo modes. Some companies continue to use this mode, which is supported by many of today's audio front-end devices.

Programs have control over various attributes of this mode. One attribute is the number of bits (8- to 32-bit word lengths). However each sample of the pair that occurs on each frame sync must be the same length. Set the late frame sync bit (LAFS bit) = 1 for left-justified sample pair mode. (See [Table 5-1 on page 5-11](#).) Then, choose the frame sync edge associated with the first word in the frame sync cycle, using the FRFS bit (1 = frame on falling edge of frame sync, 0 = frame on rising edge of frame sync).

Refer to [Table 5-1 on page 5-11](#) for additional information about specifying left-justified sample pair mode.

In left-justified sample pair mode, if both transmit channels (TXSPxA and TXSPxB) on a SPORT are enabled, then the SPORT transmits these channels simultaneously. In other words, each channel transmits a sample pair. Data is transmitted in MSB-first format. If both receive channels (RXSPxA and RXSPxB) on a SPORT are enabled, the SPORT receives simultaneously.

 Multichannel operation and companding are not supported in left-justified sample pair mode.

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

Setting the Internal Serial Clock and Frame Sync Rates

The serial clock rate for internal clocks can be set using the `CLKDIV` bit field in the `DIVx` register and the frame sync rate for internal frame sync can be set using the `FSDIV` bit field in the `DIVx` register. For details, see [Figure 5-10 on page 5-70](#).

Left-Justified Sample Pair Mode Control Bits

Several bits in the `SPCTLx` control register enable and configure left-justified sample pair mode operation:

- Operation mode (`OPMODE`)
- Channel enable (`SPEN_A` and `SPEN_B`)
- Word length (`SLEN`)
- Left channel first (`FRFS`)
- Master mode enable (`MSTR`)
- Late frame sync (`LAFS`)

For complete descriptions of these bits, see [“SPORT Serial Control Registers \(SPCTLx\)” on page A-29](#).

Setting Word Length (SLEN)

SPORTs handle data words containing 8 to 32 bits in left-justified sample pair mode. Programs need to set the bit length for transmitting and receiving data words. For details, see [“Word Length” on page 5-43](#).

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

SPORT Operation Modes

To transmit or receive words continuously in left-justified sample pair mode, load the `DIV` register with the `FSDIV` value the same as `SLEN`. For example, for 8-bit data words where `SLEN` = 7, set `FSDIV` = 7.

Enabling SPORT Master Mode (MSTR)

The SPORT's transmit and receive channels can be configured for master or slave mode. In master mode (`MSTR` = 1), the processor generates the word select and serial clock signals for the transmitter or receiver. In slave mode (`MSTR` = 0), an external source generates the word select and serial clock signals for the transmitter or receiver. [For more information, see “Setting the Internal Serial Clock and Frame Sync Rates” on page 5-17.](#)

Selecting Transmit and Receive Channel Order (FRFS)

Using the `FRFS` bit, it is possible to select which frame sync edge (rising or falling) on which the serial ports transmit or receive the first sample. See [Table 5-1 on page 5-11](#) for more details.

Selecting Frame Sync Options (DIFS)

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

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

Enabling SPORT DMA (SDEN)

DMA can be enabled or disabled independently on any SPORT transmit and receive channels. [For more information, see “Moving Data Between SPORTs and Internal Memory” on page 5-73.](#) Set `SDEN_A` or `SDEN_B` (=1) to enable DMA and set the channel in DMA-driven data transfer mode. Clear `SDEN_A` or `SDEN_B` (=0) to disable DMA and set the channel in an interrupt-driven data transfer mode.

Interrupt-Driven Data Transfer Mode

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

The SPORT generates an interrupt in every peripheral clock cycle when the transmit buffer has a vacancy or the receive buffer has data. To determine the source of an interrupt, applications must check the transmit or receive data buffer status bits. For details, see [“Single Word Transfers” on page 5-81.](#)

DMA-Driven Data Transfer Mode

Each transmitter and receiver has its own DMA registers. For details, see [“Selecting Transmit and Receive Channel Order \(FRFS\)” on page 5-18](#) and [“Moving Data Between SPORTs and Internal Memory” on page 5-73.](#) The same DMA channel drives both channels in the pair for the transmitter or receiver. The software application must stop multiplexing the left and right channel data received by the receive buffer, as the left and right data is interleaved in the DMA buffers.

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

[Figure 5-3](#) shows the relationship between frame sync (word select), serial clock, and left-justified sample pair mode data. Timing for word select is the same as for frame sync.

SPORT Operation Modes

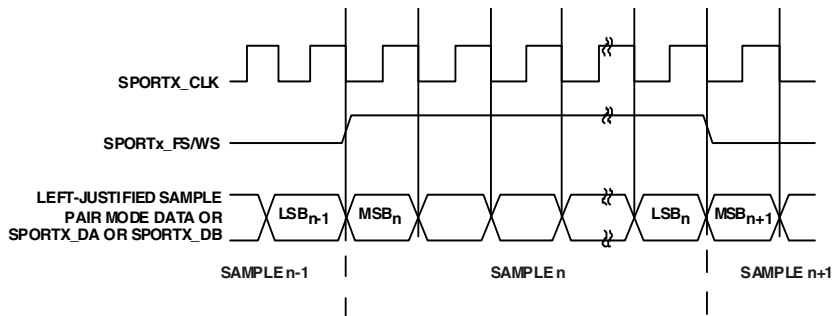


Figure 5-3. Word Select Timing in Left-Justified Sample Pair Mode¹


- ¹ This figure illustrates only one possible combination of settings attainable in the left-justified sample pair mode. In this example case, OPMODE = 1, LAFS = 1, and FRFS = 1. For additional combinations, refer to [Table 5-1 on page 5-11](#).

I²S Mode

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

- Audio D/A and A/D converters
- PC multimedia audio controllers
- Digital audio transmitters and receivers that support serial digital audio transmission standards, such as AES/EBU, SP/DIF, IEC958, CP-340, and CP-1201
- Digital audio signal processors
- Dedicated digital filter chips
- Sample rate converters

The I²S bus transmits audio data and control signals over separate lines. The data line carries two multiplexed data channels—the left channel and the right channel. In I²S mode, if both channels on a SPORT are set up to transmit, then SPORT transmit channels (TXSPxA and TXSPxB) transmit simultaneously, each transmitting left and right I²S channels. If both channels on a SPORT are set up to receive, the SPORT receive channels (RXSPxA and RXSPxB) receive simultaneously, each receiving left and right I²S channels. Data is transmitted in MSB-first format.

 Companding is not supported in I²S mode.


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

Setting the Internal Serial Clock and Frame Sync Rates

The serial clock rate for internal clocks can be set using the CLKDIV bit field in the DIVx register and the frame sync rate for internal frame sync can be set using the FSDIV bit field in the DIVx register. For details, see [Figure 5-10 on page 5-70](#).

I²S Mode Control Bits

[Table 5-1 on page 5-5](#) shows that I²S mode is simply a subset of the left-justified sample pair mode which can be invoked by setting OPMODE = 1, LAFS = 0, and FRFS = 1.

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

SPORT Operation Modes

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

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

Setting the `OPMODE` bit = 1 selects the I²S mode of operation.

Setting Word Length (`SLEN`)

SPORTs handle data words containing 8 to 32 bits in standard I²S mode. Programs need to set the bit length for transmitting and receiving data words. For details, see [“Word Length” on page 5-43](#).



More than 32-bit words can be handled using the packed I2S mode. [For more information, see “Packed I2S Mode” on page 5-33.](#)

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

In I²S mode, load the `FSDIV` value in the `DIVx` register with the same value used for `SLEN` to transmit or receive words continuously. For example, for 8-bit data words where `SLEN` = 7, set `FSDIV` = 7.

Enabling SPORT Master Mode (MSTR)

The SPORTs transmit and receive channels can be configured for master or slave mode. In master mode, the processor generates the word select and serial clock signals for the transmitter or receiver. In slave mode, an external source generates the word select and serial clock signals for the transmitter or receiver. When `MSTR` is cleared (`=0`), the processor uses an external word select and clock source and the SPORT transmitter or receiver is a slave. When `MSTR` is set (`=1`), the processor uses the processor's internal clock for word select and clock source and the SPORT transmitter or receiver is the master. For more information, see [“Setting the Internal Serial Clock and Frame Sync Rates”](#) on page 5-17.

Selecting Transmit and Receive Channel Order (FRFS)

In master and slave modes, programs can configure which I²S channel that each SPORT channel transmits or receives first. By default, the SPORT channels transmit and receive on the right I²S channel first. The left and right I²S channels are time-duplexed data channels.

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

Selecting Frame Sync Options (DIFS)

The `DIFS` bit selects whether the serial port uses the data independent frame sync or the data dependant frame sync. Normally, the internally generated frame sync signal is output only when the transmit buffer has data ready to transmit for the transmitter and when the receive buffer is not full for the receiver. The data-independent frame sync (DIFS) mode allows the continuous generation of the `FSx` signal, with or without new data in the transmit register or the receive buffer.

SPORT Operation Modes

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

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

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

When using both SPORT channels as receivers and MSTR = 1, SPTRAN = 1 and DIFS = 1, the processor generates a frame sync signal at the frequency set by FSDIV, irrespective of the receive buffer status. Bits 31–16 of the DIV register comprise the FSDIV bit field. [For more information, see “SPORT Divisor Registers \(DIVx\)” on page A-44.](#)

Enabling SPORT DMA (SDEN)

DMA can be enabled or disabled independently on any SPORT's transmit and receive channels. [For more information, see “Moving Data Between SPORTs and Internal Memory” on page 5-73.](#) Set SDEN_A or SDEN_B (=1) to enable DMA and set the channel in DMA-driven data transfer mode. Clear SDEN_A or SDEN_B (=0) to disable DMA and set the channel in an interrupt-driven data transfer mode.

Interrupt-Driven Data Transfer Mode

Both the A and B channels share a common interrupt vector in the interrupt-driven data transfer mode, regardless of whether they are configured as a transmitter or receiver.

The SPORT generates an interrupt when the transmit buffer has a vacancy or the receive buffer has data. To determine the source of an interrupt, applications must check the transmit or receive data buffer status bits. [For more information, see “Single Word Transfers” on page 5-81.](#)

DMA-Driven Data Transfer Mode

Each transmitter and receiver has its own DMA registers. For details, see [“Selecting Transmit and Receive Channel Order \(FRFS\)” on page 5-18](#) and [“Moving Data Between SPORTs and Internal Memory” on page 5-73](#). The same DMA channel drives the left and right I²S channels for the transmitter or the receiver. The software application must stop multiplexing the left and right channel data received by the receive buffer, because the left and right data is interleaved in the DMA buffers.

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

[Figure 5-4](#) shows the relationship between frame sync (word select), serial clock, and I²S data. Timing for word select is the same as for frame sync.



The `SPL` bit applies to DSP standard serial and I²S modes only.

Multichannel Operation

The processor's SPORTs offer a multichannel mode of operation, which allows the SPORT to communicate in a time division multiplexed (TDM) serial system. In multichannel communications, each data word of the

SPORT Operation Modes

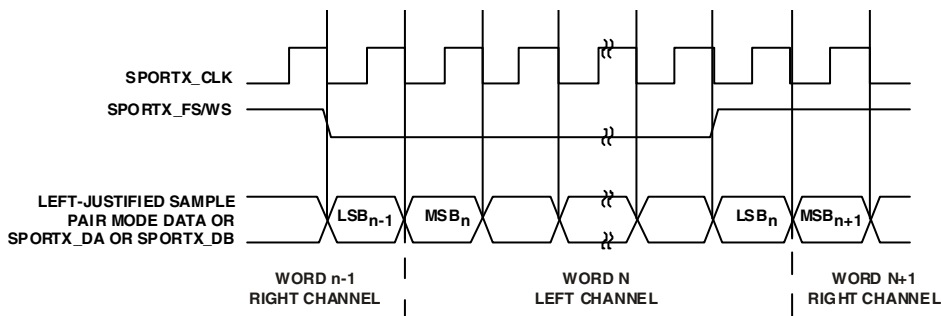


Figure 5-4. Word Select Timing in I²S Mode

serial bit stream occupies a separate channel. Each word belongs to the next consecutive channel. For example, a 24-word block of data contains one word for each of the 24 channels.

i In previous SHARC processors, μ -law and A-law compression/decompression hardware companding was available on transmitted and received words when the SPORT operated in TDM mode. In the ADSP-21367/8/9 and ADSP-2137x processors companding logic is available only in SPORT0, 2, 4, and 6 A channels and expansion logic is available only in SPORT1, 3, 5, and 7 A channels.

The SPORT can automatically select some words for particular channels while ignoring others. Up to 128 channels are available for transmitting or receiving or both. Each SPORT can receive or transmit data selectively from any of the 128 channels.

Data companding and DMA transfers can also be used in multichannel mode on channel A. Channel B can also be used in multichannel mode, but companding is not available on this channel.

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

1. The primary A channels of SPORT1, 3, 5, and 7 are capable of expansion only, and the primary A channels of SPORT0, 2, 4, and 6 are capable of companding only.
2. In previous SHARC processors in multichannel mode, SPORTs worked in pairs where, for example, SPORT0 was the transmit channel, and SPORT1 was the receive channel. In the ADSP-21367/8/9 and ADSP-2137x processors, the SPORTs work independently.

Since the SPORTs now work independently, clocks are not internally connected. Programs need to use SRU1 to connect clocks to two SPORTs separately. [For more information, see “Group A Connections—Clock Signals” on page 4-19.](#)

3. Receive comparison is not supported.

[Figure 5-5](#) shows an example of timing for a multichannel transfer with SPORT pairing using SPORT0 and 1. The transfer has the following characteristics:

- The transfer uses the TDM method where serial data is sent or received on different channels while sharing the same serial bus.
- SPORT1 is configured as the receiver and `SPORT1_FS` signals the start of a frame for each multichannel SPORT pairing.
- Unlike previous SHARC processors where the `SPORT0_FS` was used as a transmit data valid signal for external logic and was active only on transmit channels, in the ADSP-21367/8/9 and ADSP-2137x processors, `SPORT0_FS` is not used as a transmit data valid signal. Each SPORT has a dedicated data valid pin (`SPORTx_TDV`) and programs use SRU1 to connect them to external pins.

SPORT Operation Modes

- The multichannel transfer is received on channel 0 (word 0), and transmits on channels 1 and 2 (word 1 and 2).

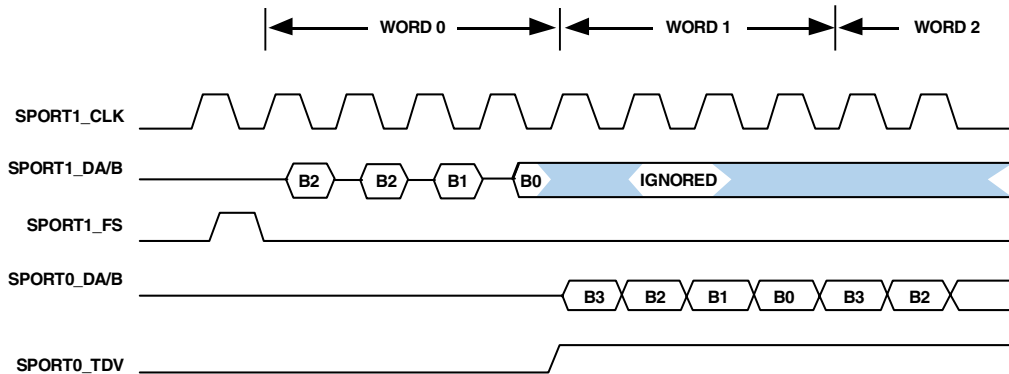


Figure 5-5. Multichannel Operation

Frame Syncs in Multichannel Mode

In previous SHARC processors, all receiving and transmitting devices in a multichannel system had to use the same timing reference. In the ADSP-21367/8/9 and ADSP-2137x processors, SPORTs operate independently and each use its own frame sync signal programmed using SRU1. [For more information, see “Group C Connections—Frame Sync Signals” on page 4-31.](#)

The frame sync signal synchronizes the channels and restarts each multichannel sequence. The `SPORTx_FS` signal initiates the beginning of the channel 0 data word.

In the ADSP-21367/8/9 and ADSP-2137x processors, each SPORT has its own transmit data valid signal, which is active during transmission of an enabled word.

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

Active State Multichannel Frame Sync Select

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

Multichannel Mode Control Bits

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

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



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

The SPCTLx control registers contain several bits that enable and configure multichannel operations. Refer to [Table 5-9 on page 5-59](#).

Multichannel mode is enabled by setting the MCEA or MCEB bit in the SPMCTL0 through SPMCTL7 control register:

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

SPORT Operation Modes

Multichannel operation is activated three serial clock cycles after the MCEA/MCEB bits are set. Internally-generated frame sync signals activate four serial clock cycles after the MCEA/MCEB bits are set.

Select the number of channels used in multichannel operation by using the 7-bit NCH field in the multichannel control register. Set NCH to the actual number of channels minus one:

$$NCH = \text{Number of channels} - 1$$

The 7-bit CHNL field in the multichannel control registers indicates the channel that is currently selected during multichannel operation. This field is a read-only status indicator. The CHNL(6:0) bits increment modulo NCH(6:0) as each channel is serviced.

The 4-bit MFD field (bits 4-1) in the multichannel control registers (SPMCTL0-7) specifies a delay between the frame sync pulse and the first data bit in multichannel mode. The value of MFD is the number of serial clock cycles of the delay. Multichannel frame delay allows the processor to work with different types of telephony interface devices.

A value of zero for MFD causes the frame sync to be concurrent with the first data bit. The maximum value allowed for MFD is 15. A new frame sync may occur before data from the last frame has been received, because blocks of data occur back-to-back.

Multichannel Frame Sync Source

Bit 14 (IMFS) in the SPCTLx registers selects whether the SPORT uses an internally-generated frame sync (if set, =1) or frame sync from an external (if cleared, =0) source.

Multichannel Status Bits

Bits 29 and 26 (DERR_A and DERR_B) in the SPCTLx registers provide error status information for the SPORT A and B channels. When the SPORT is configured as a receiver this bit indicates the receiver overflow condition.

This bit is set when the channel has received new data while the $RXSP_{xA}$ or $RXSP_{xB}$ buffer is full. New data then overwrites existing data. When the SPORT is configured as a transmitter this bit indicates the transmit underflow status (sticky, read-only). This bit is set, = 1 when multichannel $SPORT_{x_FS}$ signal (from internal or external source) occurred while the TXS buffer was empty.

Bits 31-30 and bits 28-27 (DXS_B) in the $SPCTL_x$ registers indicate the buffer status of the channel A and B buffer contents as follows: 00 = buffer empty, 01 = reserved, 10 = buffer partially full, 11 = buffer full.

Channel Selection Registers

Specific channels can be individually enabled or disabled to select the words that are received and transmitted during multichannel communications. Data words from the enabled channels are received or transmitted, while disabled channel words are ignored. Up to 128 channels are available for transmitting and receiving.

The multichannel selection registers enable and disable individual channels. The registers for each SPORT are shown in [Table 5-2](#).

Table 5-2. Multichannel Selection Registers

Register Names	Function
SP1CS(0–3) SP3CS(0–3) SP5CS(0–3) SP7CS(0–3) SP0CS(0–3) SP2CS(0–3) SP4CS(0–3) SP6CS(0–3)	Multichannel Active Channels Select. Specifies the active transmit/receive channels (4x32-bit registers for 128 channels).

SPORT Operation Modes

Table 5-2. Multichannel Selection Registers (Cont'd)

Register Names	Function
SP1CCS(0–3) SP3CCS(0–3) SP5CCS(0–3) SP7CCS(0–3)	Multichannel Receive Compand Select. Specifies which active receive channels (out of 128 channels) are companded.
SP0CCS(0–3) SP2CCS(0–3) SP4CCS(0–3) SP6CCS(0–3)	Multichannel Transmit Compand Select. Specifies which active transmit channels (out of 128 channels) are companded.

Each of the four multichannel enable and compand select registers are 32 bits in length. These registers provide channel selection for 128 (32 bits x 4 channels = 128) channels. Setting a bit enables that channel so that the SPORT selects its word from the multiple-word block of data (for either receive or transmit). For example, setting bit 0 in SP0CS0 or SP2CS0 selects word 0, setting bit 12 selects word 12, and so on. Setting bit 0 in SP0CS1 or SP2CS1 selects word 32, setting bit 12 selects word 44, and so on.

Setting a particular bit to 1 in the SP_xCS (0–3) registers causes the SPORT_x to transmit or receive the word in that channel's position of the data stream. Clearing the bit in the register causes the SPORT_x_DA data transmit signal to three-state during the time slot of that channel if the SPORT is configured as transmitter. If the SPORT is configured as receiver, the data received is ignored.

Companding may be selected on a per-channel basis. Setting a bit to 1 in any of the multichannel registers specifies that the data be companded for that channel. A-law or μ -law companding can be selected using the DTYPE bit in the SPCTL_x control registers. SPORT1, 3, 5 and 7 expand selected incoming time slot data, while SPORT0, 2, 4 and 6 can compand the data.

Packed I²S Mode

A packed I²S mode is available in ADSP-21367/8/9 SHARC processor serial ports. This mode allows applications to send more than the standard 32 bits per channel available through standard I²S mode. Packed I²S mode is implemented using standard TDM mode (and is therefore programmed similarly to TDM mode). Packed I²S mode also supports the maximum of 128 channels as does TDM mode as well as the maximum of (128 x 32) bits per left or right channel.

As shown in [Figure 5-6](#), packed I²S waveforms are the same as the waveforms used in TDM mode, except that the frame sync is toggled for every frame, and therefore emulates I²S mode.

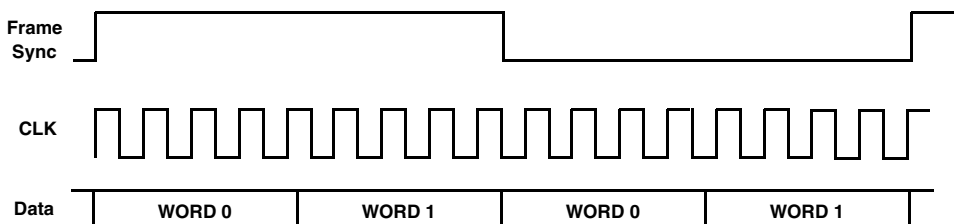


Figure 5-6. Packed I²S Mode Operation

Programming Packed I²S Mode

Since packed I²S mode is implemented on top of TDM, programming this modes is the same as programming TDM. Use the serial port control (SPCTLx) and channel selection registers to configure the serial ports to run in packed I²S mode as follows.

1. Set the OPMODE bit (bit 11, = 1) in the SPCTLx register to configure the processor to run in packed I²S mode.
2. Configure the ICLK (bit 10) and IFS (bit 14) bits to emulate I²S master or slave which in this case is internal or external clock and frame sync.

In the I²S standard, the master generates both clock and frame sync and the slave uses external clock and frame sync.

3. Configure the channel select registers. See [“Channel Selection Registers” on page 5-31](#).
4. Set the CKRE bit (bit 12) to 1 in both transmitter and receiver to emulate I²S mode.

In the I²S standard, the transmitter drives output at the clock falling edge and the receiver samples the input at the clock rising edge. This does not apply to packed I²S mode.

5. Clear (= 0) the LSBF bit in both transmitter and receiver to emulate I²S mode. In the I²S standard, the MSB of a word is sent first. In packed I²S mode, this is not the case.
6. To emulate I²S in packed I²S mode, set (=1) the MFD bit in the SPMCTLx register. In TDM mode, the transmitter starts to transmit the first bit of data immediately upon the frame sync pulsing high. In the I²S standard, data transmission starts one cycle after the frame sync pulse toggles.

SPORT Loopback

When the SPORT loopback bit, SPL bit 12, is set in the SPMCTLx, control registers, the SPORT is configured in an internal loopback connection as follows: SPORT0 and SPORT1 work as a pair for internal loopback, SPORT2 and SPORT3 work as pairs, SPORT4 and SPORT5 and SPORT6 and SPORT7 work as pairs. The loopback mode enables programs to internally test the SPORTs and to debug applications.

When loopback is configured:

- SPORTx_DA, SPORTx_DB, SPORTx_CLK and SPORTx_FS signals of SPORT0 and SPORT1 are internally connected (where x = 0 or 1).
- SPORTy_DA, SPORTy_DB, SPORTy_CLK, and SPORTy_FS signals of SPORT2 and SPORT3 are internally connected (where y = 2 or 3).
- SPORTz_DA, SPORTz_DB, SPORTz_CLK and SPORTz_FS signals of SPORT4 and SPORT5 are internally connected (where z = 4 or 5).
- SPORTn_DA, SPORTn_DB, SPORTn_CLK and SPORTn_FS signals of SPORT6 and SPORT7 are internally connected (where n = 6 or 7).

In loopback mode, either of the two paired SPORTs can be transmitters or receivers. One SPORT in the loopback pair must be configured as a transmitter; the other must be configured as a receiver. For example, SPORT0 can be a transmitter and SPORT1 can be a receiver for internal loopback. Or, SPORT0 can be a receiver and SPORT1 can be the transmitter when setting up internal loopback. The processor ignores external activity on the SPORTx_CLK, SPORTx_FS A and B channel data signals when the SPORT is configured in loopback mode which prevents contention with the internal loopback data transfer.



Only transmit clock and transmit frame sync options may be used in loopback mode—programs must ensure that the SPORT is set up correctly in the SPCTLx control registers. Multichannel mode is

Clock Signal Options

not allowed. Only standard DSP serial, left-justified sample pair, and I²S modes support internal loopback. In loopback, each SPORT can be configured as transmitter or receiver, and each SPORT is capable of generating an internal frame sync and clock.

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

Clock Signal Options

Each SPORT has a clock signal (SPORT_x_CLK) for transmitting and receiving data on the two associated data signals. The clock signals are configured by the ICLK and CKRE bits of the SPCTL_x control registers. A single clock signal clocks both A and B data signals (either configured as inputs or outputs) to receive or transmit data at the same rate.

The serial clock can be independently generated internally or input from an external source. The ICLK bit of the SPCTL_x control registers determines the clock source.

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

When ICLK is cleared (=0), the clock signal is accepted as an input on the SPORT_x_CLK signals, and the serial clock divisors in the DIV_x registers are ignored. The externally-generated serial clock does not need to be synchronous with the processor's system clock. Refer to [Table 5-10 on page 5-70](#).

Frame Sync Options

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

Framed Versus Unframed Frame Syncs

The use of frame sync signals is optional in SPORT communications. The `FSR` (transmit frame sync required) control bit determines whether frame sync signals are required. Active low or active high frame syncs are selected using the `LFS` bit. This bit is located in the `SPCTLx` control registers.

When `FSR` is set (`=1`), a frame sync signal is required for every data word. To allow continuous transmission from the processor, each new data word must be loaded into the transmit buffer before the previous word is shifted out and transmitted.

When `FSR` is cleared (`=0`), the corresponding frame sync signal is not required. A single frame sync is required to initiate communications but it is ignored after the first bit is transferred. Data words are then transferred continuously in what is referred to as an unframed mode.



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

Figure 5-7 illustrates framed serial transfers.

Frame Sync Options

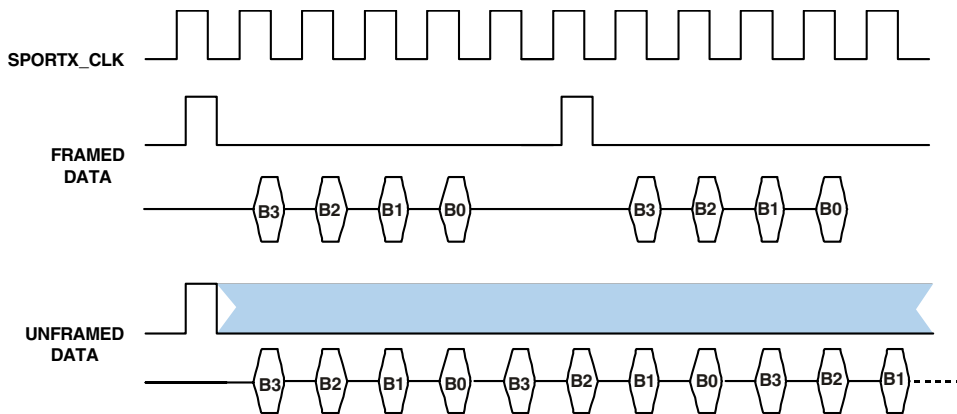


Figure 5-7. Framed Versus Unframed Data

Internal Versus External Frame Syncs

Both transmit and receive frame syncs can be generated internally or input from an external source. The `IFS` bit of the `SPCTLx` control registers determines the frame sync source.

When `IFS` is set (`=1`), the corresponding frame sync signal is generated internally by the processor, and the `SPORTx_FS` signal is an output. The frequency of the frame sync signal is determined by the value of the frame sync divisor (`FSDIV`) in the `DIVx` registers. Refer to [Figure 5-10 on page 5-70](#).

When `IFS` is cleared (`=0`), the corresponding frame sync signal is accepted as an input on the `SPORTx_FS` signals, and the frame sync divisors in the `DIVx` registers are ignored.

All frame sync options are available whether the signal is generated internally or externally.

Active Low Versus Active High Frame Syncs

Frame sync signals may be active high or active low (for example, inverted). The `LFS` bit (bit 16) of the `SPCTLx` control registers determines the frame sync's logic level:

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

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

Sampling Edge for Data and Frame Syncs

Data and frame syncs can be sampled on the rising or falling edges of the `SPORT` clock signals. The `CKRE` bit of the `SPCTLx` control registers selects the sampling edge.

For sampling receive data and frame syncs, setting `CKRE` to 1 in the `SPCTLx` registers selects the rising edge of `SPORTx_CLK`. When `CKRE` is cleared (`=0`), the processor selects the falling edge of `SPORTx_CLK` for sampling receive data and frame syncs. Note that transmit data and frame sync signals change their state on the clock edge that is not selected.

For example, the transmit and receive functions of any two `SPORT`s connected together should always select the same value for `CKRE` so internally-generated signals are driven on one edge and received signals are sampled on the opposite edge.

Early Versus Late Frame Syncs

Frame sync signals can be early or late. Frame sync signals can occur during the first bit of each data word or during the serial clock cycle immediately preceding the first bit. The `LAFS` bit of the `SPCTLx` control registers configures this option.

When `LAFS` is cleared (`=0`), early frame syncs are configured. This is the normal mode of operation. In this mode, the first bit of the transmit data word is available (and the first bit of the receive data word is latched) in the serial clock cycle after the frame sync is asserted. The frame sync is not checked again until the entire word has been transmitted (or received). In multichannel operation, this is the case when the frame delay is one clock cycle.

If data transmission is continuous in early framing mode (for example, the last bit of each word is immediately followed by the first bit of the next word), the frame sync signal occurs during the last bit of each word. Internally-generated frame syncs are asserted for one clock cycle in early framing mode.

When `LAFS` is set (`=1`), late frame syncs are configured. In this mode, the first bit of the transmit data word is available (and the first bit of the receive data word is latched) in the same serial clock cycle that the frame sync is asserted. In multichannel operation, this is the case when frame delay is zero. Receive data bits are latched by serial clock edges, but the frame sync signal is checked only during the first bit of each word. Internally-generated frame syncs remain asserted for the entire length of the data word in late framing mode. Externally-generated frame syncs are only checked during the first bit. They do not need to be asserted after that time period.

Figure 5-8 illustrates the two modes of frame signal timing.

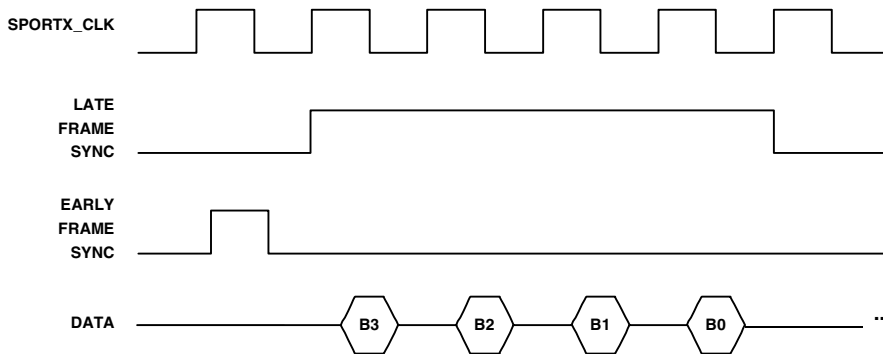


Figure 5-8. Normal Versus Alternate Framing

Data-Independent Frame Syncs

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

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

When $DIFS = 0$ and $SPTRAN = 1$, the internally-generated transmit frame sync is only output when a new data word has been loaded into the SPORT channel's transmit buffer. Once data is loaded into the transmit buffer, it is not transmitted until the next frame sync is generated. This

Frame Sync Options

mode of operation allows data to be transmitted only at specific times. When $DIFS = 0$ and $SPTRAN = 0$, a receive $SPORTx_FS$ signal is generated only when receive data buffer status is not full.

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

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

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

Frame Sync Error Detection

Similar to the SPORTs on previous SHARC processors, the SPORTs can detect underflow and overflow errors. In addition to this, the SPORTs on the ADSP-21367/8/9 and ADSP-2137x processors can also detect frame syncs that are occurring early, even before the last transmit or receive completes.

To detect these errors, these processors have a new error interrupt that works for all eight SPORTs together. It is triggered on a data underflow, data overflow, or frame sync error in their respective channels.

Unlike previous SHARC processors where programs had to poll the SPORT control registers, on the ADSP-21367/8/9 and ADSP-2137x processors, an interrupt is triggered and programs simply read the SPERRSTAT register ([Figure 5-9](#)). This reduces the processor overhead needed to do the register polling.

The frame sync error (which sets the error bit) is triggered when an early frame sync occurs during data transmission or reception. However, the current transmit/receive operation continues without interruption. Note that a frame sync error is not detected in following cases.

- When there is no active data transmit/receive and the frame sync pulse occurs due to noise in the input signal an error is not generated (and is considered as valid frame sync).
- If there is a underflow or overflow error, frame sync errors are not detected.

Each SPORT can generate an interrupt if a DERRA, DERRB, or FSYNCERR error occurs. The SPERRCTLx registers control and report the status of the interrupts generated by each SPORT (see [Figure 5-9](#)).

Data Word Formats

The format of the data words transmitted over the SPORTs is configured by the DTYPE, LSBF, SLEN, and PACK bits of the SPCTLx control registers. This is discussed in the following sections.

Word Length

Serial ports can process word lengths of 3 to 32 bits for serial and multi-channel modes and 8 to 32 bits for I²S and left-justified modes. Word length is configured using the 5-bit SLEN field in the SPCTLx control registers. Refer to [Table 5-1 on page 5-11](#) for further information.

Data Word Formats

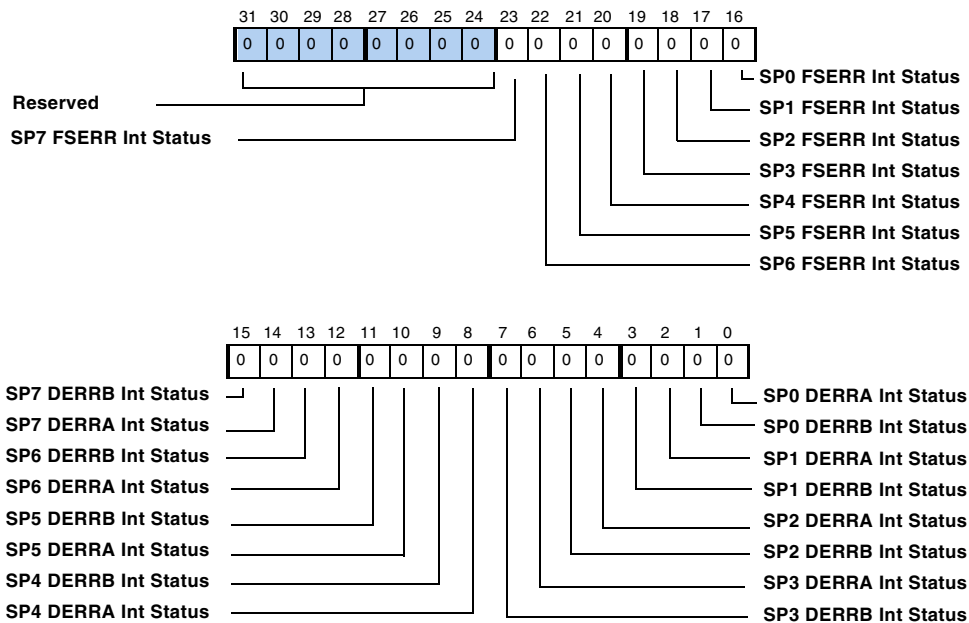


Figure 5-9. SPERRSTAT Register

The value of `SLEN` is:

$$\text{SLEN} = \text{serial word length} - 1$$

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

Although the word lengths can be 3 to 32 bits, transmitting or receiving words smaller than 7 bits at one-quarter the full clock rate of the SPORT may cause incorrect operation when DMA chaining is enabled. Chaining disables the processor's internal I/O bus for several cycles while the new transfer control block (TCB) parameters are being loaded. Receive data may be lost (for example, overwritten) during this period.

Transmitting or receiving words smaller than 5 bits may cause incorrect operation when all the DMA channels are enabled with no DMA chaining.

Endian Format

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

Data Packing and Unpacking

Received data words of 16 bits or less may be packed into 32-bit words, and 32-bit words being transmitted may be unpacked into 16-bit words. Word packing and unpacking is selected by the `PACK` bit in the `SPCTLx` control registers.

When `PACK = 1` in the control registers, two successive words received are packed into a single 32-bit word, and each 32-bit word is unpacked and transmitted as two 16-bit words.

The first 16-bit (or smaller) word is right-justified in bits 15–0 of the packed word, and the second 16-bit (or smaller) word is right-justified in bits 31–16. This applies to both receive (packing) and transmit (unpacking) operations. Companding can be used when word packing or unpacking is being used.

When `SPORT` data packing is enabled, the transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.



When 16-bit received data is packed into 32-bit words and stored in normal word space in processor internal memory, the 16-bit words can be read or written with short word space addresses.

Data Type

The `DTYPE` field of the `SPCTLx` control registers specifies one of four data formats (for non-multichannel operation) shown in [Table 5-3](#). This bit field is reserved in I²S and left-justified mode. In DSP serial mode, if companding is selected for primary A channel, the secondary B channel performs a zero-fill.



In multichannel mode, channel B looks at `XDTYPE[0]` only.

If `DTYPE[0] = 1` sign-extend

If `DTYPE[0] = 0` zero-fill

Table 5-3. `DTYPE` and Data Formatting (DSP Serial Mode)

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

These formats are applied to serial data words loaded into the receive and transmit buffers. Transmit data words are not zero-filled or sign-extended, because only the significant bits are transmitted (shown in [Table 5-4](#)).

Table 5-4. `DTYPE` and Data Formatting (Multichannel)


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

Linear transfers occur in the primary channel, if the channel is active and companding is not selected for that channel. Companded transfers occur if the channel is active and companding is selected for that channel. The multichannel compand select registers, `SPxCCSy`, specify the transmit and receive channels that are companded when multichannel mode is enabled. [For more information, see “SPORT Compand Registers \(SPxCCSy\)” on page A-47.](#)

Transmit or receive sign extension is selected by bit 0 of `DTYPE` in the `SPCTLx` registers and is common to all transmit or receive channels. If bit 0 of `DTYPE` is set, sign extension occurs on selected channels that do not have companding selected. If this bit is not set, the word contains zeros in the MSB positions. Companding is not supported for B channel. For B channels, transmit or receive sign extension is selected by bit 0 of `DTYPE` in the `SPCTLx` registers.

Companding

Companding (compressing/expanding) is the process of logarithmically encoding and decoding data to minimize the number of bits that must be sent. The processor’s SPORTs support the two most widely used companding algorithms, A-law and μ -law, performed according to the CCITT G.711 specification. The type of companding can be selected independently for each SPORT. Companding is selected by the `DTYPE` field of the `SPCTLx` registers.

-  Companding is supported on the A channel only. SPORT0, 2, 4 and 6 primary channels are capable of compression, while SPORTs 1, 3, 5 and 7 primary channels are capable of expansion. In multichannel mode, when companding and expansion is enabled, the number of channels must be programmed via the `NCH` bit in the `SPMCTLx` registers before writing to the transmit FIFO. The `SPxCSn` and `SPxCCSn` registers should also be written before writing to transmit FIFO.

Data Word Formats

When companding is enabled, the data in the `RXSPxA` buffers is the right-justified, sign-extended expanded value of the eight received LSBs. A write to `TXSPxA` compresses the 32-bit value to eight LSBs (zero-filled to the width of the transmit word) before it is transmitted. If the 32-bit value is greater than the 13-bit A-law or 14-bit μ -law maximum, it is automatically compressed to the maximum value.

Since the values in the transmit and receive buffers are actually companded in place, the companding hardware can be used without transmitting (or receiving) any data, for example during testing or debugging. This operation requires one peripheral clock cycle of overhead, as described below. For companding to execute properly, program the SPORT registers prior to loading data values into the SPORT buffers.

To compand data in place without transmitting:

1. Set the `SPTRAN` bit to 1 in the `SPCTLx` registers. The `SPEN_A` and `SPEN_B` bits should be =0.
2. Enable companding in the `DTYPE` field of the `SPCTLx` transmit control registers.
3. Write a 32-bit data word to the transmit buffer. Companding is calculated in this cycle.
4. Wait two cycles. Any instruction not accessing the Tx buffer can be used to cause this delay. This allows the SPORT companding hardware to reload the transmit buffer with the companded value.
5. Read the 8-bit companded value from the transmit buffer.

To expand data in place, use the same sequence of operations with the receive buffer instead of the transmit buffer. When expanding data in this way, set the appropriate serial word length (`SLEN`) in the `SPCTLx` registers.

With companding enabled, interfacing the SPORT to a codec requires little additional programming effort. If companding is not selected, two formats are available for received data words of fewer than 32 bits—one that fills unused MSBs with zeros, and another that sign-extends the MSB into the unused bits.

SPORT Control Registers and Data Buffers

The ADSP-21367/8/9 and ADSP-2137x processors have up to eight SPORTs. Each SPORT has two data paths corresponding to channel A and channel B. These data buffers are `TXSPxA` and `RXSPxA` (primary) and `TXSPxB` and `RXSPxB` (secondary). Channel A and B in all eight SPORTS operate synchronously to their respective `SPORTx_CLK` and `SPORTx_FS` signals. Companding is supported only on primary A channels.

The registers used to control and configure the SPORTs are part of the IOP register set. Each SPORT has its own set of 32-bit control registers and data buffers. The SPORT registers are described in [Table 5-5](#) through [Table 5-8](#).

The SPORT control registers are programmed by writing to the appropriate address in memory. The symbolic names of the registers and individual control bits can be used in programs. The definitions for these symbols are contained in the file `def21367.h` located in the `INCLUDE` directory of the ADSP-21xxx DSP development software. All control and status bits in the SPORT registers are active high unless otherwise noted.

Since the SPORT registers are memory-mapped, they cannot be written with data directly. Instead, they must be written from (or read into) processor core registers, usually one of the general-purpose universal registers (`R0-R15`) of the register file or one of the general-purpose universal status registers (`USTAT1-USTAT4`).

SPORT Control Registers and Data Buffers

[Table 5-5](#) through [Table 5-8](#) provides a complete list of the SPORT registers in IOP address order, showing the memory-mapped IOP address and a brief description of each register.

Table 5-5. SPORT0 and SPORT1 Registers

Register Name	Function	Width	No. of Registers	Memory Map [17:0]
SPCTL0–1	SPORT Control Register for SPORT0, SPORT1	32	2	00C00–00C01
DIV0–1	Clock and Frame Sync Divisors for SPORT0, SPORT1	32	2	00C02–00C03
SPMCTL0	SPORT Multichannel Control Register for SPORT0	32	1	00C04
SP0CS0–3	Multichannel Active channels select for SPORT0	32	4	00C05–00C08
SP1CS0–3	Multichannel Active channels select for SPORT1	32	4	00C09–00C0C
SP0CCS0–3	Multichannel Transmit Compand Select (128 channels) for SPORT0	32	4	00C0D–00C10
SP1CCS0–3	Multichannel Receive Compand Select (128 channels) for SPORT1	32	4	00C11–00C14
SPCNT0–1	Clock and Frame Sync Divider Counter (Internal Use Only) for SPORT0, SPORT1	32	2	00C15–00C16
SPMCTL1	SPORT Multichannel Control Register for SPORT1	32	1	00C17
SPERRCTL0–1	SPORT Error Interrupt Control Register for SPORT0, SPORT1	7	2	00C18–00C19
Reserved				00C1A–00C3F
II0A	Address for DMA Channel 0A	19	1	00C40
IM0A	Internal Modifier for DMA Channel 0A	16	1	00C41
C0A	Counter for DMA Channel 0A	16	1	00C42

Table 5-5. SPORT0 and SPORT1 Registers (Cont'd)

Register Name	Function	Width	No. of Registers	Memory Map [17:0]
CP0A	Chain Pointer for DMA Chaining Channel 0A	20	1	00C43
II0B	Address for DMA Channel 0B	19	1	00C44
IM0B	Internal Modifier for DMA Channel 0B	16	1	00C45
C0B	Counter for DMA Channel 0B	16	1	00C46
CP0B	Chain Pointer for DMA Chaining Channel 0B	20	1	00C47
II1A	Address for DMA Channel 1A	19	1	00C48
IM1A	Internal Modifier for DMA Channel 1A	16	1	00C49
C1A	Counter for DMA Channel 1A	16	1	00C4A
CP1A	Chain Pointer for DMA Chaining Channel 1A	20	1	00C4B
II1B	Address for DMA Channel 1B	19	1	00C4C
IM1B	Internal Modifier for DMA Channel 1B	16	1	00C4D
C1B	Counter for DMA Channel 1B	16	1	00C4E
CP1B	Chain Pointer for DMA Chaining Channel 1B	20	1	00C4F
Reserved				00C50–00C5F
TX0A	Transmitter FIFO Register in SP0A	32	1	00C60
RX0A	Receiver FIFO Register in SP0A	32	1	00C61
TX0B	Transmitter FIFO Register in SP0B	32	1	00C62
RX0B	Receiver FIFO Register in SP0B	32	1	00C63
TX1A	Transmitter FIFO Register in SP1A	32	1	00C64
RX1A	Receiver FIFO Register in SP1A	32	1	00C65

SPORT Control Registers and Data Buffers

Table 5-5. SPORT0 and SPORT1 Registers (Cont'd)

Register Name	Function	Width	No. of Registers	Memory Map [17:0]
TX1B	Transmitter FIFO Register in SP1B	32	1	00C66
RX1B	Receiver FIFO Register in SP1B	32	1	00C67

Table 5-6. SPORT2 and SPORT3 Registers

Register Name	Function	Width	No. of Registers	Memory Map [17:0]
SPCTL2–3	SPORT Control Register for SPORT2, 3	32	2	00400–00401
DIV2–3	Clock and Frame Sync Divisors for SPORT2, SPORT3	32	2	00402–00403
SPMCTL2	SPORT Multichannel Control Register for SPORT2	32	1	00404
SP2CS0–3	Multichannel Active channels select for SPORT2	32	4	00405–00408
SP3CS0–3	Multichannel Active channels select for SPORT3	32	4	00409–0040C
SP2CCS0–3	Multichannel Transmit Compand Select (128 channels) for SPORT2	32	4	0040D–00410
SP3CCS0–3	Multichannel Receive Compand Select (128 channels) for SPORT3	32	4	00411–00414
SPCNT2–3	Clock and Frame Sync; Divider Counter (Internal Use Only) for SPORT2, SPORT3	32	2	00415–00416
SPMCTL3	SPORT Multichannel Control Register for SPORT3	32	1	00417
SPERRCT2–3	SPORT Error Interrupt Control Register for SPORT2, SPORT3	7	2	00418–00419
Reserved				0041A–0043F
II2A	Address for DMA Channel 2A	19	1	00440

Table 5-6. SPORT2 and SPORT3 Registers (Cont'd)

Register Name	Function	Width	No. of Registers	Memory Map [17:0]
IM2A	Internal Modifier for DMA Channel 2A	16	1	00441
C2A	Counter for DMA Channel 2A	16	1	00442
CP2A	Chain Pointer for DMA Chaining Channel 2A	20	1	00443
II2B	Address for DMA Channel 2B	19	1	00444
IM2B	Internal Modifier for DMA Channel 2B	16	1	00445
C2B	Counter for DMA Channel 2B	16	1	00446
CP2B	Chain Pointer for DMA Chaining Channel 2B	20	1	00447
II3A	Address for DMA Channel 3A	19	1	00448
IM3A	Internal Modifier for DMA Channel 3A	16	1	00449
C3A	Counter for DMA Channel 3A	16	1	0044A
CP3A	Chain Pointer for DMA Chaining Channel 3A	20	1	0044B
II3B	Address for DMA Channel 3B	19	1	0044C
IM3B	Internal Modifier for DMA Channel 3B	16	1	0044D
C3B	Counter for DMA Channel 3B	16	1	0044E
CP3B	Chain Pointer for DMA Chaining Channel 3B	20	1	0044F
Reserved				00450–0045F
TX2A	Transmitter FIFO Register in SP2A	32	1	00460
RX2A	Receiver FIFO Register in SP2A	32	1	00461
TX2B	Transmitter FIFO Register in SP2B	32	1	00462
RX2B	Receiver FIFO Register in SP2B	32	1	00463
TX3A	Transmitter FIFO Register in SP3A	32	1	00464

SPORT Control Registers and Data Buffers

Table 5-6. SPORT2 and SPORT3 Registers (Cont'd)

Register Name	Function	Width	No. of Registers	Memory Map [17:0]
RX3A	Receiver FIFO Register in SP3A	32	1	00465
TX3B	Transmitter FIFO Register in SP3B	32	1	00466
RX3B	Receiver FIFO Register in SP3B	32	1	00467

Table 5-7. SPORT4 and SPORT5 Registers

Register Name	Function	Width	No. of Registers	Memory Map [17:0]
SPCTL4–5	SPORT Control Register for SPORT4, 5	32	2	00800–00801
DIV4–5	Clock and Frame Sync Divisors for SPORT4, SPORT5	32	2	00802–00803
SPMCTL4	SPORT Multichannel Control Register for SPORT4	32	1	00804
SP4CS0–3	Multichannel Active channels select for SPORT4	32	4	00805–00808
SP5CS0–3	Multichannel Active channels select for SPORT5	32	4	00809–0080C
SP4CCS0–3	Multichannel Transmit Compand Select (128 channels) for SPORT4	32	4	0080D–00810
SP5CCS0–3	Multichannel Receive Compand Select (128 channels) for SPORT5	32	4	00811–00814
SPCNT4–5	Clock and Frame Sync. Divider Counter (Internal Use Only) for SPORT4, SPORT5	32	2	00815–00816
SPMCTL5	SPORT Multichannel Control Register for SPORT5	32	1	00817
SPERRCTL 4–5	SPORT Error Interrupt Control Register for SPORT4, SPORT5	7	2	00818–00819
Reserved				0081A–0083F

Table 5-7. SPORT4 and SPORT5 Registers (Cont'd)

Register Name	Function	Width	No. of Registers	Memory Map [17:0]
II4A	Address for DMA Channel 4A	19	1	00840
IM4A	Internal Modifier for DMA Channel 4A	16	1	00841
C4A	Counter for DMA Channel 4A	16	1	00842
CP4A	Chain Pointer for DMA Chaining Channel 4A	20	1	00843
II4B	Address for DMA Channel 4B	19	1	00844
IM4B	Internal Modifier for DMA Channel 4B	16	1	00845
C4B	Counter for DMA Channel 4B	16	1	00846
CP4B	Chain Pointer for DMA Chaining Channel 4B	20	1	00847
II5A	Address for DMA Channel 5A	19	1	00848
IM5A	Internal Modifier for DMA Channel 5A	16	1	00849
C5A	Counter for DMA Channel 5A	16	1	0084A
CP5A	Chain Pointer for DMA Chaining Channel 5A	20	1	0084B
II5B	Address for DMA Channel 5B	19	1	0084C
IM5B	Internal Modifier for DMA Channel 5B	16	1	0084D
C5B	Counter for DMA Channel 5B	16	1	0084E
CP5B	Chain Pointer for DMA Chaining Channel 5B	20	1	0084F
Reserved				00850–0085F
TX4A	Transmitter FIFO Register in SP4A	32	1	00860
RX4A	Receiver FIFO Register in SP4A	32	1	00861
TX4B	Transmitter FIFO Register in SP4B	32	1	00862
RX4B	Receiver FIFO Register in SP4B	32	1	00863

SPORT Control Registers and Data Buffers

Table 5-7. SPORT4 and SPORT5 Registers (Cont'd)

Register Name	Function	Width	No. of Registers	Memory Map [17:0]
TX5A	Transmitter FIFO Register in SP5A	32	1	00864
RX5A	Receiver FIFO Register in SP35A	32	1	00865
TX5B	Transmitter FIFO Register in SP5B	32	1	00866
RX5B	Receiver FIFO Register in SP5B	32	1	00867

Table 5-8. SPORT6 and SPORT7 Registers

Register Name	Function	Width	No. of Registers	Memory Map [17:0]
SPCTL6-7	SPORT Control Register for SPORT6, SPORT7	32	2	04800-04801
DIV6-7	Clock and Frame Sync Divisors for SPORT6, 7	32	2	04802-04803
SPMCTL6	SPORT Multichannel Control Register for SPORT6	32	1	04804
SP4CS0-3	Multichannel Active channels select for SPORT6	32	4	04807-04808
SP7CS0-3	Multichannel Active channels select for SPORT7	32	4	04809-0480C
SP6CCS0-3	Multichannel Transmit Compand Select (128 channels) for SPORT6	32	4	0480D-04810
SP7CCS0-3	Multichannel Receive Compand Select (128 channels) for SPORT7	32	4	04811-04814
SPCNT6-7	Clock and Frame Sync Divider Counter (internal use only) for SPORT6, 7	32	2	04815-04816
SPMCTL7	SPORT Multichannel Control Register for SPORT7	32	1	04817
SPERRCTL6-7	SPORT Error Interrupt Control Register for SPORT6, SPORT7	7	2	04818-04819

Table 5-8. SPORT6 and SPORT7 Registers (Cont'd)

Register Name	Function	Width	No. of Registers	Memory Map [17:0]
Reserved				0481A–0483F
II6A	Address for DMA Channel 6A	19	1	04840
IM6A	Internal Modifier for DMA Channel 6A	16	1	04841
C6A	Counter for DMA Channel 6A	16	1	04842
CP6A	Chain Pointer for DMA Chaining Channel 6A	20	1	04843
II6B	Address for DMA Channel 6B	19	1	04844
IM6B	Internal Modifier for DMA Channel 6B	16	1	04845
C6B	Counter for DMA Channel 6B	16	1	04846
CP6B	Chain Pointer for DMA Chaining Channel 6B	20	1	04847
II7A	Address for DMA Channel 7A	19	1	04868
IM7A	Internal Modifier for DMA Channel 7A	16	1	04849
C7A	Counter for DMA Channel 7A	16	1	0484A
CP7A	Chain Pointer for DMA Chaining Channel 7A	20	1	0484B
II7B	Address for DMA Channel 7B	19	1	0484C
IM7B	Internal modifier for DMA Channel 7B	16	1	0484D
C7B	Counter for DMA Channel 7B	16	1	0484E
CP7B	Chain pointer for DMA chaining Channel 7B	20	1	0484F
Reserved				04850–0485F
TX6A	Transmitter FIFO Register in SP4A	32	1	04860

SPORT Control Registers and Data Buffers

Table 5-8. SPORT6 and SPORT7 Registers (Cont'd)

Register Name	Function	Width	No. of Registers	Memory Map [17:0]
RX6A	Receiver FIFO Register in SP6A	32	1	04861
TX6B	Transmitter FIFO Register in SP6B	32	1	04862
RX6B	Receiver FIFO Register in SP6B	32	1	04863
TX7A	Transmitter FIFO Register in SP7A	32	1	04864
RX7A	Receiver FIFO Register in SP7A	32	1	04867
TX7B	Transmitter FIFO Register in SP7B	32	1	04866
RX7B	Receiver FIFO Register in SP7B	32	1	04867

In the ADSP-21367/8/9 and ADSP-2137x processors, there is one global interrupt status register, `SPERRSTAT`, that checks the status of SPORT interrupts. This read-only register is located at address 0x2300 and is 24 bits wide.

Register Writes and Effect Latency

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

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

Serial Port Control Registers (SPCTLx)

The main control registers for each SPORT are the SPORT control registers, SPCTLx. These registers are described in [“SPORT Serial Control Registers \(SPCTLx\)” on page A-29](#). When changing operating modes, clear the SPORT control registers before the new mode is written to the registers.

There is one global control and status register for each SPORT (unlike previous SHARC designs where SPORTs were paired) for multichannel operation. These registers are SPMCTL0-7 and they define the number of channels, provide the status of the current channel, enable multichannel operation, and set the multichannel frame delay. These registers are described in [“SPORT Multichannel Control Registers \(SPMCTLx\)” on page A-40](#).

The SPCTLx registers control the operating modes of the SPORTs for the I/O processor. [Table 5-9](#) lists all the bits in the SPCTLx registers.

Table 5-9. SPCTLx Control Bit Comparison in Four SPORT Operation Modes

Bit	Standard DSP Serial Mode	Left-justified and I ² S Mode	Multichannel Mode
0	SPEN_A	SPEN_A	Reserved
1	DTYPE	Reserved	DTYPE
2	DTYPE	Reserved	DTYPE
3	LSBF	Reserved	LSBF
4	SLEN0	SLEN0	SLEN0
5	SLEN1	SLEN1	SLEN1
6	SLEN2	SLEN2	SLEN2
7	SLEN3	SLEN3	SLEN3
8	SLEN4	SLEN4	SLEN4
9	PACK	PACK	PACK

SPORT Control Registers and Data Buffers

Table 5-9. SPCTLx Control Bit Comparison in Four SPORT Operation Modes (Cont'd)

Bit	Standard DSP Serial Mode	Left-justified and I ² S Mode	Multichannel Mode
10	ICLK	MSTR	ICLK
11	OPMODE	OPMODE	OPMODE
12	CKRE	Reserved	CKRE
13	FSR	Reserved	Reserved
14	IFS	Reserved	IMFS
15	DIFS	DIFS	Reserved
16	LFS	FRFS	LMFS
17	LAFS	LAFS	Reserved
18	SDEN_A	SDEN_A	SDEN_A
19	SCHEN_A	SCHEN_A	SCHEN_A
20	SDEN_B	SDEN_B	SDEN_B
21	SCHEN_B	SCHEN_B	SCHEN_B
22	FS_BOTH	No effect	Reserved
23	BHD	BHD	BHD
24	SPEN_B	SPEN_B	Reserved
25	SPTRAN	SPTRAN	SPTRAN
26	ROVF_B, or TUVF_B	ROVF_B, or TUVF_B	DERR_B, ROVF_B, or TUVF_B
27	DXS_B	DXS_B	DXS_B
28	DXS_B	DXS_B	DXS_B
29	ROVF_A, or TUVF_A	ROVF_A, or TUVF_A	DERR_A, ROVF_A, or TUVF_A
30	DXS_A	DXS_A	DXS_A
31	DXS_A	DXS_A	DXS_A

The following bits, listed in bit number order, control SPORT modes and are part of the SPCTLx (transmit and receive) control registers. Other bits in the SPCTLx registers set up DMA and I/O processor-related SPORT features. For information about configuring a specific operation mode, refer to [Table 5-1 on page 5-11](#) and “Standard DSP Serial Mode” on [page 5-12](#).

Serial port enable. SPCTLx registers, bits 0 and 24 (SPEN_A and SPEN_B). These bits enable (if set, = 1) or disable (if cleared, = 0) the corresponding SPORT channel A or B. Clearing these bits aborts any ongoing operation and clears the status bits. The SPORTS are ready to transmit or receive two serial clock cycles after enabling. This description applies to I²S, left-justified sample pair, and DSP standard serial modes only.

Data type select. SPCTLx registers, bits 2–1 (DTYPE). These bits select the companding and MSB data type formatting of serial words loaded into the transmit and receive buffers. These bits applies to DSP standard serial and multichannel modes only. The transmit shift register does not zero-fill or sign-extend transmit data words; this only takes place for the receive shift register.

For standard mode, selection of companding mode and MSB format are exclusive:

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

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

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

SPORT Control Registers and Data Buffers

This description applies only to DSP standard serial and multichannel modes only.

Serial word endian select. SPCTLx registers, bit 3 (LSBF). This bit selects little endian words (LSB first, if set, = 1) or big endian words (MSB first, if cleared, = 0). This description applies to DSP standard serial and multichannel modes only.

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

Use this formula to calculate the value for SLEN:

$$\text{SLEN} = \text{actual serial word length} - 1$$



The SLEN bit cannot equal 0 or 1. I²S, and left-justified sample pair word length is limited to 8-32 bits. DSP standard mode word length varies from 3-32 bits.

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

Internal clock select. SPCTLx registers, bit 10 (ICLK). This bit selects the internal (if set, = 1) or external (if cleared, = 0) transmit or receive clock. This bit applies to DSP standard serial mode and multichannel modes.

Sport operation mode. SPCTLx registers, bit 11 (OPMODE). This bit enables I²S, left-justified sample pair, and packed I²S in multichannel modes if set (= 1), or disables if cleared (= 0). This bit applies to all operation modes. See [Table 5-1 on page 5-11](#) and [“Standard DSP Serial Mode” on page 5-12](#).

Clock rising edge select. SPCTLx registers, bit 12 (CKRE). This bit selects whether the SPORT uses the rising edge (if set, = 1) or falling edge (if cleared, = 0) of the clock signal for sampling data and the frame sync. This bit applies to DSP standard serial and multichannel modes only.

Frame sync required select. SPCTLx registers, bit 13 (FSR). This bit selects whether the SPORT requires (if set, = 1) or does not require (if cleared, = 0) a transfer frame sync. See [“Frame Sync Options” on page 5-37](#) for more details. This bit applies to DSP standard serial mode only.

Internal frame sync select. SPCTLx registers, bit 14 (IFS). This bit selects whether the SPORT uses an internally-generated frame sync (if set, = 1) or a frame sync from an external (if cleared, = 0) source. This bit is used for standard DSP serial and multichannel modes only. This bit is referred as IMFS in multichannel mode.

Low active frame sync select. SPCTLx registers, bit 16 (LFS). This bit selects the logic level of the (transmit or receive) frame sync signals. This bit selects an active low frame sync (if set, = 1) or active high frame sync (if cleared, = 0). Active high is the default. This bit is called FRFS in I²S and left-justified modes and LTDV/LMFS in multichannel mode.

Late frame sync select. SPCTLx registers, bit 17 (LAFS). This bit selects when to generate the frame sync signal. This bit selects a late frame sync if set (= 1) during the first bit of each data word. This bit selects an early frame sync if cleared (= 0) during the serial clock cycle immediately preceding the first data bit. See [“Frame Sync Options” on page 5-37](#) for more details.

This bit applies to DSP standard serial mode only. This bit is also used to select between I²S and left-justified sample pair modes. See [Table 5-1 on page 5-11](#) and [“Standard DSP Serial Mode” on page 5-12](#) for more information.

Serial port DMA enable. SPCTLx registers, bits 18 and 20 (SDEN_A and SDEN_B). These bits enable (if set, = 1) or disable (if cleared, = 0) the SPORT’s channel DMA. Bits 18 and 20 apply to all operating modes.

Serial port DMA chaining enable. SPCTLx registers, bits 19 and 21 (SCHEN_A and SCHEN_B). These bits enable (if set, = 1) or disable (if cleared, = 0) SPORT’s channels A and B DMA chaining. Bits 19 and 21 apply to all operating modes.

SPORT Control Registers and Data Buffers

Frame sync both enable. SPCTLx registers, bit 22 (FS_BOTH). This bit applies when the SPORT's channels A and B are configured to transmit/receive data. If set (= 1), this bit issues frame sync only when data is present in *both* transmit buffers, TXA and TXB. If cleared (= 0), a frame sync is issued if data is present in either transmit buffers. This bit applies to DSP standard serial mode only.

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

This bit is not used for I²S and left-justified sample pair modes. If only channel A or channel B is selected, the frame sync behaves as if FS_BOTH is cleared (= 0). If both A and B channels are selected, the word select acts as if FS_BOTH is set (= 1).

Buffer hang disable. SPCTLx registers, bit 23 (BHD). When cleared (= 0), this bit causes the processor core to hang when it attempts to write to a full buffer or read from an empty buffer. When set (= 1), this bit disables the core hang. In this case, a core read from an empty receive buffer returns previously read (invalid) data and core writes to a full transmit buffer to overwrite (valid) data that has not yet been transmitted. This bit is used in all modes.

Data direction control. SPCTLx registers, bit 25 (SPTRAN). This bit controls the data direction of the SPORT channel A and B signals.

When cleared (= 0), the SPORT is configured to receive on both channels A and B. When configured to receive, the RXSPxA and RXSPxB buffers are activated, while the receive shift registers are controlled by SPORTx_CLK and SPORTx_FS. The TXSPxA and TXSPxB buffers are inactive.

When set (= 1), the SPORT is configured to transmit on both channels A and B. When configured to transmit, the TXSPxA and TXSPxB buffers are activated, while the transmit shift registers are controlled by SPORTx_CLK and SPORTx_FS. The RXSPxA and RXSPxB buffers are inactive. This bit applies to operating modes.



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

Data buffer error status (sticky, read-only). SPCTLx registers, bits 29 and 26 (DERR_A, DERR_B). These bits indicate whether the serial transmit operation has underflowed (if set, = 1 and SPTRAN = 1) or a receive operation has overflowed (if set, = 1 and SPTRAN = 0) in the TXSPxA/RXSPxA and TXSPxB/RXSPxB data buffers.

This description applies to all operating modes. In multichannel modes, corresponding bits (TUVF, ROVF) are used for this function.

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

Specifically, the operation of the TUVF bit is:

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


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

SPORT Control Registers and Data Buffers

Specifically, the operation of the ROVF bit is:


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

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

 In the ADSP-21367/8/9 and ADSP-2137x processors, programs no longer need to poll these registers to detect an underflow or overflow condition. Instead, an interrupt is generated and programs only need to read the new SPORT error status register (SPERRSTAT) to determine what register contains the error and what the error is. [For more information, see “Frame Sync Error Detection” on page 5-42.](#)

Data buffer status. SPCTLx registers, bits 31–30 (DXS_A) and bits 28–27 (DXS_B). These read-only bits indicate the status of the SPORT’s data buffer as follows: 11 = buffer full, 00 = buffer empty, 10 = buffer partially full, 01 = reserved.


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

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

Transmit and Receive Data Buffers (TXSPxA/B, RXSPxA/B)

The transmit buffers (TXSP0A–TXSP7A and TXSP0B–TXSP7B) are the 32-bit transmit data buffers for SPORT0 through SPORT7 respectively. These buffers must be loaded with the data to be transmitted if the SPORT is configured to transmit on the A and B channels. The data is loaded automatically by the DMA controller or loaded manually by the program running on the processor core.

The receive buffers (RXSP0A–RXSP7A and RXSP0B–RXSP7B) are the 32-bit receive data buffers for SPORT0 through SPORT7 respectively. These 32-bit buffers become active when the SPORT is configured to receive data on the A and B channels. When a SPORT is configured as a receiver, the RXSPxA and RXSPxB registers are automatically loaded from the receive shifter when a complete word has been received. The data is then loaded to internal memory by the DMA controller or read directly by the program running on the processor core.

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

The transmit buffers act like a two-location FIFO because they have a data register plus an output shift register. Two 32-bit words may both be stored in the transmit queue at any one time. When the transmit register is loaded and any previous word has been transmitted, the register contents are automatically loaded into the output shifter. An interrupt occurs when the output transmit shifter has been loaded, signifying that the transmit

SPORT Control Registers and Data Buffers

buffer is ready to accept the next word (for example, the transmit buffer is not full). This interrupt does not occur when SPORT DMA is enabled or when the corresponding mask bit in the LIRPTL/IRPTL register is set.

When the SPORT is configured as a transmitter ($SPTRAN = 1$) and a transmit frame sync occurs and no new data has been loaded into the transmit buffer, a transmit underflow status bit is set in the SPORT control registers. The TUVF_A/ROVF_A or TUVF_A status bit is sticky and is only cleared by disabling the SPORT.

When the SPORT is configured as a receiver ($SPTRAN = 0$), the receive buffers are activated. The receive buffers act like a three-location FIFO because they have two data registers plus an input shift register. Two complete 32-bit words can be stored in the receive buffer while a third word is shifted in. The third word overwrites the second if the first word has not been read out (by the processor core or the DMA controller). When this happens, the receive overflow status bit is set in the SPORT control registers. Almost three complete words can be received without the receive buffer being read before an overflow occurs. The overflow status is generated on the last bit of the third word. The ROVF_A/ROVF_A or TUVF_A status bit is sticky and is cleared only by disabling the SPORT.

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

If your program causes the core processor to attempt to read from an empty receive buffer or to write to a full transmit buffer, the access is delayed until the buffer is accessed by the external I/O device. This delay is called a core processor hang. In order to access the receive or transmit buffer without a hang, read the buffer's status to determine whether the access can be made.

To support debugging buffer transfers, the ADSP-21367/8/9 and ADSP-2137x processors have a buffer hang disable (BHD) bit. When set (= 1), this bit prevents the processor core from detecting a buffer-related stall condition, permitting debugging of this type of stall condition. For more information, see the BHD bit description on [page 5-64](#).

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



When programming the SPORT channel (A or B) as a transmitter, only the corresponding TXSPxA and TXSPxB buffers become active while the receive buffers RXSPxA and RXSPxB remain inactive. Similarly, when the SPORT channel A and B are programmed as receive-only, the corresponding RXSPxA and RXSPxB are activated. Do not attempt to read or write to inactive data buffers. If the processor operates on the inactive transmit or receive buffers while the SPORT is enabled, unpredictable results may occur.

Clock and Frame Sync Frequency Registers (DIVx)

The DIVx registers contain divisor values that determine frequencies for internally-generated clocks and frame syncs. The DIVx registers are described in Appendix A in “SPORT Divisor Registers (DIVx)” on [page A-44](#) and are shown in [Figure 5-10](#).

The CLKDIV bit field specifies how many times the processor’s internal clock (CCLK) is divided to generate the transmit and receive clocks. The frame sync (SPORTx_FS) is considered a receive frame sync if the data signals are configured as receivers. Likewise, the frame sync SPORTx_FS is considered a transmit frame sync if the data signals are configured as

SPORT Control Registers and Data Buffers

transmitters. The divisor is a 15-bit value, allowing a wide range of serial clock rates. Use the following equation to calculate the serial clock frequency:

$$f_{\text{SPORTx_CLK}} = \frac{f_{\text{CCLK}}}{8(\text{CLKDIV} + 1)}$$

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

$$\text{CLKDIV} = \frac{f_{\text{CCLK}}}{8(f_{\text{SPORTx_CLK}})} - 1$$

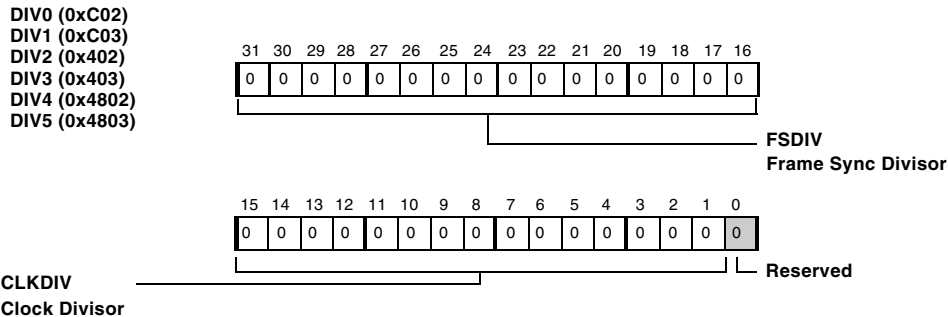


Figure 5-10. DIVx Register

The FSDIV bit field specifies how many transmit or receive clock cycles are counted before a frame sync pulse is generated. In this way, a frame sync can initiate periodic transfers. The counting of serial clock cycles applies to internally- or externally-generated serial clocks. The formula for the number of cycles between frame sync pulses is:

$$\# \text{ of serial clocks between frame syncs} = \text{FSDIV} + 1$$

Use the following equation to determine the value of $FSDIV$, given the serial clock frequency and desired frame sync frequency:

$$FSDIV = \frac{f_{SPORTx_CLK}}{f_{SPORTx_FS}} - 1$$

The frame sync is continuously active when $FSDIV = 0$. The value of $FSDIV$ should not be less than the serial word length minus one (the value of the $SLEN$ field in the SPORT control registers), as this may cause an external device to abort the current operation or cause other unpredictable results. If the SPORT is not being used, the $FSDIV$ divisor can be used as a counter for dividing an external clock or for generating a periodic pulse or periodic interrupt. The SPORT must be enabled for this mode of operation to work properly.

Exercise caution when operating with externally-generated transmit clocks near the frequency of one-eighth of the processor's internal clock. There is a delay between when the clock arrives at the $SPORTx_CLK$ node and when data is output. This delay may limit the receiver's speed of operation. Refer to the product-specific data sheet for exact timing specifications.

Externally-generated late transmit frame syncs also experience a delay from when they arrive to when data is output. This can also limit the maximum serial clock speed. Refer to the product-specific data sheet for exact timing specifications.

SPORT Reset

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

SPORT Control Registers and Data Buffers

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

A hardware reset ($\overline{\text{RESET}}$) disables the entire processor, including the SPORTs, by clearing the `SPCTLx` registers. Any ongoing operations are aborted.

SPORT Interrupts

Each SPORT has an interrupt associated with it. For each SPORT, both the A and B channel transmit and receive data buffers share the same interrupt vector. The interrupts can be used to indicate the completion of the transfer of a block of serial data when the SPORTs are configured for DMA. They can also be used to perform single word transfers (refer to [“Single Word Transfers” on page 5-81](#)). The priority of the SPORT interrupts is shown in [Table 5-10](#).

Table 5-10. Priority of the Serial Port Interrupts

Interrupt Name ¹	Interrupt
SP1I	SPORT1 DMA Channels (Highest Priority)
SP3I	SPORT3 DMA Channels
SP5I	SPORT5 DMA Channels
SP0I	SPORT0 DMA Channels
SP2I	SPORT2 DMA Channels
SP4I	SPORT4 DMA Channels
SP6I	SPORT6 DMA Channels
SP7I	SPORT7 DMA Channels

¹ The interrupt names are defined in the `def21367.h` file supplied with the ADSP-21xxx DSP development software.

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

Moving Data Between SPORTs and Internal Memory

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


SPORT DMA provides a mechanism for receiving or transmitting an entire block of serial data before the interrupt is generated. When SPORT DMA is not enabled, the SPORT generates an interrupt every time it receives or starts to transmit a data word. The processor's on-chip DMA controller handles the DMA transfer, allowing the processor core to continue running until the entire block of data is transmitted or received. Service routines can then operate on the block of data rather than on single words, significantly reducing overhead.

DMA Block Transfers

The DSP's on-chip DMA controller allows automatic DMA transfers between internal memory and each of the two channels of each SPORT. Each SPORT has two channels for transferring data, and each can be configured to receive or to transmit. There are twelve DMA channels for SPORT operations. The SPORT DMA channels are numbered as shown in [Table 5-11](#).

Moving Data Between SPORTs and Internal Memory

Table 5-11. Serial Port DMA Channels

Channel	Data Buffer	Description	Priority
0	RXSP1A/TXSP1A	SPORT1 A data	Highest
1	RXSP1B/TXSP1B	SPORT1 B data	
2	RXSP0A/TXSP0A	SPORT0 A data	
3	RXSP0B/TXSP0B	SPORT0 B data	
4	RXSP3A/TXSP3A	SPORT3 A data	
5	RXSP3B/TXSP3B	SPORT3 B data	
6	RXSP2A/TXSP2A	SPORT2 A data	
7	RXSP2B/TXSP2B	SPORT2 B data	
8	RXSP5A/TXSP5A	SPORT5 A data	
9	RXSP5B/TXSP5B	SPORT5 B data	
10	RXSP4A/TXSP4A	SPORT4 A data	
11	RXSP4B/TXSP4B	SPORT4 B data	
12	RXSP7A/TXSP7A	SPORT7 A data	
13	RXSP7B/TXSP7B	SPORT7 B data	
14	RXSP6A/TXSP6A	SPORT6 A data	
15	RXSP6B/TXSP6B	SPORT6 B data	Lowest

Data-direction programmability is supported in standard DSP standard serial, left-justified sample pair, and I²S modes. The value of the `SPTRAN` bit in `SPCTLx` (0 = RX, 1 = TX) determines whether the receive or transmit register for the SPORT becomes active.

The SPORT DMA channels are assigned higher priority than all other DMA channels (for example, the SPI port) because of their relatively low service rate and their inability to hold off incoming data. Having a higher priority causes the SPORT DMA transfers to perform first when multiple DMA requests occur in the same cycle.

Although the DMA transfers are performed with 32-bit words, SPORTs can handle word sizes from 3 to 32 bits, with 8 to 32 bits for I²S mode. If serial words are 16 bits or smaller, they can be packed into 32-bit words for each DMA transfer. DMA transfers are configured using the `PACK` bit in the `SPCTLx` registers. When SPORT data packing is enabled (`PACK = 1`), the transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.

The following sections present an overview of SPORT DMA operations; additional details are covered in the “Memory” chapter in the *ADSP-2136x SHARC Processor Programming Reference*.

- For information on SPORT DMA channel setup, see [“Setting Up DMA on SPORT Channels” on page 5-75](#).
- For information on SPORT DMA parameter registers, see [“SPORT DMA Parameter Registers” on page 5-76](#).
- For information on SPORT DMA chaining, see [“SPORT DMA Chaining” on page 5-81](#).

Setting Up DMA on SPORT Channels

Each SPORT DMA channel has an enable bit (`SDEN_A` and `SDEN_B`) in its `SPCTLx` registers. When DMA is disabled for a particular channel, the SPORT generates an interrupt every time it receives a data word or whenever there is a vacancy in the transmit buffer. For more information, see [“Single Word Transfers” on page 5-81](#).

Each channel also has a DMA chaining enable bit (`SCHEN_A` and `SCHEN_B`) in its `SPCTLx` registers.

To set up a SPORT DMA channel, write a set of memory buffer parameters to the SPORT DMA parameter registers as shown in [Table 5-12](#).

Moving Data Between SPORTs and Internal Memory

Table 5-12. SPORT DMA Parameter Registers

Register (Y = A or B, and x = 0 – 5)	Width	Description
IISP _{xy}	19 bits	DMA channel; x index; start address for data buffer
IMSP _{xy}	16 bits	DMA channel; x modify; address increment
CSP _{xy}	16 bits	DMA channel; x count; number of words to transmit
CPSP _{xy}	20 bits	DMA channel; x chain pointer; address containing the next set of data buffer parameters

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

Once SPORT DMA is enabled, the processor's DMA controller automatically transfers received data words in the receive buffer to the buffer in internal memory. Likewise, when the SPORT is ready to transmit data, the DMA controller automatically transfers a word from internal memory to the transmit buffer. The controller continues these transfers until the entire data buffer is received or transmitted.

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

SPORT DMA Parameter Registers

A DMA channel consists of a set of parameter registers that implements a data buffer in internal memory and the hardware the SPORT uses to request DMA service. The parameter registers for each SPORT DMA channel and their addresses are shown in [Table 5-13](#). These registers are part of the processor's memory-mapped IOP register set.

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

Each DMA channel has a count register (CSP_{xA}/CSP_{xB}) which must be initialized with a word count that specifies the number of words to transfer. The count register decrements after each DMA transfer on the channel. When the word count reaches zero, the SPORT generates an interrupt, then automatically disables the DMA channel.

Each SPORT DMA channel also has a chain pointer register ($CPSP_{xy}$). The $CPSP_{xy}$ register functions are used in chained DMA operations. For more information on SPORT DMA chaining registers, see [Table 5-12 on page 5-76](#).

Table 5-13. SPORT DMA Parameter Registers Addresses

Register	Address	DMA Channel	SPORT Buffer
IISP0A	0xC40	0	RXSP0A or TXSP0A
IMSP0A	0xC41	0	RXSP0A or TXSP0A
CSP0A	0xC42	0	RXSP0A or TXSP0A
CPSP0A	0xC43	0	RXSP0A or TXSP0A
IISP0B	0xC44	1	RXSP0B or TXSP0B
IMSP0B	0xC45	1	RXSP0B or TXSP0B
CSP0B	0xC46	1	RXSP0B or TXSP0B
CPSP0B	0xC47	1	RXSP0B or TXSP0B
IISP1A	0xC48	2	RXSP1A or TXSP1A

Moving Data Between SPORTs and Internal Memory

Table 5-13. SPORT DMA Parameter Registers Addresses (Cont'd)

Register	Address	DMA Channel	SPORT Buffer
IMSP1A	0xC49	2	RXSP1A or TXSP1A
CSP1A	0xC4A	2	RXSP1A or TXSP1A
CPSP1A	0xC4B	2	RXSP1A or TXSP1A
IISP1B	0xC4C	3	RXSP1B or TXSP1B
IMSP1B	0xC4D	3	RXSP1B or TXSP1B
CSP1B	0xC4E	3	RXSP1B or TXSP1B
CPSP1B	0xC4F	3	RXSP1B or TXSP1B
Reserved			
IISP2A	0x440	4	RXSP2A or TXSP2A
IMSP2A	0x441	4	RXSP2A or TXSP2A
CSP2A	0x442	4	RXSP2A or TXSP2A
CPSP2A	0x443	4	RXSP2A or TXSP2A
IISP2B	0x444	5	RXSP2B or TXSP2B
IMSP2B	0x445	5	RXSP2B or TXSP2B
CSP2B	0x446	5	RXSP2B or TXSP2B
CPSP2B	0x447	5	RXSP2B or TXSP2B
IISP3A	0x448	6	RXSP3A or TXSP3A
IMSP3A	0x449	6	RXSP3A or TXSP3A
CSP3A	0x44A	6	RXSP3A or TXSP3A
CPSP3A	0x44B	6	RXSP3A or TXSP3A
IISP3B	0x44C	7	RXSP3B or TXSP3B
IMSP3B	0x44D	7	RXSP3B or TXSP3B
CSP3B	0x44E	7	RXSP3B or TXSP3B
CPSP3B	0x44F	7	RXSP3B or TXSP3B

Table 5-13. SPORT DMA Parameter Registers Addresses (Cont'd)

Register	Address	DMA Channel	SPORT Buffer
Reserved			
IISP4A	0x840	8	RXSP4A or TXSP4A
IMSP4A	0x841	8	RXSP4A or TXSP4A
CSP4A	0x842	8	RXSP4A or TXSP4A
CPSP4A	0x843	8	RXSP4A or TXSP4A
IISP4B	0x844	9	RXSP4B or TXSP4B
IMSP4B	0x845	9	RXSP4B or TXSP4B
CSP4B	0x846	9	RXSP4B or TXSP4B
CPSP4B	0x847	9	RXSP4B or TXSP4B
IISP5A	0x848	10	RXSP5A or TXSP5A
IMSP5A	0x849	10	RXSP5A or TXSP5A
CSP5A	0x84A	10	RXSP5A or TXSP5A
CPSP5A	0x84B	10	RXSP5A or TXSP5A
IISP5B	0x84C	11	RXSP5B or TXSP5B
IMSP5B	0x84D	11	RXSP5B or TXSP5B
CSP5B	0x84E	11	RXSP5B or TXSP5B
CPSP5B	0x84F	11	RXSP5B or TXSP5B
IISP6A	0x4840	12	RXSP6A or TXSP6A
IMSP6A	0x4841	12	RXSP6A or TXSP6A
CSP6A	0x4842	12	RXSP6A or TXSP6A
CPSP6A	0x4843	12	RXSP6A or TXSP6A
IISP6B	0x4844	13	RXSP6B or TXSP6B
IMSP6B	0x4845	13	RXSP6B or TXSP6B
CSP6B	0x4846	13	RXSP6B or TXSP6B

Moving Data Between SPORTs and Internal Memory

Table 5-13. SPORT DMA Parameter Registers Addresses (Cont'd)

Register	Address	DMA Channel	SPORT Buffer
CPSP6B	0x4847	13	RXSP6B or TXSP6B
IISP7A	0x4848	14	RXSP7A or TXSP7A
IMSP7A	0x4849	14	RXSP7A or TXSP7A
CSP7A	0x484A	14	RXSP7A or TXSP7A
CPSP7A	0x484B	14	RXSP7A or TXSP7A
IISP7B	0x484C	15	RXSP7B or TXSP7B
IMSP7B	0x484D	15	RXSP7B or TXSP7B
CSP7B	0x484E	15	RXSP7B or TXSP7B
CPSP7B	0x484F	15	RXSP7B or TXSP7B
Reserved (0x850 to 0x85F)			

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

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

Therefore, set the direction bit, the SPORT enable bit, and DMA enable bits before initiating any operations on the SPORT data buffers. If the processor operates on the inactive transmit or receive buffers while the SPORT is enabled, it can cause unpredictable results.

SPORT DMA Chaining

In chained DMA operations, the processor's DMA controller automatically sets up another DMA transfer when the contents of the current buffer are transmitted (or received). The chain pointer registers (CPSP_{xy}) function as a pointer to the next set of buffer parameters stored in memory. The DMA controller automatically downloads these buffer parameters to set up the next DMA sequence. For more information on SPORT DMA chaining, see [“Setting Up DMA Parameter Registers” on page 2-24](#).


DMA chaining occurs independently for the transmit and receive channels of each SPORT. Each SPORT DMA channel has a chaining enable bit (SCHEN_A or SCHEN_B) that when set (= 1) enables DMA chaining and when cleared (= 0) disables DMA chaining. Writing all zeros to the address field of the chain pointer registers (CPSP_{xy}) also disables chaining.

Single Word Transfers

Individual data words may also be transmitted and received by the SPORTs, with interrupts occurring as each 32-bit word is transmitted or received. When a SPORT is enabled and DMA is disabled, the SPORT interrupts are generated whenever a complete 32-bit word has been received in the receive buffer, or whenever the transmit buffer is not full. Note that both channel A and B buffers share the same interrupt vector. Single word interrupts can be used to implement interrupt-driven I/O on the SPORTs.

SPORT Programming Examples

To avoid hanging the processor core, check the buffer's full/empty status when the processor core's program reads a word from a SPORT's receive buffer or writes a word to its transmit buffer. The full/empty status can be read in the `DXS` bits of the `SPCTLx` registers. Reading from an empty receive buffer or writing to a full transmit buffer causes the processor to hang, while it waits for the status to change.

 To support debugging buffer transfers, the processor has a buffer hang disable (BHD) bit. When set (= 1), this bit prevents the processor core from detecting a buffer-related stall condition, permitting debugging of this type of stall condition. For more information, see the BHD bit discussion on [page 5-64](#).

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

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

SPORT Programming Examples

This section provides three programming examples written for the ADSP-21367/8/9 and ADSP-2137x processors. The first, [Listing 5-1](#), transmits a buffer of data from `SPORT5` to `SPORT4` using DMA and the internal loopback feature of the SPORT. In this example, `SPORT5` drives the clock and frame sync, and the buffer is transferred only one time.

The second listing, [Listing 5-2](#), transmits a buffer of data from `SPORT2` to `SPORT3` using direct core reads and writes and the internal loopback feature of the SPORT. In this example, `SPORT2` drives the clock and frame sync, and the buffer is transferred only one time.

The third listing, [Listing 5-3](#), transmits a buffer of data from SPORT1 to SPORT0 using DMA chaining and the internal loopback feature of the SPORT. In this example, SPORT5 drives the clock and frame sync, and the two TCBs for each SPORT are set up to ping-pong back and forth to continually send and receive data.

Listing 5-1. SPORT Transmit Using DMA Chaining

```

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

/* SPORT Control Registers */
#define DIV4     0x802
#define DIV5     0x803
#define SPCTL4   0x800
#define SPCTL5   0x801
#define SPMCTL4  0x804

/* SPMCTL Bits */
#define SPL      0x00001000

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

```

SPORT Programming Examples

```
/* Default Buffer Length */
#define BUFSIZE 10

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

/*Receive buffer*/
.var rx_buf4a[BUFSIZE];
/* Main code section */
.global _main;
.SECTION/PM seg_pmco;
_main:

/*    SPORT Loopback: Use SPORT4 as RX & SPORT5 as TX    */

/* initially clear SPORT control register */
r0 = 0x00000000;
dm(SPCTL4) = r0;
dm(SPCTL5) = r0;
dm(SPMCTL4) = r0;

SPORT_DMA_setup:
```

```

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

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

/* set internal loopback bit for SPORT4 & SPORT5 */
bit set ustat3 SPL;
dm(SPMCTL4) = ustat3;

/* Configure SPORT5 as a transmitter */
/* internally generating clock and frame sync */
/* CLKDIV = [fCCLK(333 MHz)/2 x FSCLK(8.325 MHz)] - 1 = 0x0004 */
/* FSDIV = [FSCLK(8.325 MHz)/TFS(.26 MHz)] - 1 = 31 = 0x001F */
R0 = 0x001F0004;   dm(DIV5) = R0;
ustat4 = SPEN_A|    /* Enable Channel A */
          SLEN32|    /* 32-bit word length */
          FSR|       /* Frame Sync Required */
          SPTRAN|    /* Transmit on enabled channels */
          SDEN_A|    /* Enable Channel A DMA */
          IFS|       /* Internally Generated Frame Sync */
          ICLK;      /* Internally Generated Clock */
dm(SPCTL5) = ustat4;

/* Configure SPORT4 as a receiver */
/* externally generating clock and frame sync */

```

SPORT Programming Examples

```
r0 = 0x0;      dm(DIV4) = R0;
ustat3 = SPEN_A|    /* Enable Channel A */
          SLEN32|    /* 32-bit word length */
          FSR|      /* Frame Sync Required */
          SDEN_A;    /* Enable Channel A DMA */
dm(SPCTL4) = ustat3;

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

Listing 5-2. SPORT Transmit Using Direct Core Access

```
/* SPORT Control Registers */
#define TXSP2A  0x460
#define RXSP3A  0x465
#define DIV2    0x402
#define DIV3    0x403
#define SPCTL2  0x400
#define SPCTL3  0x401
#define SPMCTL2 0x404

/* SPMCTL Bits */
#define SPL      0x00001000

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

/* Default Buffer Length */
#define BUFSIZE 10
```

```

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

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

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

/* SPORT Loopback: Use SPORT2 as RX & SPORT3 as TX */

/* Initially clear SPORT control registers */
r0 = 0x00000000;
dm(SPCTL2) = r0;
dm(SPCTL3) = r0;
dm(SPMCTL2) = r0;

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

```

SPORT Programming Examples

```
SPORT_DMA_setup:
/* set internal loopback bit for SPORT2 & SPORT3 */
bit set ustat3 SPL;
dm(SPMCTL2) = ustat3;

/* Configure SPORT2 as a transmitter */
/* internally generating clock and frame sync */
/* CLKDIV = [fCCLK(333MHz)/2 x FSCCLK(8.325 MHz)] - 1 = 0x0004 */
/* FSDIV = [FSCCLK(8.325 MHz)/TFS(.26 MHz)] - 1 = 31 = 0x001F */
R0 = 0x001F0004;    dm(DIV2) = R0;
ustat4 = SPEN_A|    /* Enable Channel A */
          SLEN32|    /* 32-bit word length */
          FSR|       /* Frame Sync Required */
          SPTRAN|    /* Transmit on enabled channels */
          IFS|       /* Internally Generated Frame Sync */
          ICLK;      /* Internally Generated Clock */
dm(SPCTL2) = ustat4;

/* Configure SPORT3 as a receiver */
/* externally generating clock and frame sync */
r0 = 0x0;    dm(DIV3) = R0;
ustat3 = SPEN_A|    /* Enable Channel A */
          SLEN32|    /* 32-bit word length */
          FSR;       /* Frame Sync Required */
dm(SPCTL3) = ustat3;
/* Set up loop to transmit and receive data */
lcntr = LENGTH(tx_buf2a), do (pc,4) until lce;
/* Retrieve data using DAG1 and send TX via SPORT2 */
r0 = dm(i4,m4);
dm(TXSP2A) = r0;
/* Receive data via SPORT3 and save via DAG2 */
r0 = dm(RXSP3A);
pm(i12,m12) = r0;
_main.end:  jump (pc,0);
```


Listing 5-3. SPORT Transmit Using DMA

```

/* SPORT DMA Parameter Registers */
#define CPSP0A 0xC43
#define CPSP1A 0xC4B
/* SPORT Control Registers */
#define DIV0 0xC02
#define DIV1 0xC03
#define SPCTL0 0xC00
#define SPCTL1 0xC01
#define SPMCTL0 0xC04

/* SPMCTL Bits */
#define SPL 0x00001000

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

/* Default Buffer Length */
#define BUFSIZE 10

.SECTION/DMA seg_dmda;
/* TX Buffers */
.var tx_buf1a[BUFSIZE] = 0x11111111,
                        0x22222222,
                        0x33333333,

```

SPORT Programming Examples

```
                                0x44444444,  
                                0x55555555,  
                                0x66666666,  
                                0x77777777,  
                                0x88888888,  
                                0x99999999,  
                                0xAAAAAAAA;  
  
.var tx_buf1b[BUFSIZE] = 0x12345678,  
                            0x23456789,  
                            0x3456789A,  
                            0x456789AB,  
                            0x56789ABC,  
                            0x6789ABCD,  
                            0x789ABCDE,  
                            0x89ABCDEF,  
                            0x9ABCDEF0,  
                            0xABCDEF01;  
  
/* RX Buffers */  
.var rx_buf0a[BUFSIZE];  
.var rx_buf0b[BUFSIZE];  
  
/* TX Transfer Control Blocks */  
.var tx_tcb1[4] = 0,BUFSIZE,1,tx_buf1a;  
.var tx_tcb2[4] = 0,BUFSIZE,1,tx_buf1b;  
  
/* RX Transfer Control Blocks */  
.var rx_tcb1[4] = 0,BUFSIZE,1,rx_buf0a;  
.var rx_tcb2[4] = 0,BUFSIZE,1,rx_buf0b;  
  
/* Main code section */  
.global _main;  
.SECTION/PM seg_pmco;
```

```

_main:

//      SPORT Loopback: Use SPORT0 as RX & SPORT1 as TX      //

/* initially clear SPORT control register */
r0 = 0x00000000;
dm(SPCTL0) = r0;
dm(SPCTL1) = r0;
dm(SPMCTL1) = r0;

SPORT_DMA_setup:
/* set internal loopback bit for SPORT0 & SPORT1 */
bit set ustat3 SPL;
dm(SPMCTL1) = ustat3;

/* Configure SPORT1 as a transmitter */
/* internally generating clock and frame sync */
/* CLKDIV = [fCLK(333 MHz)/8xFCLK(8.325 MHz)]-1 = 0x0004 */
/* FSDIV = [FCLK(8.325 MHz)/TFS(.26 MHz)]-1 = 31 = 0x001F */
R0 = 0x001F0004;    dm(DIV1) = R0;
ustat4 = SPEN_A|    /* Enable Channel A */
          SLEN32|    /* 32-bit word length */
          FSR|       /* Frame Sync Required */
          SPTRAN|    /* Transmit on enabled channels */
          SDEN_A|    /* Enable Channel A DMA */
          SCHEN_A|   /* Enable Channel A DMA Chaining */
          IFS|       /* Internally-generated Frame Sync */
          ICLK;      /* Internally-generated Clock */
dm(SPCTL1) = ustat4;

/* Configure SPORT0 as a receiver */
/* externally generating clock and frame sync */
r0 = 0x0;    dm(DIV0) = R0;
ustat3 = SPEN_A|    /* Enable Channel A */

```

SPORT Programming Examples

```
SLEN32|      /* 32-bit word length */
FSR|         /* Frame Sync Required */
SDEN_A|      /* Enable Channel A DMA */
SCHEN_A;     /* Enable Channel A DMA Chaining */
dm(SPCTL0) = ustat3;

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

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

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

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

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

6 SERIAL PERIPHERAL INTERFACE PORTS

The ADSP-21367/8/9 and ADSP-2137x processors are equipped with two synchronous serial peripheral interface ports that are compatible with the industry-standard serial peripheral interface (SPI). The SPI ports are routed through the digital peripheral interface pins (DPI14–1). At reset, SPI functionality is available on DPI pins 1–8. [For more information, see Chapter 4, Digital Audio/Digital Peripheral Interfaces](#). Each SPI port has its own unique set of control registers (the secondary register set is differentiated by a B in the register name as in `SPIBAUDB`). The SPI ports support communication with a variety of peripheral devices including CODECs, data converters, sample rate converters, S/PDIF or AES/EBU digital audio transmitters and receivers, LCDs, shift registers, microcontrollers, and FPGA devices with SPI emulation capabilities.

Each SPI port provides the following features and capabilities:

- A simple 4-wire interface consisting of two data pins, a device select pin, and a clock pin
- Full-duplex operation that allows the processor to transmit and receive data simultaneously on the same port
- Special data formats to accommodate little and big endian data, different word lengths, and various packing modes
- Master and slave modes as well as multimaster mode in which the processors can be connected to four other SPI devices
- Open drain outputs to avoid possible driver damage due to data contention and to support multimaster scenarios

Functional Description

- Programmable baud rates, clock polarities, and phases
- Master or slave booting from a master SPI device ([“DPI/SRU2 Connection Groups” on page 4-51](#))
- DMA capability to allow data transfers without core overhead

Functional Description

Each SPI port contains its own transmit shift (TXSR, TXSRB) and receive shift (RXSR, RXSRB) registers which are not user accessible. The TXSRx registers serially transmit data and the RXSRx registers receive data synchronously with the SPI clock signal (SPICLK). [Figure 6-1](#) shows a block diagram of the ADSP-21367/8/9 and ADSP-2137x processor's SPI interface. The data is shifted into or out of the shift registers on two separate pins: the master in slave out (MISO) pin and the master out slave in (MOSI) pin.

During data transfers, one SPI device acts as the SPI master by controlling the data flow. It does this by generating the SPICLK and asserting the SPI device select signal ($\overline{\text{SPIDS}}$). The SPI master receives data using the MISO pin and transmits using the MOSI pin. The other SPI device acts as the SPI slave by receiving new data from the master into its RXSRx register using the MOSI pin. It transmits requested data out of the TXSR register using the MISO pin.

Each SPI port contains a dedicated transmit data buffer (TXSPI, TXSPIB) and a receive data buffer (RXSPI, RXSPIB). Data to be transmitted is written to TXSPIx and then automatically transferred into the TXSR register. Once a full data word is received in the RXSR register, the data is automatically transferred into RXSPI, from which the data is read. When the port is in SPI master mode, programmable flag pins provide slave selection. Connect these pins to the $\overline{\text{SPIDS}}$ of the slave devices.

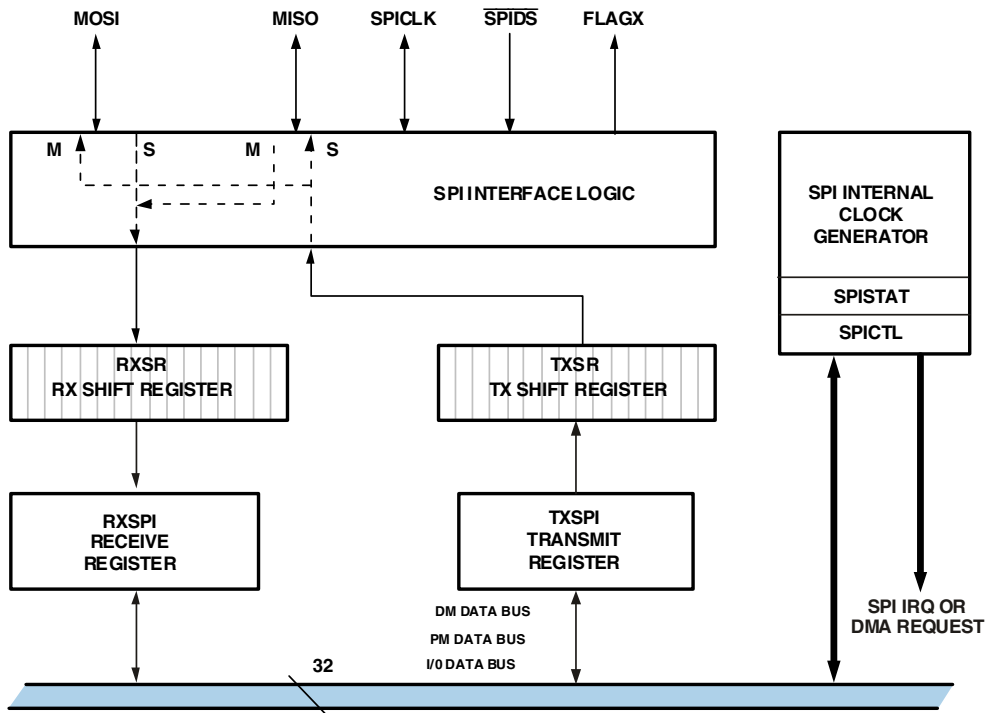


Figure 6-1. SPI Block Diagram

Different CPUs or processors can take turns being master, or one master may simultaneously shift data into multiple slaves (broadcast mode). However, only one slave may drive its output to write data back to the master at any given time. This must be enforced in broadcast mode, where several slaves can be selected to receive data from the master, but only one slave can be enabled to send data back to the master.

In a multimaster or multidevice environment where multiple processors are connected through their SPI ports, all MOSI pins are connected together, all MISO pins are connected together, as are the SPICLK pins. The master's FLAGx pins connect to each of the slave SPI devices in the system

SPI Interface Signals

through their $\overline{\text{SPID}}\text{S}$ pins. The SPI ports also provide a mechanism to disable the MISO pin for multimaster systems where the slave SHARC processor does not need to transmit any data to the master.

SPI Interface Signals

The SPI uses a 4-wire protocol to enable full-duplex serial communication. This section describes the signals used to connect the ADSP-21367/8/9 and ADSP-2137x processor's SPI ports in a system that has multiple devices. Figure 6-2 shows the master-slave connections between two devices.

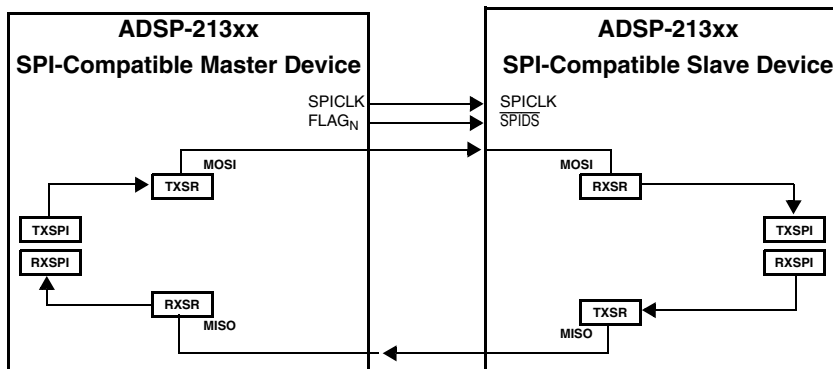


Figure 6-2. Master-Slave Interconnections

SPI Clock Signal (SPICLK)

The SPICLK signal is the serial peripheral interface clock signal. This control signal is driven by the master and regulates the flow of data bits. The master may transmit data at a variety of baud rates. One data bit is transferred for each SPICLK cycle.

The `SPICLK` signal is a gated clock that is active only during data transfers, and only for the duration of the transferred word. The number of active edges is equal to the number of bits driven on the data lines. The clock rate can be as high as one-fourth the peripheral clock rate.¹ For master devices, the clock rate is determined by the 15-bit value of the baud rate registers (`SPIBAUD`, `SPIBAUDB`). [For more information, see “SPI Baud Rate Registers \(SPIBAUD, SPIBAUDB\)” on page A-60.](#) For slave devices, the value in the `SPIBAUDx` register is ignored. When the SPI device is a master, `SPICLK` is an output signal. When the SPI is a slave, `SPICLK` is an input signal. Slave devices ignore the serial clock if its device select (`SPIDS`) signal is deasserted (HIGH).

Data is shifted in reference to `SPICLK`. The data is always shifted out on one edge of the clock (referred to as the active edge) and sampled on the opposite edge of the clock (referred to as the sampling edge). Clock polarity and clock phase relative to data are programmable through bit 11 (`CLKPL`) and bit 10 (`CPHASE`) in the `SPICLx` control registers.

SPICLK Timing

When the processor is configured as an SPI slave, the SPI master must drive an `SPICLK` signal that conforms with [Figure 6-3](#). For exact timing parameters, please refer to the appropriate data sheet.

The `SPIDS` lead time (T_1), the `SPIDS` lag time (T_2), and the sequential transfer delay time (T_3) must always be greater than or equal to one-half the `SPICLK` period. The minimum time between successive word transfers (T_4) is two `SPICLK` periods. This time period is measured from the last active edge of `SPICLK` of one word to the first active edge of `SPICLK` of the next word. This calculation is independent from the configuration of the SPI (`CPHASE`, `SPIMS`, and so on).

¹ For complete information on device clock signals and timing, see the processor-specific data sheet.

SPI Interface Signals

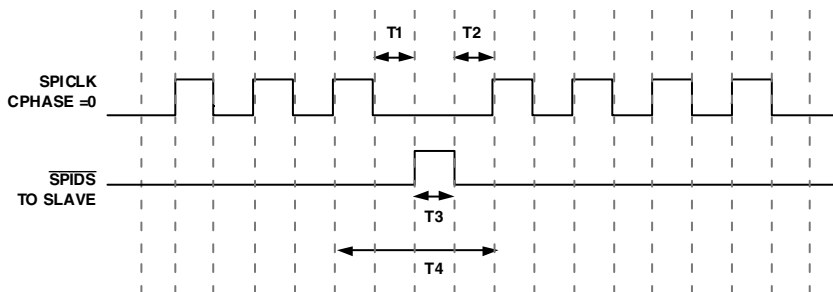


Figure 6-3. SPICLK Timing

SPI Slave Select Input ($\overline{\text{SPIDS}}$)

The $\overline{\text{SPIDS}}$ signal is the serial peripheral interface device select input signal. This active low signal is used to enable a processor that is configured as a slave device. As an input only pin, $\overline{\text{SPIDS}}$ behaves like a chip select, and is driven by the master device for the slave devices. When the processor is the SPI master in a multimaster environment, the $\overline{\text{SPIDS}}$ pin acts as an error signal. In multimaster mode, if the $\overline{\text{SPIDS}}$ input signal of a master is asserted (driven low), a multimaster error condition occurs which means that another device is also trying to be the master device. For a single-master, multiple-slave configuration, the $\overline{\text{SPIDS}}$ signal of the master device must be tied high, or the SPI port allows this signal to be disabled and used as a GPIO input.

SPI Flag Signals (SPIFLG3-0)

These signals are driven by the processor as an SPI master to the $\overline{\text{SPIDS}}$ pin of the slave. When $\text{CPHASE} = 0$, the SPI port hardware controls the device-select signal automatically (determined by the DSxEN bits in the SPIFLG register). Setting $\text{CPHASE} = 1$ requires these signals be manually controlled by the software through the SPIFLGx bits in the SPIFLG and SPIFLGB registers. The SPIFLGx bits are ignored when $\text{CPHASE} = 0$.

Master Out Slave In (MOSI)

The MOSI pin is one of the bidirectional I/O data pins. If the ADSP-21367/8/9 and ADSP-2137x processors are configured as masters, the MOSI pin becomes a data transmit (output) pin. If the processors are configured as slaves, the MOSI pin becomes a data receive (input) pin. In an SPI interconnection, the data is shifted out from the MOSI output pin of the master and shifted into the MOSI input of the slave.

Master In Slave Out (MISO)

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

[Figure 6-4](#) illustrates how the processor can be used as the slave SPI device. The 8-bit host microcontroller is the SPI master. The processor can be booted through its SPI interface to allow application code and data to be downloaded prior to runtime. When a system is comprised of multiple slaves, only one slave is allowed to transmit data back to the master at any given time.

[Figure 6-4](#) also shows an example SPI interface where a processor is the SPI master. When it uses the SPI interface, the processor can be directed to alter the conversion resources, mute the sound, modify the volume, and power down the AD1855 stereo DAC by accessing the control register of the DAC serially.

SPI General Operations

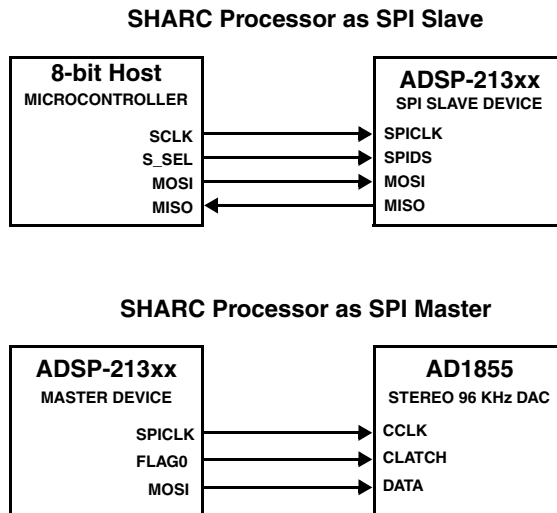


Figure 6-4. SHARC Processor as SPI Master and Slave

SPI General Operations

The SPI in the ADSP-21367/8/9 and ADSP-2137x processors can be used in a single master as well as in a multimaster environment. In both of these configurations, every MOSI pin in the SPI system is connected. Likewise, every MISO pin in the system is on a single node, and every SPICLK pin is connected. SPI transmission and reception are always enabled simultaneously, unless broadcast mode has been selected. In broadcast mode, several slaves can be configured to receive from the master, but only one of the slaves can be in transmit mode. This is done by driving the MISO line, to communicate back with the master. If the transmit or receive is not needed, MISO can be ignored and does not need to be connected. This section describes the clock signals, SPI operation as a master and as a slave, and error generation conditions.

SPI Enable

For slaves, the slave-select input acts like a reset for the internal SPI logic. For this reason, the $\overline{\text{SPIDS}}$ line must be error free. The SPIEN signal can also be used as a software reset of the internal SPI logic. An exception to this is the W1C-type (write 1-to-clear) bits in the SPISTAT_x (SPI status) registers which remain set if they are already set. For a list of write W1C bits, see [Table A-11 on page A-57](#).



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

Open Drain Mode (OPD)

In a multimaster or multislave SPI system, the data output pins (MOSI and MISO) can be configured to behave as open drain drivers to prevent possible damage to pin drivers due to contention. An external pull-up resistor is required on both the MOSI and MISO pins when this option is selected.

When the OPD bit is set and the SPI ports are configured as masters, the MOSI pin is three-stated when the data driven out on MOSI is logic high. The MOSI pin is not three-stated when the driven data is logic low. A zero is driven on the MOSI pin in this case. Similarly, when OPD is set and the SPI ports are configured as slaves, the MISO pin is three-stated if the data driven out on MISO is logic high.

Master Mode Operation

When the SPI is configured as a master, configure the SPI port and start transfers using the following steps:

1. Before enabling the SPI port, programs should specify which of the slave-select signals to use, setting one or more of the required SPI flag select bits ($DSxEN$) in the $SPIFLGx$ registers.
2. Write to the $SPICTLx$ and $SPIBAUDx$ registers, enabling the device as a master and configuring the SPI system by specifying the appropriate word length, transfer format, baud rate, and other necessary information.

When $CPHASE$ is set to 0, the slave-select signals are automatically controlled by the SPI port. When $CPHASE = 1$, the slave selects are controlled by the core and the user software has to control the pins through the $SPIFLGx$ bits. If $CPHASE = 1$ (user-controlled, slave-select signals), activate the desired slaves by clearing one or more of the SPI flag bits ($SPIFLGx$) in the $SPIFLGx$ registers.

3. Initiate the SPI transfer. The trigger mechanism for starting the transfer is dependant upon the $TIMOD$ bits in the $SPICTLx$ registers. See [“Master Transfer Preparation” on page 6-18](#) for details.

The SPI generates the programmed clock pulses on $SPICLK$. The data is shifted out of $MOSI$ and shifted in from $MISO$ simultaneously. Before starting to shift, the $TXSR$ register is loaded with the contents of the $TXSPIx$ registers. At the end of the transfer, the contents of the $RXSR$ register are loaded into the $RXSPIx$ registers.

With each new transfer initiate command, the SPI continues to send and receive words, according to the SPI transfer mode ($TIMOD$ bit in the $SPICTLx$ registers). See [“Master Transfer Preparation” on page 6-18](#) for more details.

In master mode, if the transmit buffer remains empty, or the receive buffer remains full, the device operates according to the states of the `SENDZ` and `GM` bits in the `SPICTLx` registers.

- If `SENDZ = 1` and the transmit buffer is empty, the device repeatedly transmits zeros on the `MOSI` pin. One word is transmitted for each new transfer initiate command.
- If `SENDZ = 0` and the transmit buffer is empty, the device repeatedly transmits the last word it transmitted before the transmit buffer became empty.
- If `GM = 1` and the receive buffer is full, the device continues to receive new data from the `MISO` pin, overwriting the older data in the `RXSPI` buffer.
- If `GM = 0` and the receive buffer is full, the incoming data is discarded, and the `RXSPI` register is not updated.

Slave Mode Operation

When a device is enabled as a slave (and DMA mode is not selected), the start of a transfer is triggered by a transition of the `SPIDS` select signal to the active state (`LOW`) or by the first active edge of the clock (`SPICLK`), depending on the state of `CPHASE`.

The following steps illustrate SPI operation in slave mode:

1. Write to the `SPICTLx` registers making sure that the slave's configuration matches the current SPI master.
2. To prepare for the data transfer, write the data to be transmitted into the `TXSPIx` registers.

SPI General Operations

Once the $\overline{\text{SPIDS}}$ signal's falling edge is detected, the slave starts sending and receiving data on active SPICLK edges.

The reception or transmission continues until $\overline{\text{SPIDS}}$ is released. The slave device continues to receive or transmit with each new active SPICLK clock edge while the $\overline{\text{SPIDS}}$ signal is active.

In slave mode, if the transmit buffer remains empty, or the receive buffer remains full, the devices operate according to the states of the SENDZ and GM bits in the SPICLX registers.

- If $\text{SENDZ} = 1$ and the transmit buffer is empty, the device repeatedly transmits zeros on the MISO pin.
- If $\text{SENDZ} = 0$ and the transmit buffer is empty, the device repeatedly transmits the last word it transmitted before the transmit buffer became empty.
- If $\text{GM} = 1$ and the receive buffer is full, the device continues to receive new data from the MOSI pin, overwriting the older data in the RXSPI buffer.
- If $\text{GM} = 0$ and the receive buffer is full, the incoming data is discarded, and the RXSPIx registers are not updated.

Multimaster Operation

A multimaster mode is implemented in the processor to allow an SPI system to transfer mastership from one SPI device to another. In a multidevice SPI configuration, several SPI ports are connected and any one (but only one) can become a master at any given time.

If a processor is a slave and wishes to become the SPI master, it asserts the $\overline{\text{SPIDS}}$ pin for the processor that is currently master and then drives the SPICLK signal. Once the master device receives the $\overline{\text{SPIDS}}$ signal, it is

immediately reconfigured as a slave. In order to safely transition from one master to the other, the SPI port uses open drain outputs for the data pin drivers. This helps to avoid possible damage from data contention.

More information on this topic is described in [“Mode Fault Error \(MME\)” on page 6-35](#).

SPI Data Transfer Operations

The following sections provide information on the two methods the processors use to transfer data—through the core or through DMA.

SPI Operation Using the Core

For core-driven SPI transfers, the software must read from or write to the `RXSPIx` and `TXSPIx` registers respectively to control the transfer. It is important to check the buffer status before reading from or writing to these registers because the core *does not* hang when it attempts to read from an empty buffer or write to a full buffer. When the core writes to a full buffer, the data that is in that buffer is overwritten and the SPI begins transmitting the new data. Invalid data is obtained when the core reads from an empty buffer.

For a master, when the transmit buffer becomes empty, or the receive buffer becomes full, the SPI device stalls the SPI clock until all the data from the receive buffer is read or it detects that the transmit buffer contains a piece of data, depending upon the `TIMOD` setting in the `SPICTL` register.

- When a master is configured with `TIMOD = 01` and the transmit buffer becomes empty, the SPI device stalls the SPI clock until a piece of data is written to the transmit buffer.

SPI Data Transfer Operations

- When a master is configured with `TIMOD = 00` and the receive buffer becomes full the SPI device stalls the SPI clock until all of the data is read from the receive buffer.

SPI Operation Using DMA

Each SPI has a single DMA channel associated with it that can be configured to support either an SPI transmit or a receive, but not both simultaneously. In addition to the `TXSPIx` and `RXSPIx` data buffers, there is a four-word deep DMA FIFO that the SPI ports use to improve the data throughput.

The SPI ports support both master and slave mode DMA. The following sections describe master and slave mode DMA operations, DMA chaining, switching between transmit and receive DMA operations, and processing DMA interrupt errors. The following are guidelines to follow when performing DMA transfers over the SPI.

- Do not write to the `TXSPIx` registers during an active SPI transmit DMA operation because DMA data will be overwritten.
- Similarly, do not read from the `RXSPIx` registers during active SPI DMA receive operations because DMA data will be overwritten.
- Writes to the `TXSPIx` registers during an active SPI receive DMA operation are permitted. The `RXS` register is cleared when the `RXSPIx` registers are read.
- Reads from the `RXSPIx` registers are allowed at any time during transmit DMA.
- Interrupts are generated based on DMA events and are configured in the `SPIDMACx` registers.

In order for a transmit DMA operation to begin, the transmit buffer must initially be empty (`TXS = 0`). While this is normally the case, this means that the `TXSPIx` registers should not be used for any purpose other than

SPI transfers. For example, the `TXSPIx` registers should not be used as a scratch register for temporary data storage. Writing to the `TXSPIx` registers sets the `TXS` bit.

When the SPI DMA engine is configured for transmitting:

1. The receive interface cannot generate an interrupt, but the status can be polled.
2. The four-deep FIFO is not available in the receive path.
3. The `RXSPIx` register is used to receive data.

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

1. The transmit interface cannot generate an interrupt, but the status can be polled.
2. The four-deep FIFO is not available in the transmit path.
3. The `TXSPIx` register is used to transmit data.

Master Mode DMA Operation

To configure the SPI port for master mode DMA transfers:

1. Specify which `FLAG` pins to use as the slave-select signals by setting one or more of the `DSxEN` bits (bits 3–0) in the SPI flag (`SPIFLGx`) registers.
2. Enable the device as a master and configure the SPI system by selecting the appropriate word length, transfer format, baud rate, and so on in the `SPIBAUDx` and `SPICTLx` registers. The `TIMOD` field (bits 1–0) in the `SPICTLx` registers is configured to select transmit or receive with DMA mode (`TIMOD` = 10).
3. Activate the desired slaves by clearing one or more of the SPI flag bits (`SPIFLGx`) of `SPIFLGx` registers if `CPHASE` = 1.

SPI Data Transfer Operations

4. For a single DMA, define the parameters of the DMA transfer by writing to the `IISPIx`, `IMSPIx`, and `CSPIx` registers. For DMA chaining, also write the chain pointer address to the `CPSPIx` registers after the other DMA registers. [For more information, see “Setting Up and Starting Chained DMA over the SPI” on page 2-42.](#)
5. Write to the SPI DMA configuration registers, (`SPIDMACx`), to specify the DMA direction (`SPIRCV`, bit 1) and to enable the SPI DMA engine (`SPIDEN`, bit 0). If DMA chaining is desired, set (= 1) the `SPICHEN` bit (bit 4) in the `SPIDMACx` registers.



To avoid data corruption, enable the SPI port before enabling DMA.

If flags are used as slave selects, programs should activate the flags by clearing the flag after the `SPICTLx` and the `SPIBAUDx` registers are configured, but before enabling the DMA. When `CPHASE = 0`, and a program is using DMA, the flags are automatically activated by the SPI ports.


When enabled as a master, the DMA engine transmits or receives data as follows:

1. If the SPI system is configured for transmitting, the DMA engine reads data from memory into the SPI DMA FIFO. Data from the DMA FIFO is loaded into the `TXSPIx` registers and then into the transmit shift register. This initiates the transfer on the SPI port.
2. If configured to receive, data from the `RXSPIx` registers is automatically loaded into the SPI DMA FIFO and the DMA engine reads data from the SPI DMA FIFO and writes to memory. Finally, the SPI initiates the receive transfer.

3. The SPI generates the `SPICLK` signal (as specified by `CPHASE`, `SPIBAUD`, and other bit settings) and the data is shifted out of `MOSI` and in from `MISO` simultaneously.
4. The SPI continues sending or receiving words until the SPI DMA word count register decrements to 0.

If the DMA engine is unable to keep up with the transmit stream during a transmit operation because the IOP requires the IOD (I/O data) bus to service another DMA channel (or for another reason), the `SPICLK` signal stalls until data is written into the `TXSPI` register. All aspects of SPI receive operation should be ignored. The data in the `RXSPI` register is not intended to be used, and the `RXS` (bits 28–27 and 31–30 in the `SPCTLx` registers) and `SPISTAT` bits (26 and 29) should be ignored. The `ROVF` overrun condition cannot generate an error interrupt in this situation.

If the DMA engine cannot keep up with the receive data stream during receive operations, then `SPICLK` stalls until data is read from `RXSPI`. While performing a receive DMA, the processor core assumes the transmit buffer is empty. If `SENDZ` = 1, the device repeatedly transmits zeros. If `SENDZ` = 0, it repeatedly transmits the contents of the `TXSPI` register. The `TUNF` under-run condition cannot generate an error interrupt in this situation.

 For receive DMA in master mode, the `SPICLK` signal stops only when the FIFO and the `RXSPI` buffer is full (even if the DMA count is zero). Therefore, the `SPICLK` signal runs for an additional five word transfers filling junk data in the FIFO and the `RXSPIx` buffers. This data must be cleared before a new DMA is initiated.

A master SPI DMA sequence may involve back-to-back transmission and/or reception of multiple chained DMA transfers. The SPI controller supports such a sequence with minimal processor core interaction.

Master Transfer Preparation

When the processor is enabled as a master, the initiation of a transfer is defined by the two bit fields (bits 1–0) of `TIMOD` in the `SPICTLx` registers. Based on these two bits and the status of the interface, a new transfer is started upon either a read of the `RXSPIx` registers or a write to the `TXSPIx` registers. This is summarized in [Table 6-1](#).

Table 6-1. Transfer Initiation

TIMOD	Function	Transfer Initiated Upon	Action, Interrupt
00	Transmit and Receive	Initiate new single word transfer upon read of <code>RXSPI</code> and previous transfer completed.	The SPI interrupt is latched in every core clock cycle in which the <code>RXSPI</code> buffer has a word in it. Emptying the <code>RXSPI</code> buffer or disabling the SPI port at the same time (<code>SPIEN</code> = 0) stops the interrupt latch.
01	Transmit and Receive	Initiate new single word transfer upon write to <code>TXSPI</code> and previous transfer completed.	The SPI interrupt is latched in every core clock cycle in which the <code>TXSPI</code> buffer is empty. Writing to the <code>TXSPI</code> buffer or disabling the SPI port at the same time (<code>SPIEN</code> = 0) stops the interrupt latch.
10	Transmit or Receive with DMA	Initiate new multiword transfer upon write to DMA enable bit. Individual word transfers begin with either a DMA write to <code>TXSPI</code> or a DMA read of <code>RXSPI</code> depending on the direction of the transfer as specified by the <code>SPIRCV</code> bit.	If chaining is disabled, the SPI interrupt is latched in the cycle when the DMA count decrements from 1 to 0. If chaining is enabled, interrupt function is based on the <code>CPI</code> bit in the <code>CP</code> register. If <code>CPI</code> = 0, the SPI interrupt is latched at the end of the DMA sequence. If <code>CPI</code> = 1, then the SPI interrupt is latched after each DMA in the sequence. For more information, see “DMA Transfer Direction” on page 2-24.
11	Reserved		

Slave Mode DMA Operation

A slave mode DMA transfer occurs when the SPI port is enabled and configured in slave mode, and DMA is enabled. When the `SPIDSS` signal transitions to the low state or when the first active edge of `SPICLK` is detected, it triggers the start of a transfer.

To configure for slave mode DMA:

1. Write to the `SPICTLx` registers to make the mode of the serial link the same as the mode that is set up in the SPI master. Configure the `TIMOD` field to select transmit or receive DMA mode (`TIMOD = 10`).
2. Define DMA receive (or transmit) transfer parameters by writing to the `IISPIx`, `IMSPIx`, and `CSPIx` registers. For DMA chaining, write the chain pointer address to the `CPSPIx` registers.
3. Write to the `SPIDMACx` registers to enable the SPI DMA engine and configure the following:
 - A receive access (`SPIRCV = 1`) or
 - A transmit access (`SPIRCV = 0`)

If DMA chaining is desired, set the `SPICHEN` bit in the `SPIDMACx` registers.



Enable the SPI port before enabling DMA to avoid data corruption.

Slave Transfer Preparation

When enabled as a slave, the device prepares for a new transfer according to the function and actions described in [Table 6-1](#).

The following steps illustrate the SPI receive or transmit DMA sequence in an SPI slave in response to a master command:

1. Once the slave-select input is active, the processor starts receiving and transmitting data on active `SPICLK` edges. The data for one channel (TX or RX) is automatically transferred from/to memory by the IOP. The function of the other channel is dependant on the `GM` and `SENDZ` bits in the `SPICTL` register.
2. Reception or transmission continues until the SPI DMA word count register transitions from 1 to 0.
3. A number of conditions can occur while the processor is configured for the slave mode:
 - If the DMA engine cannot keep up with the receive data stream during receive operations, the receive buffer operates according to the state of the `GM` bit in the `SPICTLx` registers.
 - If `GM = 0` and the DMA buffer is full, the incoming data is discarded and the `RXSPIx` register is not updated. While performing a receive DMA, the transmit buffer is assumed to be empty. If `SENDZ = 1`, the device repeatedly transmits zeros on the `MISO` pin. If `SENDZ = 0`, it repeatedly transmits the contents of the `TXSPIx` registers.
 - If `GM = 1` and the DMA buffer is full, the device continues to receive new data from the `MOSI` pin, overwriting the older data in the DMA buffer.
 - If the DMA engine cannot keep up with the transmit data stream during a transmit operation because another DMA engine has been granted the bus (or for another reason), the

transmit port operates according to the state of the `SENDZ` bit in the `SPICTLx` registers.

If `SENDZ = 1` and the DMA buffer is empty, the device repeatedly transmits zeros on the `MISO` pin. If `SENDZ = 0` and the DMA buffer is empty, it repeatedly transmits the last word it transmitted before the DMA buffer became empty. All aspects of SPI receive operation should be ignored. The data in the `RXSPIx` registers is not intended to be used, and the `RXS` and `ROVF` bits should be ignored. The `ROVF` overrun condition cannot generate an error interrupt in this situation.



While a DMA transfer is occurring on one channel (`TX` or `RX`), the core (based on the `RXS` and `TXS` status bits) can transfer data in the other direction.

Changing SPI Configuration

Programs should take the following precautions when changing SPI configurations.

- The SPI configuration must not be changed during a data transfer.
- Change the clock polarity only when no slaves are selected.
- Change the SPI configuration only when `SPIEN = 0`. For example, if operating as a master in a multislave system, and there are slaves that require different data or clock formats, then the master SPI should be disabled, reconfigured, and then re-enabled.

However, when an SPI communication link consists of 1) a single master and a single slave, 2) `CPHASE = 1`, and 3) the slave's slave select input is tied low, then the program can change the SPI

SPI Data Transfer Operations

configuration. In this case, the slave is always selected. Data corruption can be avoided by enabling the slave only after configuring both the master and slave devices.

When performing transmit operations with the SPI port, disabling the SPI port prematurely can cause data corruption and/or faulty transmission. Before the program disables the SPI port in order to reconfigure it, the status bits should be polled to ensure that all valid data has been completely transferred. For core-driven transfers, data moves from the `TXSPI` buffer into a shift register. The following bits should be checked before disabling the SPI port:

1. Wait for the `TXSPIx` buffers to empty into the shift register. This is done when the `TXS` bit (bit 3) of the `SPISTATx` registers becomes zero.
2. Wait for the SPI shift registers to finish shifting out data. This is done when the `SPIF` bit (bit 0 of `SPISTATx` registers) becomes one.
3. Disable the SPI ports by setting the `SPIEN` bit (bit 0) in the `SPICTLx` registers to zero.

When performing transmit DMA transfers, data moves through a four-deep SPI DMA FIFO, then into the `TXSPIx` buffers, and finally into the shift register. DMA interrupts are latched when the I/O processor moves the last word from memory to the peripheral. For the SPI, this means that the SPI “DMA complete” interrupt is latched when there are still six words yet to be fully transmitted (four in the FIFO, one in the `TXSPIx` buffers, and one being shifted out of the shift register). To disable the SPI port after a DMA transmit operation, use the following steps:

1. Wait for DMA FIFO to empty. This is done when the `SPIStx` bits (bits 13–12 in the `SPIIDMACx` registers) become zero.
2. Wait for the `TXSPIx` registers to empty. This is done when the `TXS` bit (bit 3) in the `SPISTATx` registers becomes zero.

3. Wait for the SPI shift register to finish transferring the last word. This is done when the `SPIF` bit (bit 0 of the `SPISTATx` registers), becomes one.
4. Disable the SPI ports by setting the `SPIEN` bit (bit 0 of the `SPICTLx` registers), to zero.

Switching From Transmit To Receive DMA

The following sequence details the steps for switching from transmit to receive DMA.

With disabling the SPI:

1. Write 0x00 to the `SPICTLx` registers to disable SPI. Disabling the SPI also clears the `RXSPIx/TXSPIx` registers and the buffer status.
2. Disable DMA by writing 0x00 to the `SPIDMACx` registers.
3. Clear all errors by writing to the `WIC`-type bits in the `SPISTATx` registers. This ensures that no interrupts occur due to errors from a previous DMA operation.
4. Reconfigure the `SPICTLx` registers and enable the SPI ports.
5. Configure DMA by writing to the DMA parameter registers and enable DMA.

Without disabling the SPI:

1. Clear `RXSPIx/TXSPIx` and the buffer status without disabling SPI. This can be done by ORing 0xC0000 with the present value in the `SPICTLx` registers. For example, programs can use the `RXFLSH` and `TXFLSH` bits to clear `TXSPIx/RXSPIx` and the buffer status.
2. Disable DMA by writing 0x00 to the `SPIDMAC` register.

SPI Data Transfer Operations

3. Clear all errors by writing to the W1C-type bits in the `SPISTATx` registers. This ensures that no interrupts occur due to errors from a previous DMA operation.
4. Reconfigure the `SPICTL` register to remove the clear condition on the `TXSPI`/`RXSPI` registers.
5. Configure DMA by writing to the DMA parameter registers and enable DMA.

Switching From Receive to Transmit DMA

Use the following sequence to switch from receive to transmit DMA. Note that `TXSPIx` and `RXSPIx` are registers but they may not contain any bits, only address information.

With disabling of the SPI:

1. Write 0x00 to the `SPICTLx` registers to disable the SPI. Disabling the SPI also clears the contents of the `RXSPIx`/`TXSPIx` registers and the buffer status.
2. Disable DMA and clear the DMA FIFO by writing 0x80 to the `SPIDMACx` registers. This ensures that any data from a previous DMA operation is cleared because the `SPICLK` signal runs for five more word transfers even after the DMA count falls to zero in the receive DMA.
3. Clear all errors by writing to the `SPISTATx` registers. This ensures that no interrupts occur due to errors from a previous DMA operation.
4. Reconfigure the `SPICTLx` registers and enable the SPI.
5. Configure DMA by writing to the DMA parameter registers and the `SPIDMACx` register.

Without disabling the SPI:

1. Clear the `RXSPIx/TXSPIx` registers and the buffer status without disabling the SPI by ORing `0xc0000` with the present value in the `SPICTLx` registers. Use the `RXFLSH` (bit 19) and `TXFLSH` (bit 18) bits in the `SPICTLx` registers to clear the `RXSPIx/TXSPIx` registers and the buffer status.
2. Disable DMA and clear the FIFO by writing `0x80` to the `SPIDMACx` registers. This ensures that any data from a previous DMA operation is cleared because the `SPICLK` signal runs for five more word transfers even after the DMA count is zero in receive DMA.
3. Clear all errors by writing to the `W1C`-type bits in the `SPISTATx` registers. This ensures that no interrupts occur due to errors from a previous DMA operation.
4. Reconfigure the `SPICTLx` registers to remove the clear condition on the `TXSPIx/RXSPIx` registers.
5. Configure DMA by writing to the DMA parameter registers (described in [Table 2-6 on page 2-29](#)) and the `SPIDMACx` registers using the `SPIDEN` bit (bit 0).

DMA Error Interrupts

The `SPIUNF` and `SPIOVF` bits of the `SPIDMACx` registers indicate transmission errors during a DMA operation in slave mode. When one of the bits is set, an SPI interrupt occurs. The following sequence details the steps to respond to this interrupt.

SPI Data Transfer Operations

With disabling the SPI:

1. Disable the SPI port by writing 0x00 to the SPICTLx registers.
2. Disable DMA and clear the FIFO by writing 0x80 to the SPIDMACx registers. This ensures that any data from a previous DMA operation is cleared before configuring a new DMA operation.
3. Clear all errors by writing to the W1C-type bits (see [Table A-11 on page A-57](#)) in the SPISTATx registers. This ensures that the error bits SPIOVF and SPIUNF (in the SPIDMACx registers) are cleared when a new DMA is configured.
4. Reconfigure the SPICTLx registers and enable SPI using the SPIEN bit.
5. Configure DMA by writing to the DMA parameter registers and the SPIDMACx registers.

Without disabling the SPI:

1. Disable DMA and clear the FIFO by writing 0x80 to the SPIDMAC register. This ensures that any data from a previous DMA operation is cleared before configuring a new DMA operation.
2. Clear the RXSPIx/TXSPIx registers and the buffer status without disabling the SPI. This can be done by ORing 0xC0000 with the present value in the SPICTLx registers. Use the RXFLSH and TXFLSH bits to clear the RXSPIx/TXSPIx registers and the buffer status.
3. Clear all errors by writing to the W1C-type bits in the SPISTAT register. This ensures that error bits SPIOVF and SPIUNF in the SPIDMACx registers are cleared when a new DMA is configured.
4. Reconfigure SPICTL to remove the clear condition of the RXSPI/TXSPI register bits.

5. Configure DMA by writing to the DMA parameter registers and the SPIDMACx register.

DMA Chaining

For information about chaining, refer to [“Setting Up and Starting Chained DMA over the SPI” on page 2-42](#).

SPI Transfer Formats

The SPI ports support four different combinations of serial clock phases and polarity. The application code can select any of these combinations using the CLKPL and CPHASE bits in the SPICTL register.

[Figure 6-5 on page 6-28](#) shows the transfer format when CPHASE = 0 and [Figure 6-6 on page 6-29](#) shows the transfer format when CPHASE = 1. Each diagram shows two waveforms for SPICLK—one for CLKPL = 0 and the other for CLKPL = 1. The diagrams may be interpreted as master or slave timing diagrams since the SPICLK, MISO, and MOSI pins are directly connected between the master and the slave. The MISO signal is the output from the slave (slave transmission), and the MOSI signal is the output from the master (master transmission).

The SPICLK signal is generated by the master, and the $\overline{\text{SPIDS}}$ signal represents the slave device select input to the SPI slave from the SPI master. The diagrams represent 8-bit transfers (WL = 0) with MSB first (MSBF = 1). Any combination of the WL and MSBF bits of the SPICTL register is allowed. For example, a 16-bit transfer with the LSB first is one possible configuration.

The clock polarity and the clock phase should be identical for the master and slave devices involved in the communication link. The transfer format from the master may be changed between transfers to adjust to various requirements of a slave device.

SPI Transfer Formats

When $CPHASE = 0$, the slave select line, \overline{SPIDS} , must be inactive (HIGH) between each word in the transfer. When $CPHASE = 1$, \overline{SPIDS} may either remain active (LOW) between successive transfers or be inactive (HIGH).

Figure 6-5 shows the SPI transfer protocol for $CPHASE = 0$. Note that $SPICLK$ starts toggling in the middle of the data transfer, $WL = 0$, and $MSBF = 1$.

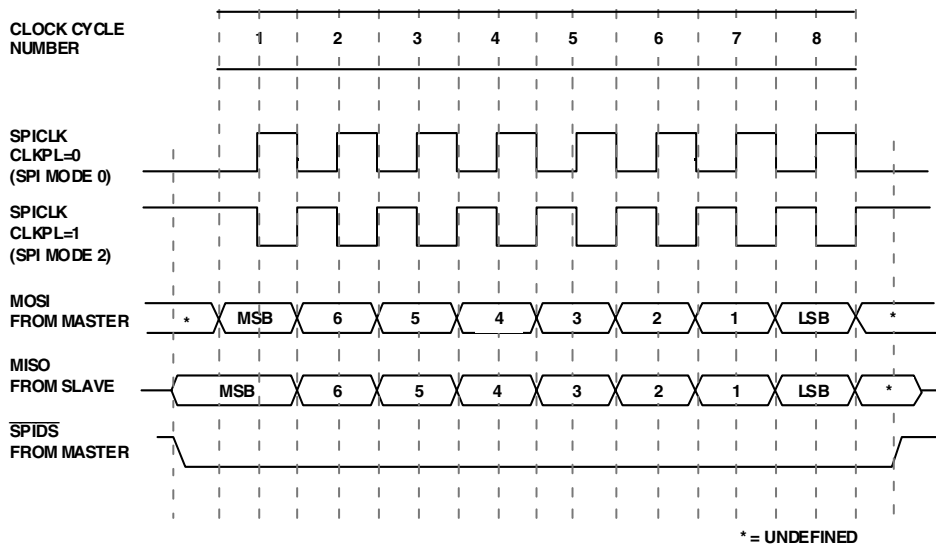


Figure 6-5. SPI Transfer Protocol for $CPHASE = 0$

Figure 6-6 shows the SPI transfer protocol for $CPHASE = 1$. Note that $SPICLK$ starts toggling at the beginning of the data transfer, $WL = 0$, and $MSBF = 1$.

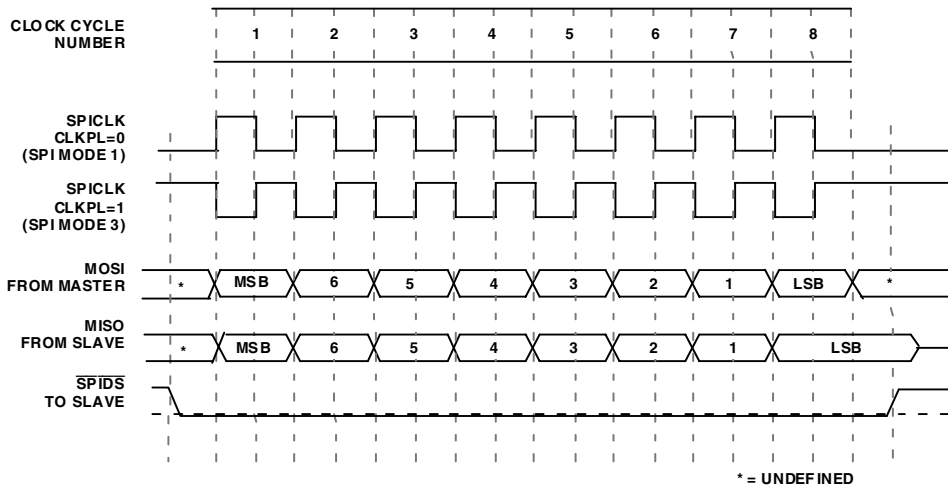


Figure 6-6. SPI Transfer Protocol for CPHASE = 1

Beginning and Ending an SPI Transfer

An SPI transfer's defined start and end depend on: whether the device is configured as a master or a slave, whether CPHASE mode is selected, and which transfer initiation mode (TIMOD) is selected. For a master SPI with CPHASE = 0, a transfer starts when either the TXSPI register is written or the RXSPI register is read, depending on the TIMOD selection. At the start of the transfer, the enabled slave-select outputs are driven active (LOW). However, the SPICLK starts toggling after a delay equal to one-half the SPICLK period. For a slave with CPHASE = 0, the transfer starts as soon as the SPIDS input transitions to low.

For CPHASE = 1, a transfer starts with the first active edge of SPICLK for both slave and master devices. For a master device, a transfer is considered complete after it sends and simultaneously receives the last data bit. A transfer for a slave device is complete after the last sampling edge of SPICLK.

SPI Transfer Formats

The `RXS` bit defines when the receive buffer can be read. The `TXS` bit defines when the transmit buffer can be filled. The end of a single word transfer occurs when the `RXS` bit is set. This indicates that a new word has been received and latched into the receive buffer, `RXSPI`. The `RXS` bit is set shortly after the last sampling edge of `SPICLK`. The latency is typically a few core clock cycles and is independent of `CPHASE`, `TIMOD`, and the baud rate. If configured to generate an interrupt when `RXSPI` is full (`TIMOD = 00`), the interrupt becomes active one core clock cycle after `RXS` is set. When not relying on this interrupt, the end of a transfer can be detected by polling the `RXS` bit.

To maintain software compatibility with other SPI devices, the SPI transfer finished bit (`SPIF`) is also available for polling. This bit may have slightly different behavior from that of other commercially available devices. For a slave device, `SPIF` is set at the same time as `RXS`. For a master device, `SPIF` is set one-half of the `SPICLK` period after the last `SPICLK` edge, regardless of `CPHASE` or `CLKPL`.

The baud rate determines when the `SPIF` bit is set. In general, when (`SPIBAUD < 4`) `SPIF` is set *after* `RXS`. The `SPIF` bit is set before the `RXS` bit is set, and consequently before new data has been latched into the `RXSPI` buffer. Therefore, for `SPIBAUD = 2` or `SPIBAUD = 3`, the processor must wait for the `RXS` bit to be set (after `SPIF` is set) before reading the `RXSPI` buffer. For larger `SPIBAUD` settings (`SPIBAUD > 4`), `RXS` is set before `SPIF` is set.

SPI Word Lengths

The processor's SPI port can transmit and receive the word widths described in the following sections.

8-Bit Word Lengths

Programs can use 8-bit word lengths when transmitting or receiving. When transmitting, the SPI port sends out only the lower eight bits of the word written to the SPI buffer.

For example, if the processor executes the following instructions:

```
r0 = 0x12345678
dm(TXSPI) = r0;
```

the SPI port transmits 0x78.

When receiving, the SPI port packs the 8-bit word to the lower 32 bits of the RXSPI buffer while the upper bits in the registers are zeros.

For example, if an SPI host sends the processor the 32-bit word 0x12345678, the processor receives the following words:

```
0x00000078 //first word
0x00000056 //second word
0x00000034 //third word
0x00000012 //fourth word
```

This code works only if the MSBF bit is zero in both the transmitter and receiver. If MSBF = 1 in the transmitter and receiver, the received words follow the order 0x12, 0x34, 0x56, 0x78.

16-Bit Word Lengths

Programs can use 16-bit word lengths when transmitting or receiving. When transmitting, the SPI port sends out only the lower 16 bits of the word written to the SPI buffer.

For example, if the processor executes the following instructions:

```
r0 = 0x12345678
dm(TXSPI) = r0;
```

the SPI port transmits 0x5678.

When receiving, the SPI port packs the 16-bit word to the lower 32 bits of the RXSPI buffer while the upper bits in the register are zeros.

For example, if an SPI host sends the processor the 32-bit word 0x12345678, the processor receives the following words:

```
0x00005678 //first word
0x00001234 //second word
```

32-Bit Word Lengths

Programs can use 32-bit word lengths when transmitting or receiving. No packing of the RXSPI or TXSPI registers is necessary as the entire 32-bit register is used for the data word.

Packing

In order to communicate with 8-bit SPI devices and store 8-bit words in internal memory, a packed transfer feature is built into the SPI port. Packing is enabled through the `PACKEN` bit in the `SPICTL` register. The SPI port unpacks data when it transmits and packs data when it receives. When packing is enabled, two 8-bit words are packed into one 32-bit word.

When the SPI port is transmitting, two eight-bit words are unpacked from one 32-bit word. When receiving, words are packed into one 32-bit word from two eight-bit words.

An example of unpacking the data before transmitting:

The value `0xXXLMXXJK` (where `XX` is any random value and `JK` and `LM` are the data words to be transmitted out of the SPI port) is written to the `TXSPI` register. The processor transmits `0xJK` first and then transmits `0xLM`.

An example of packing on the received data (`SGN` is sign extend data where 1 = sign extend and 0 = no sign extend):

The receiver packs the two words received, `0xJK` and then `0xLM`, into a 32-bit word. They appear in the `RXSPI` register as:

`0x00LM00JK` => if `SGN` is configured to 0 or `L, J < 7`

`0xFFLMFFJK` => if `SGN` is configured to 1 and `L, J > 7`

SPI Interrupts

The SPI ports can generate interrupts in five different situations. During core-driven transfers, an SPI interrupt is triggered:

1. When the `TXSPI` buffer has the capacity to accept another word from the core.
2. When the `RXSPI` buffer contains a valid word to be retrieved by the core.

The `TIMOD` (transfer initiation and interrupt) register determines whether the interrupt is based on the `TXSPI` or `RXSPI` buffer status. For more information, refer to the `TIMOD` bit descriptions in the `SPICTL` register in [Table A-10 on page A-54](#).

SPI Interrupts

During IOP-driven transfers (DMA), an SPI interrupt is triggered:

3. At the completion of a single DMA transfer,
4. At the completion of a number of DMA sequences (if DMA chaining is enabled),
5. When a DMA error has occurred.

Again, the `SPIDMAC` register must be initialized properly to enable DMA interrupts.

Each of these five interrupts are serviced using the interrupt associated with the module being used. The primary SPI uses the `SPIHI` (programmable interrupt 1 by default, 0x90030) interrupt and the secondary SPI uses the `SPIII` (programmable interrupt 18 by default, 0x90074) interrupt. Whenever an SPI interrupt occurs (regardless of the cause), the `SPIII` or `SPIHI` interrupts are latched. To service the primary SPI port, unmask (set = 1) the `SPIHI` bit (bit 12) in the `IMASK` register. To service the secondary SPI port, unmask (set = 1) the `SPIIIMSK` bit (bit 19) in the `LIRPTL` register. For a list of these bits, see [Table 2-3 on page 2-9](#) and “[LIRPTL Register Bit Descriptions](#)” on page B-8.

To globally enable interrupts, set (= 1) the `IRPTEN` bit in the `MODE1` register. When using DMA transfers, programs must also specify whether to generate interrupts based on transfer or error status. For DMA transfer status based interrupts, set the `INTEN` bit in the `SPIDMAC` register; otherwise, set the `INTERR` bit to trigger the interrupt if one of the error conditions occurs during the transmission like multimaster error (MME), transmit buffer underflow (`TUNF` – only if `SPIRCV` = 0), or receive buffer overflow (`ROVF` – only if `SPIRCV` = 1). During core-driven transfers, the `TUNF` and `ROVF` error conditions do not generate interrupts.

- See [“Interrupt Registers” on page B-6](#) for `IRPTL` and `LIRPTL` register bit descriptions.
- See [“SPI DMA Configuration Registers \(SPIDMAC, SPID-MACB\)” on page A-62](#) for `SPIDMAC` register bit descriptions.

Error Signals and Flags

This section describes the error signals and flags that determine the cause of transmission errors for an SPI port. The bits `MME`, `TUNF`, and `ROVF` are set in the `SPISTAT` register when a transmission error occurs. Corresponding bits (`SPIMME`, `SPIUNF`, and `SPIOVF`) in the `SPIDMAC` register are set when an error occurs during a DMA transfer. These `W1C` bits generate an SPI interrupt when any one of them are set.

- See [“SPI Port Status \(SPISTAT, SPISTATB\) Registers” on page A-56](#) for more information about the `SPISTAT` register bits.
- See [“SPI DMA Configuration Registers \(SPIDMAC, SPID-MACB\)” on page A-62](#) for more information about the `SPIDMAC` register bits.

Mode Fault Error (MME)

The `MME` bit is set in the `SPISTAT` register when the `SPIDS` input pin of a device that is enabled as a master is driven low by some other device in the system. This occurs in multimaster systems when another device is also trying to be the master.

Error Signals and Flags

The SPI ports are able to respond appropriately to this situation. To enable this feature, set the `ISSEN` bit in the `SPICTL` register. As soon as this error is detected, the following actions are taken:

1. The `SPIMS` control bit in `SPICTL` is cleared, configuring the SPI interface as a slave.
2. The `SPIEN` control bit in `SPICTL` is cleared, disabling the SPI system.
3. The `MME` status bit in `SPISTAT` is set.
4. An SPI interrupt is generated.

These four conditions persist until the `MME` bit is cleared by a write 1-to-clear (W1C-type) software operation. Until the `MME` bit is cleared, the SPI cannot be re-enabled, even as a slave. Hardware prevents the program from setting either `SPIEN` or `SPIMS` while `MME` is set.

When `MME` is cleared, the interrupt is deactivated. Before attempting to re-enable the SPI as a master, the state of the `SPIDS` input pin should be checked to ensure that it is high; otherwise, once `SPIEN` and `SPIMS` are set, another mode-fault error condition occurs immediately. The state of the input pin is reflected in the input slave-select status bit (bit 7) in the `SPI-FLG` register.



As a result of `SPIEN` and `SPIMS` being cleared, the SPI data and clock pin drivers (`MOSI`, `MISO`, and `SPICLK`) are disabled. However, the slave-select output pins revert to control by the processor flag I/O module registers. This may cause contention on the slave-select lines if these lines are still being driven by the processor. In order to ensure that the slave-select output drivers are disabled once a `MME` error occurs, the program must configure these pins as inputs by setting (= 1) the flag output select bits, `FLAG3-00`, in the `FLAGS` register prior to configuring the SPI port. See the `FLAGS` value register description in the *ADSP-2136x SHARC Processor Programming Reference* “Registers” Appendix.

Transmission Error Bit (TUNF)

The TUNF bit is set in the SPISTAT register when all of the conditions of transmission are met and there is no new data in the TXSPI buffer (TXSPI is empty). In this case, the transmission contents depend on the state of the SENDZ bit in the SPICTL register. The TUNF bit is cleared by a W1C-type software operation.

Reception Error Bit (ROVF)

The ROVF flag is set in the SPISTAT register when a new transfer has completed before the previous data could be read from the RXSPI register. This bit indicates that a new word was received while the receive buffer was full. The ROVF flag is cleared by a W1C-type software operation. The state of the GM bit in the SPICTL register determines whether the RXSPI register is updated with the newly received data or whether that new data is discarded.

Transmit Collision Error Bit (TXCOL)

The TXCOL flag is set in the SPISTAT register when a write to the TXSPI register coincides with the load of the shift register. The write to TXSPI can be initiated by the core or by DMA. This bit indicates that corrupt data may have been loaded into the shift register and transmitted. In this case, the data in TXSPI may not match what was transmitted. This error can easily be avoided by proper software control. The TXCOL bit is cleared by a W1C-type software operation.



This bit is never set when the SPI is configured as a slave with CPHASE = 0. The collision may occur, but it cannot be detected.

Programming Notes

The following sections provide information to help the programmer use the SPI in an ADSP-21367/8/9 and ADSP-2137x processor system.

Routing SPI Signals Using The DPI

All signals of both SPI ports are routed to the DPI pins using the SRU2. Special considerations must be made when routing the signals, especially with regards to using the correct pin enables that work with the SPI mode being used. In addition, open drain mode also has special requirements for DPI routing. See [“Configuring the SPI” on page 4-72](#) for more information.

Programming Examples

The following three programming examples are for the ADSP-21369 processor. The example shown in [Listing 6-1](#) transmits a buffer of data from the SPI port in master mode using DMA. In this example, the I/O processor (IOP) automatically moves data from internal memory to the SPI’s four-deep DMA FIFO.

The second example, shown in [Listing 6-2 on page 6-41](#), also transmits a buffer, but the transfer is core-driven using interrupts. In this example, only the SPI’s one-deep transmit buffer (TXSPI) is serviced by the core and the four-deep DMA FIFO is not used. The core supplies the SPI port with data in a short loop which causes the core to hang at each write to the transmit buffer until the SPI is ready for new data.

The third example, shown in [Listing 6-3 on page 6-43](#), receives multiple buffers using DMA chaining. DMA chaining on the ADSP-21367/8/9 processor’s SPI is initialized differently than on other SHARC processors, as described in [Chapter 2, I/O Processor](#).

Listing 6-1. SPI Master Mode Core-Driven Transmit

```
/* SPI Control Registers */
#define SPICTL (0x1000)
#define SPIFLG (0x1001)
#define SPIBAUD (0x1005)
#define TXSPI (0x1003)

/*SPICTL bits*/
#define TIMOD1 (0x0001) /* Use TX buffer for transfers */
#define DMISO (0x0020) /* Disable MISO pin */
#define WL32 (0x0100) /* SPI Word Length = 32 */
#define SPIMS (0x1000) /* SPI Master if 1, Slave if 0 */
#define SPIEN (0x4000) /* SPI port Enable */

/*SPIFLG bits */
#define DS0EN (0x0001) /* use FLG0 as SPI device-select*/

/* Default Buffer Length */
#define BUFSIZE 10

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

Programming Examples

```
/* Main code section */
.global _main;
.SECTION/PM seg_pmco;
_main:
/* Init SPI MASTER TX */
r0=0;
dm(SPICTL) = r0;
dm(SPIFLG) = r0;

/* Set up DAG registers */
i4 = tx_buf;
m4 = 1;
ustat3 = DMISO|      /* Disable MISO on transfers */
        WL32|        /* 32-bit words */
        SPIMS|        /* Master mode (internal SPICLK) */
        SPIEN|        /* Enable SPI port */
        TIMOD1;      /* Initialize SPI port to begin
                        transmitting when DMA is enabled */
dm(SPICTL) = ustat3;

/* set the SPI baud rate to CCLK/8*64 (650.39KHz @ 333MHz)*/
ustat3 = 0x80;
dm(SPIBAUD) = ustat3;
/* Set up loop to transmit data */
lcntr = LENGTH(tx_buf), do txloop until lce;
/* Retrieve data using DAG1 and send TX through SPI */
r0 = dm(i4,m4);
dm(TXSPI) = r0;
ustat3=dm(SPISTAT);
bit tst ustat3 SPITXS;
if tf jump (pc,-2);
txloop: nop;

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

Listing 6-2. SPI Master Mode DMA-Driven Transmit

```

/* SPI Control registers */
#define SPICTL (0x1000) /* SPI Control register */
#define SPIFLG (0x1001) /* SPI Flag register */
#define SPIBAUD (0x1005) /* SPI baud setup register */

/* SPI DMA registers */
#define IISPI (0x1080) /* Internal DMA address */
#define IMSPI (0x1081) /* Internal DMA access modifier */
#define CSPI (0x1082) /* Number of words to transfers */
#define CPSPI (0x1083) /* Points to next DMA parameters */
#define SPIDMAC (0x1084) /* SPI DMA control register */

/*SPICTL bits */
#define TIMOD2 (0x0002) /* Use DMA for transfers */
#define DMISO (0x0020) /* Disable MISO pin */
#define WL32 (0x0100) /* SPI Word Length = 32 */
#define SPIMS (0x1000) /* SPI Master if 1, Slave if 0 */
#define SPIEN (0x4000) /* SPI port Enable */

/*SPIFLG bits */
#define DS0EN (0x0001) /* use FLG0 as SPI device-select */

/*SPIDMAC bits */
#define SPIDEN (0x0001) /* enable DMA on the SPI port */

/* Default buffer size */
#define BUFSIZE 0x100
/*=====*/
/* Source data to be transmitted through SPI DMA */
.section/dm seg_dmda;
.var src_buf[BUFSIZE] = "source.dat";

```

Programming Examples

```
/* Application code */
.global _main;
.segment/pm seg_pmco;
_main:

/* Init SPI MASTER TX DMA */
r0 = 0;
dm(SPICTL) = r0;
dm(SPIFLG) = r0;
dm(SPIDMAC) = r0;

r0 = DS0EN;
dm(SPIFLG) = r0; /*use flag0 as spi device select */

ustat3 = src_buf; dm(IISPI) = ustat3; /* point to 'src_buf'*/
ustat3 = LENGTH(src_buf); dm(CSPI) = ustat3; /* count = 256 */
ustat3 = 1; dm(IMSPI) = ustat3; /* step size = 1 */

/* set the SPI baud rate to CCLK/8*64 (650.39KHz @ 333MHz)*/
ustat3 = 0x80;
dm(SPIBAUD) = ustat3;

ustat3 = DMISO| /* Disable MISO on transfers */
        WL32| /* 32-bit words */
        SPIMS| /* Master mode (internal SPICLK) */
        SPIEN| /* Enable SPI port */
        TIMOD2; /* Initialize SPI port to begin
                  transmitting when DMA is enabled */
dm(SPICTL) = ustat3;

ustat3 = SPIDEN; dm(SPIDMAC) = ustat3; /* begin DMA */
/*=====*/
_main.end: jump (pc,0);
```

Listing 6-3. SPI DMA Chaining Example

```
/* SPI Control registers */
#define SPICTL (0x1000) /* SPI Control register */
#define SPIFLG (0x1001) /* SPI Flag register */
#define SPIBAUD (0x1005) /* SPI baud setup register */

/* SPI DMA registers */
#define IISPI (0x1080) /* Internal DMA address */
#define IMSPI (0x1081) /* Internal DMA access modifier */
#define CSPI (0x1082) /* Number of words to transfers */
#define CPSPI (0x1083) /* Points to next DMA parameters */
#define SPIDMAC (0x1084) /* SPI DMA control register */

/*SPIFLG bits */
#define DS0EN (0x0001) /* enable SPI device select 0 */
#define SPIFLG0 (0x0100) /* manually set SPIFLG0 state */
#define SPIFLG1 (0x0200) /* manually set SPIFLG1 state */
#define SPIFLG2 (0x0400) /* manually set SPIFLG2 state */
#define SPIFLG3 (0x0800) /* manually set SPIFLG3 state */

/*SPIDMAC bits */
#define SPIDEN (0x0001) /* enable DMA on the SPI port */
#define SPIRCV (0x0002) /* set to have DMA receive */
#define SPICHEN (0x0010) /* set to enable DMA chaining */

/*SPICTL bits */
#define TIMOD2 (0x0002) /* Use DMA for transfers */
#define SENDZ (0x0004) /* when TXSPI empty, MOSI sends 0 */
#define WL32 (0x0100) /* SPI Word Length = 32 */
#define SPIMS (0x1000) /* SPI Master if 1, Slave if 0 */
#define SPIEN (0x4000) /* SPI port Enable */
#define CLKPL (0x0800) /* if 1, rising edge samples data */
```

Programming Examples

```
#define CPHASE (0x0400) /* if 1, data's sampled on second
                        (middle) edge of SPICLK cycle*/
/*=====*/
.section/dm seg_dmda;

/* Destinations for incoming data */
.var dest_bufC[8];
.var dest_bufB[8];
.var dest_bufA[8];

/* Transfer Control Blocks (TCB's) */
.var first_tcb[] =
    (0x7FFFF&second_tcb + 3), /* for CPSPi (next tcb) */
    LENGTH(dest_bufB),        /* for CSPI (next count) */
    1,                        /* for IMSPI (next modify) */
    dest_bufB;                /* for IISPI (next index) */

.var second_tcb[] = 0,        /* null CPSPi ends chain */
    LENGTH(dest_bufC),        /* count for final DMA */
    1,                        /* IM for final DMA */
    dest_bufC;                /* II for final DMA */
/* NOTE: Chain Pointer registers must point to the LAST
    location in the TCB, "tcb + 3". */

/*Main code section */
.global _main;
.section/pm seg_pmco;
_main:
/* clear SPI settings */
r0 = 0;
dm(SPICTL) = r0;
dm(SPIFLG) = r0;
dm(SPIDMAC) = r0;
```



```
/* setup first DMA in chain */
ustat3 = 8;          dm(CSPI) = ustat3; /* count = 8 words */
ustat3 = 1;          dm(IMSPI) = ustat3; /* step size = 1 */
ustat3 = dest_bufA; dm(IISPI) = ustat3; /* point to dest_bufA */

/* set the SPI baud rate to CCLK/8*64 (650.39KHz @ 333MHz)*/
ustat3 = 0x80;
dm(SPIBAUD) = ustat3;

/* configure processor's SPI slave-select signals */
ustat3 = DS0EN|      /*enable SPI slave device select zero */
SPIFLG3|SPIFLG2|SPIFLG1; /* Set SPIFLG0 low to */
dm(SPIFLG) = ustat3; /*select SPI slave on FLAG0 pin */

/* configure SPI port to power-on settings */
ustat3 = CPHASE| /* sample MISO on second edge of SPICLK */
CLKPL| /* sampling edge of SPICLK is rising */
WL32| /* 32-bit words */
SPIMS| /* Master mode (internal SPICLK) */
SPIEN| /* Enable SPI port */
SENDZ| /* when TXSPI empty, MOSI sends zeros */
TIMOD2; /* Start SPICLK when DMA is enabled */
dm(SPICTL) = ustat3;

/*configure SPI for chained receive DMA operation */
ustat3 = SPIRCV| /* DMA direction = receive */
SPICHEN| /* enable DMA chaining */
SPIDEN; /* enabling DMA initiates the transfer */
dm(SPIDMAC) = ustat3;

/* 1st DMA starts when a valid address is written to CPSPI */
ustat3 = (0x7FFF&(first_tcb+3));
dm(CPSPI) = ustat3; /* point to tcb_A */
_main.end: jump(pc,0);
```


7 INPUT DATA PORT

The signal routing unit (SRU) provides paths among both on-chip and off-chip peripherals. To make this feature effective in a real-world system, a low overhead method of making data from various serial and parallel formats and routing them back to the main core memory is needed. The input data port (IDP) provides this mechanism for a large number of asynchronous channels.

This chapter describes how data is routed into the core's memory space. [Figure 7-1](#) provides a graphical overview of the input data port architecture. Notice that each channel is independent and each contains a separate clock and frame sync input.

Channels 0 through 7 can accept serial data in audio format. Channel 0 can also be configured to accept parallel data. The parallel input bypasses the serial-to-parallel converter and latches up to 20 bits per clock cycle.

The parallel data is acquired through the parallel data acquisition port (PDAP) which provides a means of moving high bandwidth data to the core's memory space. The data may be sent to memory as one 32-bit word per input clock cycle or packed together (for up to four clock cycles worth of data). [Figure 7-2 on page 7-3](#) illustrates the data flow for IDP channel 0, where either the PDAP or serial input can be selected via `IDP_PDAP_EN` (bit 31 of the `IDP_PP_CTL` register). At the falling edge of `IDP_PDAP_EN`, the FIFO is cleared. Data transfer from the channels to the FIFO happens on a fixed priority with channel 0 having the highest priority and channel 7 the lowest.

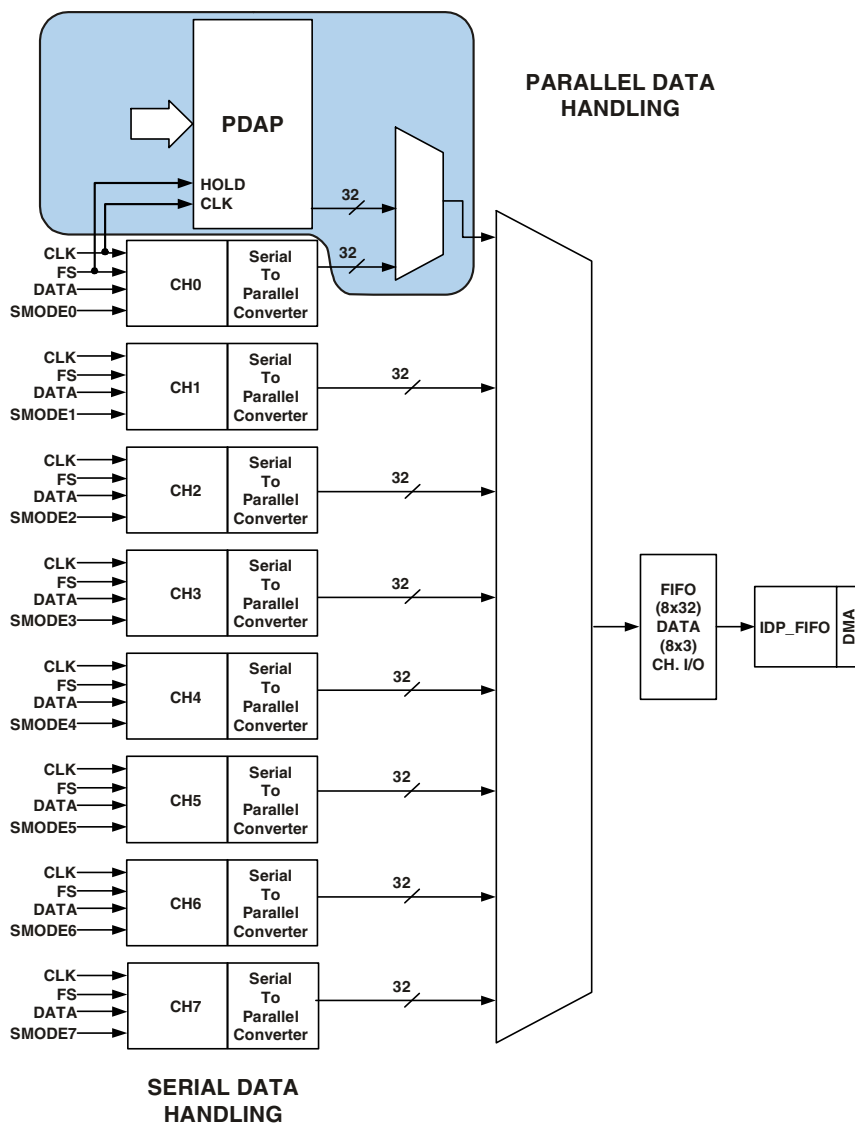


Figure 7-1. Input Data Port

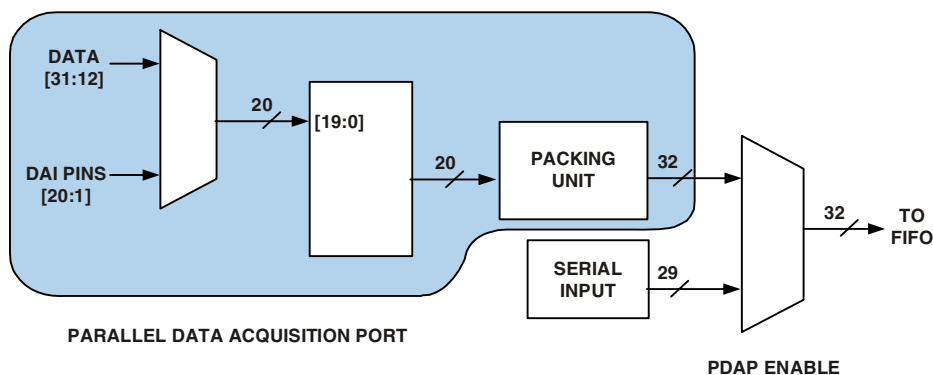


Figure 7-2. Detail of IDP Channel 0

The IDP's DMA engine implements DMA for all eight channels. Each of the eight channels has a set of DMA parameter registers for directing the data to memory location.

The following sections describe each of the input data port functions.

Serial Inputs

The IDP provides up to eight serial input channels—each with its own clock, frame sync, and data inputs. The eight channels are automatically multiplexed into a single 32-bit by eight-deep FIFO. Data is always formatted as a 64-bit frame and divided into two 32-bit words. The serial protocol is designed to receive audio channels in I²S, left-justified sample pair, or right-justified mode. One frame sync cycle indicates one 64-bit left/right pair, but data is sent to the FIFO as 32-bit words (that is, one-half of a frame at a time). The processor supports 24- and 32-bit I²S, 24- and 32-bit left-justified, and 24-, 20-, 18- and 16-bit right-justified formats.

Serial Inputs

An audio signal that is normally 24 bits wide is contained within the 32-bit word. Four more bits are available for status and formatting data (compliant with the IEC 90958, S/PDIF, and AES3 standards). An additional bit identifies the left/right one-half of the frame. If the data is not in IEC standard format, the serial data can be any data word up to 28 bits wide. Regardless of mode, bit 3 always specifies whether the data is received in the first half (left channel) or the second half (right channel) of the same frame, as shown in [Figure 7-3](#). The remaining three bits are used to encode one of the eight channels being passed through the FIFO to the core. The FIFO output may feed eight DMA channels, where the appropriate DMA channel (corresponding to the channel number) is selected automatically.

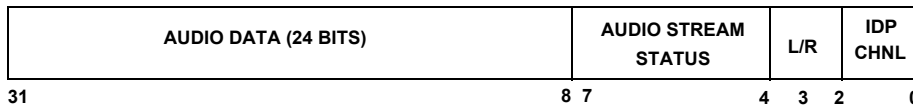


Figure 7-3. Word Format

Note that each input channel has its own clock and frame sync input, so unused IDP channels do not produce data and therefore have no impact on FIFO throughput. The clock and frame sync of any unused input should be assigned to low to avoid unintentional acquisition. The input data port supports a maximum clock speed of 41.6 MHz.

The framing format is selected by using the `IDP_SMODEx` bits (three bits per channel) in the `IDP_CTL0` register. The bits (31–8) of the `IDP_CTL0` register control the input format modes for each of the eight channels. The eight groups of three bits indicate the mode of the serial input for each of the eight IDP channels, as shown in [Table 7-1](#).

[Figure 7-4](#) through [Figure 7-9](#) show the FIFO data packing for the different serial modes.

Table 7-1. Serial Modes

Bit Field Values IDP_SMODEx	Mode
000	Left-justified sample pair
001	I ² S
010	Left-justified 32 bits. This is a single data and not a left/right channel pair. It can be read as 32-bit data.
011	I ² S-32 bit. This is a single data and not a left/right channel pair. It can be read as 32-bit data.
100	Right-justified sample pair 24 bits
101	Right-justified sample pair 20 bits
110	Right-justified sample pair 18 bits
111	Right-justified sample pair 16 bits

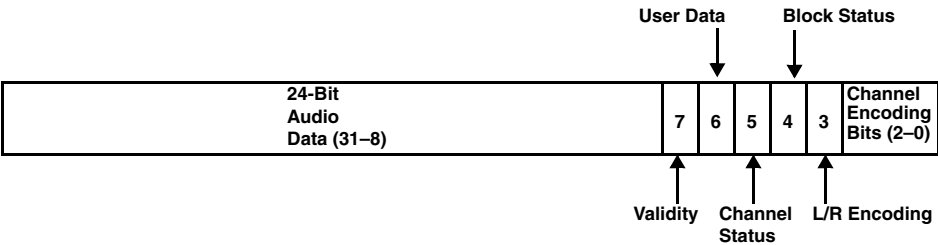


Figure 7-4. FIFO Data Packing for I²S and Left-Justified



Figure 7-5. FIFO Data Packing for Right-Justified

Serial Inputs



Figure 7-6. FIFO Data Packing for Right-Justified (20-Bit Data)

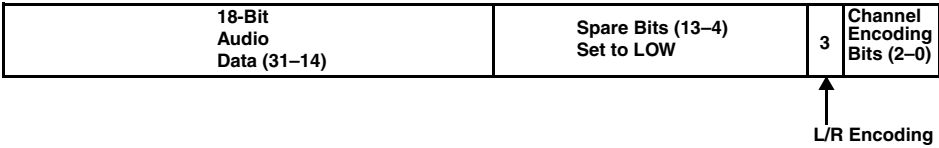


Figure 7-7. FIFO Data Packing for Right-Justified (18-Bit Data)

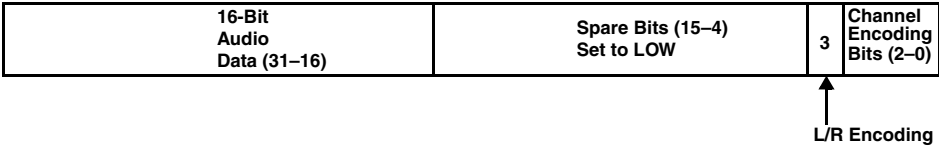


Figure 7-8. FIFO Data Packing for Right-Justified (16-Bit Data)

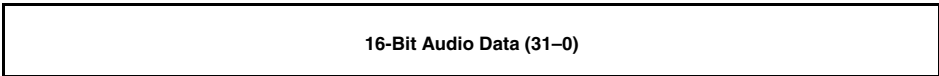


Figure 7-9. FIFO Data Packing for I²S and Left-Justified (32-Bit Data)

The polarity of left/right encoding is independent of the serial mode frame sync polarity selected in `IDP_SMODE` for that channel ([Table 7-1](#)). Note that I²S mode uses a low frame sync (left/right) signal to dictate the first (left) channel, and left-justified sample pair mode uses a HIGH frame sync (left/right) signal to dictate the first (left) channel of each frame. In either mode, the left channel has bit 3 set (= 1) and the right channel has bit 3 cleared (= 0).

Figure 7-10 shows the relationship between frame sync, serial clock, and left-justified sample pair data.

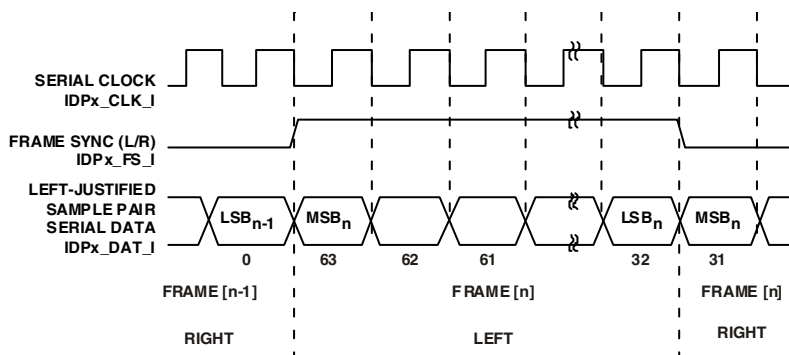


Figure 7-10. Timing in Left-justified Sample Pair Mode

Figure 7-11 shows the relationship between frame sync, serial clock, and I²S data.

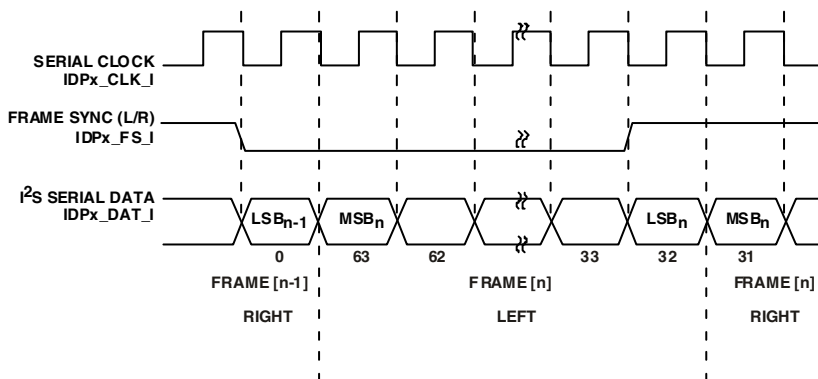


Figure 7-11. Timing in I²S Mode

Parallel Data Acquisition Port (PDAP)

The input to channel 0 of the IDP is multiplexed, and may be used either in the serial mode, described in [“Serial Inputs” on page 7-3](#), or in a direct parallel input mode. Serial or parallel input is selected by setting IDP_PDAP_EN bit 31 in the IDP_PP_CTL register. When used in parallel mode, the clock input for channel 0 is used to latch parallel subwords. Multiple latched parallel subword samples may be packed into 32-bit words for efficiency. The frame sync input is used to hold off latching of the next sample (that is, ignore the clock edges). The data then flows through the FIFO and is transferred by a dedicated DMA channel into the core’s memory as with any IDP channel. As shown in [Figure 7-12](#), the PDAP can accept input words up to 20 bits wide, or can accept input words that are packed as densely as 4 input words up to 8 bits wide.

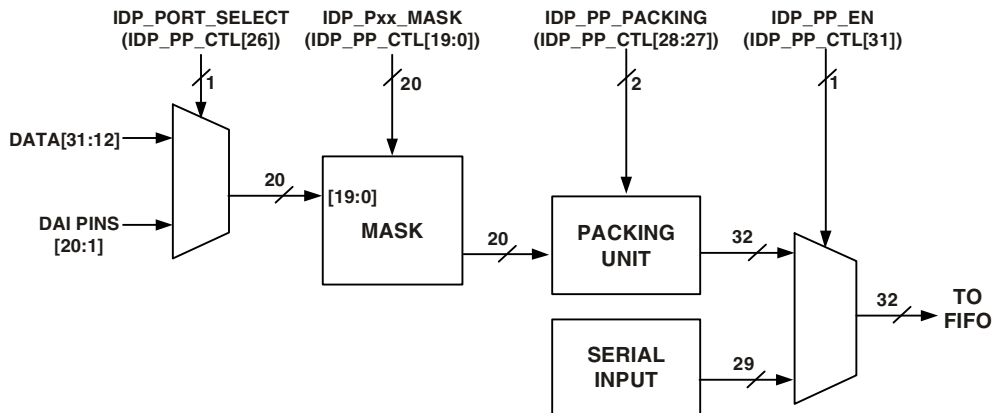


Figure 7-12. Parallel Data Acquisition Port (PDAP) Functions

The IDP_PP_CTL register also provides a reset bit that zeros any data that is waiting in the packing unit to be latched into the FIFO. When asserted, the IDP_PDAP_RESET bit (bit 30 in the IDP_PP_CTL register) causes the reset circuit to strobe, then automatically clear itself. Therefore, this bit always returns a value of zero when read. The IDP_PORT_SELECT bit

(bit 26 in the IDP_PP_CTL register) selects between the two sets of pins that may be used as the parallel input port. When IDP_PORT_SELECT is set (= 1), the data bits are read from DATA31-12 and the control signals come from DATA11-8. When IDP_PORT_SELECT is cleared (= 0), the data bits are read from DAI_P20-1. When IDP_PORT_SELECT is set to 1, the PDAP can be operated through data pins alone (data and controls can be completely routed through the DATA pins).

Masking

The IDP_PP_CTL register provides 20 mask bits that allow the input from any of the 20 pins to be ignored. The mask is specified by setting the IDP_PXX_PDAPMASK bits (bits 19-0 of the IDP_PP_CTL register) for the 20 parallel input signals. For each of the parallel inputs, a bit is set (= 1) to indicate the bit is unmasked and therefore its data can be passed on to read, or a bit is masked (= 0) so its data is not read. After this masking process, data gets passed along to the packing unit.

Packing Unit

The parallel data acquisition port (PDAP) packing unit receives masked parallel subwords from the 20 parallel input signals and packs them into a 32-bit word. The IDP_PDAP_PACKING bit field (bits 28-27 of the IDP_PP_CTL register), indicates how data is to be packed. Data can be packed in any of four modes. Selection of packing mode is made based on the application.

Packing Mode 11

Mode 11 provides for 20 bits coming into the packing unit and 32 bits going out to the FIFO in a single cycle. On every clock edge, 20 bits of data are moved and placed in a 32-bit register, left-aligned. That is, bit 19 maps to bit 31. The lower bits, 11-0, are always set to zero, as shown in [Figure 7-13](#).

Parallel Data Acquisition Port (PDAP)

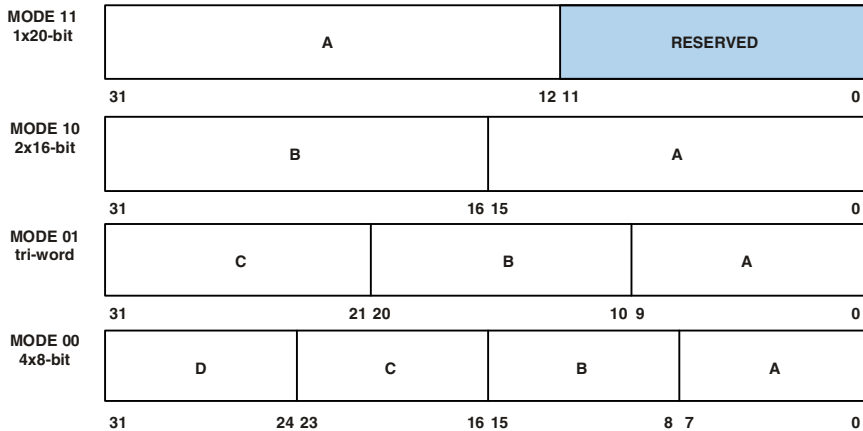


Figure 7-13. Packing Modes in PDAP

This mode sends one 32-bit word to FIFO for each input clock cycle—the DMA transfer rate matches the PDAP input clock rate.

Packing Mode 10

On the first clock edge (cycle A), the packing unit latches parallel data up to 16 bits wide (bits 19–4 of the parallel input) and places it in bits 15–0 (the lower half of the word), then waits for the second clock edge (cycle B). On the second clock edge (cycle B), the packing unit takes the same set of inputs and places the word into bits 31–16 (the upper half of the word).

This mode sends one packed 32-bit word to FIFO for every two input clock cycles—the DMA transfer rate is one-half the PDAP input clock rate.

Packing Mode 01

Mode 01 packs three acquired samples together. Since the resulting 32-bit word is not divisible by three, up to 10 bits are acquired on the first clock edge and up to 11 bits are acquired on each of the second and third clock edges:

- On clock edge 1, bits 19–10 are moved to bits 9–0 (10 bits)
- On clock edge 2, bits 19–9 are moved to bits 20–10 (11 bits)
- On clock edge 3, bits 19–9 are moved to bits 31–21 (11 bits)

This mode sends one packed 32-bit word to FIFO for every three input clock cycles—the DMA transfer rate is one-third the PDAP input clock rate.

Packing Mode 00

Mode 00 moves data in four cycles. Each input word can be up to 8 bits wide.

- On clock edge 1, bits 19–12 are moved to bits 7–0
- On clock edge 2, bits 19–12 are moved to bits 15–8
- On clock edge 3, bits 19–12 are moved to bits 23–16
- On clock edge 4, bits 19–12 are moved to bits 31–24

This mode sends one packed 32-bit word to FIFO for every four input clock cycles—the DMA transfer rate is one-quarter the PDAP input clock rate.

Clocking Edge Selection

Notice that in all four packing modes described, data is read on a clock edge, but the specific edge used (rising or falling) is not indicated. Clock edge selection is configured using the `IDP_PDAP_CLKEDGE` bit (bit 29 of the `IDP_PP_CTL` register). Setting this bit (= 1) causes the data to latch on the falling edge. Clearing this bit (= 0) causes data to latch on the rising edge (default).

Hold Input

A synchronous clock enable signal can be passed from any DAI pin to the PDAP packing unit. This signal is called `PDAP_HOLD`.



The `PDAP_HOLD` signal is actually the same physical internal signal as the frame sync for IDP channel 0. Its functionality is determined by the PDAP enable bit (`IDP_PDAP_EN`).

When the `PDAP_HOLD` signal is HIGH, all latching clock edges are ignored and no new data is read from the input pins. The packing unit operates as normal, but it pauses and waits for the `PDAP_HOLD` signal to be deasserted and waits for the correct number of distinct input samples before passing the packed data to the FIFO.

[Figure 7-14](#) shows the affect of the hold input (B) for four 8-bit words in packing mode 00, and [Figure 7-15](#) shows the affect of the hold input (B) for two 16-bit words in packing mode 10.

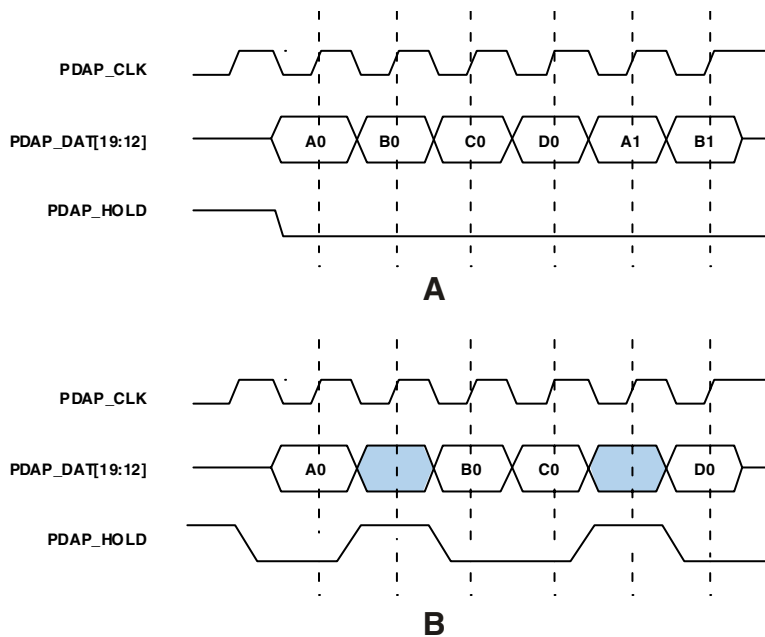


Figure 7-14. Hold Timing for Four 8-Bit Words to 32 Bits (Mode 00)

Parallel Data Acquisition Port (PDAP)

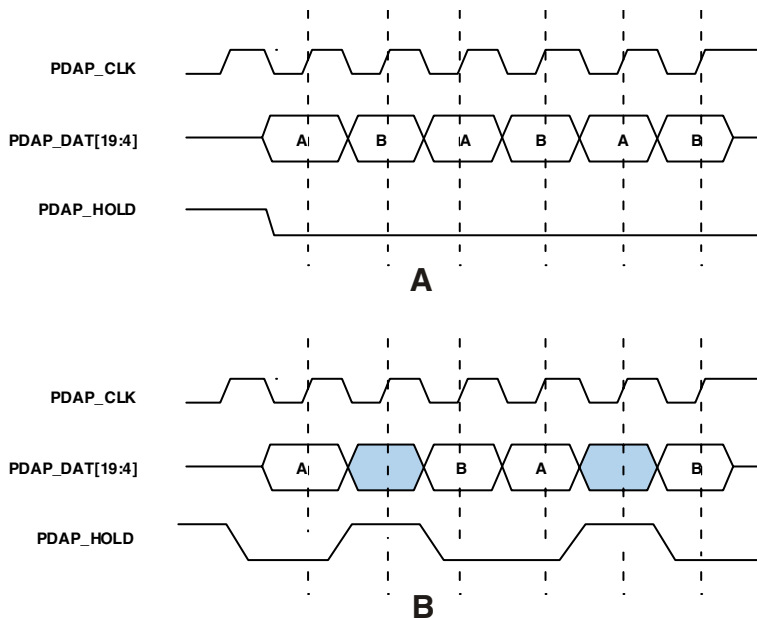


Figure 7-15. Hold Timing for Two 16-Bit Words to 32 Bits (Mode 10)

As shown in [Figure 7-15](#), PDAP_DATA and PDAP_HOLD are driven by the inactive edges of the clock (falling edge in the above figures) and these signals are sampled by the active edge of the clock (rising edge in the above figures).

PDAP Strobe

Whenever the PDAP packing unit receives the number of subwords corresponding to its select mode, it asserts the PDAP output strobe signal. This signal can be routed through the SRU using the MISC unit to any of the DAI pins. See [“DAI/SRU1 Connection Groups” on page 4-18](#) for more information.

FIFO Control and Status

Several bits can be used to control and monitor FIFO operations:

- **IDP Enable.** The `IDP_ENABLE` bit (bit 7 of the `IDP_CTL0` register) enables the IDP. This is a global control bit. This bit and the corresponding IDP channel enable bit (`IDP_ENx`) in the `IDP_CTL1` register must be set for data from a channel to get into the FIFO.
- **IDP Buffer Hang Disable.** The `IDP_BHD` bit (bit 4 in the `IDP_CTL0` register) determines whether or not the core hangs on reads when the FIFO is empty.
- **Number of Samples in FIFO.** The `IDP_FIFOSZ` bits (bits 31–28 in the `DAI_STAT` register) monitor the number of valid data words in the FIFO.
- **FIFO Overflow Status.** The `SRU_OVFx` bits in the `DAI_STAT` register monitor the overflow error conditions in the FIFO for each of the channels.
- **FIFO Overflow Clear.** The `IDP_CLR0VR` bit (bit 6 of the `IDP_CTL0` register) clears an indicated FIFO overflow error.

To enable the IDP, two separate bits in two different registers must be set. The first is the `IDP_ENABLE` bit in the `IDP_CTL0` register and the second is the specific channel enable bit which is located in the `IDP_CTL1` register. When these bits are set (= 1), the IDP is enabled. When these bits are cleared (= 0), the IDP is disabled, and data cannot come to the `IDP_FIFO` register from the IDP channels. When the `IDP_ENABLE` bit transitions from 1 to 0, all data in the IDP FIFO is cleared. Writing a 1 to bit 31 of the `IDP_CTL1` register also clears the FIFO. This is a write-only bit and always returns a zero on reads.

The `IDP_BHD` bit is used for buffer hang disable control. When there is no data in the FIFO, reading the `IDP_FIFO` register causes the core to hang. This condition continues until the FIFO contains valid data. Setting the

FIFO to Memory Data Transfer

IDP_BHD bit (= 1) prevents the core from hanging on reads from an empty IDP_FIFO register. Clearing this bit (= 0) causes the core to hang under the conditions described previously.

The IDP_FIFOSZ bits track the number of words in the FIFO. This 4-bit field identifies the number of valid data samples in the IDP FIFO.

The SRU_OVF bits (bits 8–15) in the DAI_STAT register provide IDP FIFO overflow status information for each of the channels. These bits are set (= 1), whenever an overflow occurs. When these bits are cleared (= 0), it indicates there is no overflow condition. These read-only bits are W1C bits, which do not automatically reset to 0 when an overflow condition changes to a no-overflow condition. These bits must be reset manually, using the IDP_CLROVR bit in the IDP_CTL0 register. Writing one to this bit clears the overflow conditions for the channels in the DAI_STAT register. Since IDP_CLROVR is a write-only bit, it always returns low when read.

FIFO to Memory Data Transfer

The data from each of the eight IDP channels is inserted into an eight-register deep FIFO, which can only be transferred to the core's memory space sequentially. Data is moved into the FIFO as soon as it is fully received. When more than one channel has data ready, the channels access the FIFO with fixed priority, from a low to high channel number (that is, channel 0 is the highest priority and channel 7 is the lowest priority).

One of two methods can be used to move data from the IDP FIFO to internal memory:

- The core can remove data from the FIFO manually by reading the memory-mapped register, IDP_FIFO. The output of the FIFO is held in the (read-only) IDP_FIFO register. When this register is read, the corresponding element is removed from the IDP FIFO, and the next element is moved into the IDP_FIFO register.

A mechanism is provided to generate an interrupt when more than a specified number of words are in the FIFO. This interrupt signals the core to read the `IDP_FIFO` register.

This method of moving data from the IDP FIFO is described in the next section, [“IDP Transfers Using the Core”](#).

- Eight dedicated DMA channels can sort and transfer the data into one buffer per source channel. When the memory buffer is full, the DMA channel raises an interrupt in the DAI interrupt controller.

This method of moving data from the IDP FIFO is described in [“IDP Transfers Using DMA”](#) on page 7-20.

IDP Transfers Using the Core

The output of the FIFO can be directly fetched by reading from the `IDP_FIFO` register. The `IDP_FIFO` register is used only to read and remove the top sample from the FIFO, which is eight locations deep.

As data is read from the `IDP_FIFO` register, it is removed from the FIFO and new data is copied into the register. The contents of the `IDP_NSET` bits (bits 3–0 in the `IDP_CTL0` register) represent a threshold number of entries (N) in the FIFO. When the FIFO fills to a point where it has more than N words (data in FIFO exceeds the value set in the `IDP_NSET` bit field), a DAI interrupt is generated. This DAI interrupt corresponds to the `IDP_FIFO_GTN_INT` bit, the eighth-level interrupt in the `DAI_IRPTL_L` or `DAI_IRPTL_H` registers. The core uses this interrupt to detect when data needs to be read.

Starting an Interrupt-Driven Transfer

To start an interrupt-driven transfer:

1. Clear and halt FIFO by setting (= 1) and clearing (= 0) the `IDP_ENABLE` bit (bit 7 in the `IDP_CTL0` register).
2. Set the required values for:
 - `IDP_SMODEx` bits in the `IDP_CTL0` register to specify the frame sync format for the serial inputs (left-justified sample pair, right-justified sample pair, or I²S mode).
 - `IDP_Pxx_PDAPMASK` bits in the `IDP_PP_CTL` register to specify the input mask, if the PDAP is used.
 - `IDP_PORT_SELECT` bits in the `IDP_PP_CTL` register to specify input from the DAI pins, if the PDAP is used.
 - `IDP_PDAP_CLKEDGE` bit (bit 29) in the `IDP_PP_CTL` register to specify if data is latched on the rising or falling clock edge, if the PDAP is used.
3. Keep the clock and frame sync inputs of all serial inputs and/or the PDAP connected to low. Use the `SRU_CLK2`, `SRU_CLK3`, `SRU_FS2`, and `SRU_FS3` registers to specify these inputs. See [“Group A Connections—Clock Signals” on page 4-19](#) and [“Group C Connections—Frame Sync Signals” on page 4-31](#).
4. Connect all of the inputs to the IDP by writing to the `SRU_DAT4`, `SRU_DAT5`, `SRU_FS2`, `SRU_FS3`, `SRU_CLK2`, and `SRU_CLK3` registers. Connect the clock and frame sync of any unused ports to low.
5. Set the desired value for the *N_SET* variable (the `IDP_NSET` bits, 3–0, in the `IDP_CTL0` register).

6. Set the `IDP_FIFO_GTN_INT` bit (bit 8 of the `DAI_IRPTL_RE` register) to HIGH and set the corresponding bit in the `DAI_IRPTL_FE` register to low to unmask the interrupt. Set bit 8 of the `DAI_IRPTL_PRI` register (`IDP_FIFO_GTN_INT`) as needed to generate a high priority or low priority core interrupt when the number of words in the FIFO is greater than the value of N set.
7. Enable the PDAP by setting the `IDP_PDAP_EN` bit (bit 31 in the `IDP_PP_CTL` register), if required.
8. Enable the IDP by setting the `IDP_ENABLE` bit (bit 7 in the `IDP_CTL0` register) and the `IDP_ENx` bits in the `IDP_CTL1` register.



Do *not* set the `IDP_DMA_EN` bit (bit 5 of the `IDP_CTL0` register) as this enables DMA transfers.

Core Transfer Notes

The following items provide general information about interrupt-driven transfers.

- The three LSBs of FIFO data are the encoded channel number. These are transferred “as is” for this mode. These bits can be used by software to decode the source of data.
- The number of data samples in the FIFO at any time is reflected in the `IDP_FIFOSZ` bit field (bits 31–28 in the `DAI_STAT` register), which tracks the number of samples in FIFO.

When using the interrupt scheme, the `IDP_NSET` bits (bits 3–0 of the `IDP_CTL0` register) can be set to N , so $N + 1$ data can be read from the FIFO in the interrupt service routine (ISR).

- If the `IDP_BHD` bit (bit 4 in the `IDP_CTL0` register) is not set, attempts to read more data than is available in the FIFO results in a core hang.

FIFO to Memory Data Transfer

- When the data transfer to the core is 32 bits, as in the case of PDAP data or I²S and left-justified modes with 32 bits, there is no channel information in the data. Therefore, PDAP or I²S and left-justified 32 bit modes can not be used with other channels in the core/interrupt-driven mode.

IDP Transfers Using DMA

The ADSP-21367/8/9 and ADSP-2137x processors support two types of DMA transfers, simple and ping-pong.

Simple DMA

This DMA access is enabled when the `IDP_DMA_EN` bit (bit 5 of the `IDP_CTL0` register) is set (= 1) and the `IDP_DMA_ENx` bits in the `IDP_CTL1` register are set to select a particular channel.

The DMA is performed according to the parameters set in the various DMA registers and IDP control registers. The DMA transfer is completed when the count register (`IDP_DMA_Cx`) reaches zero. The `IDP_DMA_EN` bit and `IDP_DMA_ENx` bits must be reset before starting another DMA. An interrupt is generated at the end of DMA transfer.

Starting a Simple DMA Transfer

To start a DMA transfer from the FIFO to memory:

1. Clear and halt the FIFO by setting (= 1) and then clearing (= 0) the `IDP_ENABLE` bit (bit 7 in the `IDP_CTL0` register).
2. While the `IDP_DMA_EN` and `IDP_ENABLE` bits are low, set the values for the DMA parameter registers that correspond to channels 7–0. If some channels are not going to be used, then the corresponding parameter registers can be left in their default states:

- Index registers (IDP_DMA_Ix)
- Modifier registers (IDP_DMA_Mx)
- Counter registers (IDP_DMA_Cx)

For each of these registers, x is 0 to 7. Refer to “[DMA Channel Parameter Registers](#)” on page 7-27.

3. Keep the clock and the frame sync input of the serial inputs and/or the PDAP connected to low, by setting proper values in the SRU_CLK2, SRU_CLK3, SRU_FS2, and SRU_FS3 registers.
4. Set required values for:
 - IDP_SMODEx bits in the IDP_CTL0 register to specify the frame sync format for the serial inputs (left-justified sample pair, right-justified sample pair, or I²S modes).
 - IDP_Pxx_PDAPMASK bits in the IDP_PP_CTL register to specify the input mask, if the PDAP is used.
 - IDP_PORT_SELECT bits in the IDP_PP_CTL register to specify input from the DAI pins or the IDP, if the PDAP is used.
 - IDP_PDAP_CLKEDGE bit (bit 29) in the IDP_PP_CTL register to specify if data is latched on the rising or falling clock edge, if the PDAP is used.
5. Connect all of the inputs to the IDP by writing to the SRU_DAT4, SRU_DAT5, SRU_FS2, SRU_FS3, SRU_CLK2, and SRU_CLK3 registers. Keep the clock and frame sync of the ports connected to low when data transfer is not intended.

FIFO to Memory Data Transfer

6. Enable DMA, IDP, and PDAP (if required) by setting each of the following bits = 1:

- IDP_DMA_EN bit (bit 5 of the IDP_CTL0 register)
- IDP_DMA_ENx bits in IDP_CTL1 register to enable the DMA of the selected channel
- IDP_PDAP_EN bit (bit 31 in IDP_PP_CTL register)
- IDP_ENx of IDP_CTL1 to enable the selected channel
- IDP_ENABLE bit (bit 7 in the IDP_CTL0 register)

A DAI interrupt is generated at the end of each DMA.

7. After the DMA completes, connect the clock and frame sync signals to 0.

Ping-Pong DMA

This mode gets activated when the IDP_DMA_EN bit of the IDP_CTL0 register and the IDP_PINGx bit in the IDP_CTL1 register are set for a particular channel.

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

Starting Ping-Pong DMA Transfers

To start a ping-pong DMA transfer from the FIFO to memory:

1. Clear and halt the FIFO by setting (= 1) and then clearing (= 0) the `IDP_ENABLE` bit (bit 7 in the `IDP_CTL0` register).
2. While the `IDP_DMA_EN` and `IDP_ENABLE` bits are low, set the values for the following DMA parameter registers that correspond to channels 7–0. If some channels are not going to be used, then the corresponding parameter registers can be left in their default states:
 - First index registers `IDP_DMA_AIx`
 - Second index registers `IDP_DMA_BIx`
 - Modifier register `IDP_DMA_Mx`
 - Counter register `IDP_DMA_PCx`. For each of these registers `x` is 0–7 which corresponds to channels 0 to 7. See [“IDP Ping-Pong Count Registers \(IDP_DMA_PCx\)”](#) on [page A-73](#).
3. Keep the clock and the frame sync input of the serial inputs and/or the PDAP connected to low, by setting proper values in the `SRU_CLK2` and `SRU_CLK3` as well as the `SRU_FS2` and `SRU_FS3` registers. [For more information, see “DAI/SRU1 Connection Groups” on page 4-18.](#)
4. Set the required values for:
 - `IDP_SMODEx` bits in the `IDP_CTL0` register to specify the frame sync format for the serial inputs (I^2S , left-justified sample pair, or right-justified sample pair modes).

FIFO to Memory Data Transfer

- IDP_Pxx_PDAPMASK bits in the IDP_PP_CTL register to specify the input mask, if the PDAP is used. [For more information, see “Parallel Data Acquisition Port Control Register \(IDP_PP_CTL\)” on page A-74.](#)
 - IDP_PORT_SELECT bits in the IDP_PP_CTL register to specify input from the DAI pins or the data pins, if the PDAP is used.
 - IDP_PDAP_CLKEDGE bit (bit 29) in the IDP_PP_CTL register to specify if data is latched on the rising or falling clock edge, if the PDAP is used.
5. Connect all of the inputs to the IDP by writing to the SRU_DAT4 and SRU_DAT5, SRU_FS2 and SRU_FS3, SRU_CLK2 and SRU_CLK3 registers. Keep the clock and frame sync of the ports connected to low when data transfer is not intended.
 6. Enable DMA, IDP, and PDAP (if required) by setting each of the following bits = 1:
 - IDP_DMA_EN bit (bit 5 of the IDP_CTL0 register)
 - IDP_PINGx bit in IDP_CTL1 register to enable the ping-pong DMA of the selected channel
 - IDP_PDAP_EN bit (bit 31 in IDP_PP_CTL register)
 - IDP_ENx of IDP_CTL1 to enable the selected channel
 - IDP_ENABLE bit (bit 7 in the IDP_CTL0 register)
 7. After the DMA completes, connect the clock and frame sync signals to 0.



An interrupt is generated after every ping and pong DMA transfer (when the count = 0).

DMA Transfer Notes

The following items provide general information about DMA transfers.

- A DMA can be interrupted by changing the `IDP_DMA_EN` bit in the `IDP_CTL0` register. None of the other control settings (except for the `IDP_ENABLE` bit) should be changed. Clearing the `IDP_DMA_EN` bit (= 0) does not affect the data in the FIFO, it only stops DMA transfers. If the IDP remains enabled, an interrupted DMA can be resumed by setting the `IDP_DMA_EN` bit again. But resetting the `IDP_ENABLE` bit flushes the data in the FIFO. If the bit is set again, FIFO starts accepting new data.
- Using DMA transfers overrides the mechanism that is used for interrupt-driven manual reads from the FIFO. When the `IDP_DMA_EN` bit and at least one `IDP_DMA_ENx` bit in the `IDP_CTL1` register are set, the eighth interrupt in the `DAI_IRPTL_L` or `DAI_IRPTL_H` registers (`IDP_FIFO_GTN_INT`) is *not* generated. This interrupt detects the condition that the number of data available in FIFO is more than the number set in the `IDP_NSET` bits (bits 3–0 of the `IDP_CTL0` register).
- At the end of the DMA transfer for individual channels, interrupts are generated. These interrupts are generated after the last DMA data from a particular channel has been transferred to memory. These interrupts are mapped to the `IDP_DMA7_INT` bit (bit 17), and to the `IDP_DMA0_INT` bit (bit 10) in the `DAI_IRPTL_L` or `DAI_IRPTL_H` registers and they generate interrupts when they are set (= 1). These bits are ORed and reflected in high-level interrupts sent to the core.
- If the combined data rate from the channels is more than the DMA can service, a FIFO overflow occurs. This condition is reflected for each channel by the individual overflow bits (`SRU_OVF`) in the `DAI_STAT` register. These are W1C bits that must be cleared by writing to the `IDP_CLR0VR` bit (bit 6 of the `IDP_CTL0` register).

FIFO to Memory Data Transfer

When an overflow occurs, incoming data from IDP channels is not accepted into the FIFO, and data values are lost. New data is only accepted once space is again created in the FIFO.

- For serial input channels, data is received in an alternating fashion from left and right channels. Data is not pushed into the FIFO as a full left/right frame. Rather, data is transferred as alternating left/right words as it is received. For the PDAP and 32-bit (non-audio) serial input, data is transferred as packed 32-bit words.
- The state of all eight DMA channels is reflected in the IDP_DMAx_STAT bits (bits 24–17 of DAI_STAT register). These bits are set once the IDP_DMA_EN bit and IDP_DMA_ENx bits are set, and remain set until the last data from that channel is transferred. Even if IDP_DMA_EN bit and IDP_DMA_ENx bits remain set, this bit clears once the required number of data transfers takes place. [For more information, see “DAI Pin Buffer Status Register \(DAI_PIN_STAT\)” on page A-112.](#)



- Note that when a DMA channel is not used (that is, parameter registers are at their default values) that DMA channel's corresponding IDP_DMAx_STAT bit is set (= 1).
- The three LSBs of data from the serial inputs are channel encoding bits. Since the data is placed into a separate buffer for each channel, these bits are not required and are set to low when transferring data to internal memory through the DMA. Bit 3 still contains the left/right status information. In the case of PDAP data or 32-bit I²S and left-justified modes, these three bits are part of the 32-bit data.
 - An interrupt is generated at the end of a DMA, which is cleared by reading the DAI_IRPTL_H or DAI_IRPTL_L registers.
 - A read of the DAI_IRPTL_H_SH register provides the same data as a read of the DAI_IRPTL_H register. Likewise, a read of the DAI_IRPTL_L_SH register provides the same data as a read of the

DAI_IRPTL_L register. Reading these DAI shadow registers (DAI_IRPTL_H_SH and DAI_IRPTL_L_SH) does not destroy the contents of the DAI_IRPTL_H and DAI_IRPTL_L registers.

- The IDP can run both simple and ping-pong DMAs in different channels. When running simple DMA, initialize the corresponding IDP_DMA_Ix, IDP_DMA_Mx, and IDP_DMA_Mx registers. When running ping-pong DMA, initialize the corresponding IDP_DMA_AIx, IDP_DMA_BIx, IDP_DMA_Mx, and IDP_DMA_PCx registers.
- A new feature of dropping DMA requests from the FIFO has been added. If one channel has finished its DMA, but the IDP_DMA_EN bit is still high, any data corresponding to that channel is skipped by the DMA controller. This feature is provided to avoid stalling the DMA of other channels, which are still in an active DMA state. To avoid this data loss, programs can toggle IDP_DMA_EN low.
- Disabling IDP DMA by resetting the IDP_DMA_EN bit requires 1 HCLK cycle. Disabling an individual channel DMA by resetting the IDP_DMA_ENx bits requires 2 HCLK cycles.

DMA Channel Parameter Registers

The eight DMA channels each have two sets of registers for simple and ping-pong DMA. For simple DMA, an I-register (index pointer, 19 bits), an M-register (modifier/stride, 6 bits), and a C-register (count, 16 bits) are used. For ping-pong DMA, A and B index registers (AI/BI register pointer, 19 bits) and a PC register (DMA count, 16 bits) are used, along with the M-register.

FIFO to Memory Data Transfer

The IDP DMA parameter registers have these functions:

- **Internal index registers** (IDP_DMA_Ix, IDP_DMA_AIx, IDP_DMA_BIx). Index registers provide an internal memory address, acting as a pointer to the next internal memory location where data is to be written.
- **Internal modify registers** (IDP_DMA_Mx). Modify registers provide the signed increment by which the DMA controller post-modifies the corresponding internal memory index register after each DMA write.
- **Count registers** (IDP_DMA_Cx, IDP_DMA_PCx). Count registers indicate the number of words remaining to be transferred to internal memory on the corresponding DMA channel.

For a descriptions of these registers see [“Input Data Port DMA Control Registers” on page A-70](#) and [“Input Data Port Ping-Pong DMA Registers” on page A-72](#).

IDP (DAI) Interrupt Service Routines for DMAs

The IDP can trigger either the high priority DAI core interrupt reflected in the DAI_IRPTL_H register or the low priority DAI core interrupt reflected in the DAI_IRPTL_L register. The ISR must read the corresponding DAI_IRPTL_H or DAI_IRPTL_L register to find all the interrupts currently latched. The DAI_IRPTL_H register reflects the high priority interrupts and the DAI_IRPTL_L register reflects the low priority interrupts. When these registers are read, it clears the latched interrupt bits. This is a destructive read.

The following steps describe how an IDP ISR is handled.

1. When the DMA for a channel completes, an interrupt is generated and program control jumps to the ISR.
2. The program clears the `IDP_DMA_EN` bit in the `IDP_CTL0` register (= 0).
3. The program should read the `DAI_IRPTL_L` or `DAI_IRPTL_H` registers to determine which DMA channels have completed. Programs may read these register's shadow registers (`DAI_IRPTL_L_SH` and `DAI_IRPTL_H_SH`) without clearing the contents of the primary registers.

To ensure that the DMA of a particular IDP channel is complete, (all data is transferred into internal memory) wait until the `IDP_DMAx_STAT` bit of that channel becomes zero in the `DAI_STAT` register. This is required if a high priority DMA (for example a SPORT DMA) is occurring at the same time as the IDP DMA.

As each DMA channel completes, a corresponding bit in either the `DAI_IRPTL_L` or `DAI_IRPTL_H` registers for each DMA channel is set (`IDP_DMAx_INT`). Refer to [“DAI Interrupt Controller Registers” on page A-112](#) for more information on the `DAI_IRPTL_L` or `DAI_IRPTL_H` registers.


4. Reprogram the DMA registers for finished DMA channels.

More than one DMA channel may have completed during this time period. For each channel, a bit is latched in the `DAI_IRPTL_L` or `DAI_IRPTL_H` registers. Ensure that the DMA registers are reprogrammed. If any of the channels are not used, then its clock and frame sync must be held low.

FIFO to Memory Data Transfer

5. Read the `DAI_IRPTL_L` or `DAI_IRPTL_H` registers to see if more interrupts have been generated.
 - If the value(s) are not zero, repeat step 4.
 - If the value(s) are zero, continue to step 6.
6. Re-enable the `IDP_DMA_EN` bit in the `IDP_CTL0` register (set to 1).
7. Exit the ISR.

If a zero is read in step 5 (no more interrupts are latched), then all of the interrupts needed for that ISR have been serviced. If another DMA completes after step 5 (that is, during steps 6 or 7), as soon as the ISR completes, the ISR is called again because the OR of the latched bits will be nonzero again. DMAs in process run to completion.

 If step 5 is not performed, and a DMA channel expires during step 4, then when IDP DMA is re-enabled (step 6), the completed DMA will *not* have been reprogrammed and its buffer will overrun.

FIFO Overflow

If the data out of the FIFO (either through DMA or core reads) is not sufficient to transfer at the combined data rate of all the channels, then a FIFO overflow can occur. When this happens, new data is not accepted. Additionally, data coming from the serial input channels (except for 32-bit I²S and left-justified modes) are not accepted in pairs, so that alternate data from a channel is always from left and right channels. If overflow occurs, then sticky bits in the `DAI_STAT` register are set and an interrupt is generated. Data is accepted again when space has been created in the FIFO.

Input Data Port Programming Example

[Listing 7-1](#) shows a data transfer using an interrupt service routine (ISR). The transfer takes place through the digital applications interface (DAI). This code implements the algorithm outlined in [“FIFO to Memory Data Transfer” on page 7-16](#).

Listing 7-1. Interrupt-Driven Data Transfer

```

/* Using Interrupt-Driven Transfers from the IDP FIFO */

#define IDP_ENABLE    (8)          /* IDP_ENABLE = IDP_CTL0[7] */
#define IDP_CTL0      (0x24B0)    /* Memory-mapped register */
#define IDP_FIFO_GTN_INT (8)      /* Bit 8 in interrupt regs */
#define IDP_FIFO      (0x24D0)    /* IDP FIFO packing mode */
#define DAI_IRPTL_FE  (0x2480)    /* Falling edge int latch */
#define DAI_IRPTL_RE  (0x2481)    /* Rising edge int latch */
#define DAI_IRPTL_PRI (0x2484)    /* Interrupt priority */

.section/dm seg_dmda;
.var OutBuffer[6];

.section/pm seg_pmco;

initIDP:

    r0 = dm(IDP_CTL0);          /* Reset the IDP */
    r0 = BSET r0 BY IDP_ENABLE;
    dm(IDP_CTL0) = r0;
    r0 = BCLR r0 BY IDP_ENABLE;

    r0 = BCLR r0 BY 10;         /* Set IDP serial input channel 0 */
    r0 = BCLR r0 BY 9;          /* to receive in I2S format */
    r0 = BCLR r0 BY 8;

```

Input Data Port Programming Example

```
dm(IDP_CTL0) = r0;
/*****
/* Connect the clock, data and frame sync of IDP */
/*   channel 0 to DAI pin buffers 10, 11 and 12. */
*****/

/*   Connect IDP0_CLK_I to DAI_PB10_0   */
/*       (SRU_CLK1[19:15] = 01001)      */

/*   Connect IDP0_DAT_I to DAI_PB11_0   */
/*       (SRU_DAT3[11:6] = 001010)      */

/*   Connect IDP0_FS_I to DAI_PB12_0    */
/*       (SRU_FS1[19:15] = 01011)       */

/*****
/* Pin buffers 10, 11 and 12 are always being used as */
/*   inputs. Tie their enables to LOW (never driven). */
*****/

/* Connect PBEN10_I to LOW */
/* (SRU_PIN1[29:24] = 111110) */

/* Connect PBEN11_I to LOW */
/*   (SRU_PIN2[5:0] = 111110) */

/* Connect PBEN12_I to LOW */
/*   (SRU_PIN2 11-6 = 111110) */

/*****
/* Assign a value to N_SET. An interrupt will be raised */
/*   when there are N_SET+1 words in the FIFO.          */
*****/
```

```

r0 = dm(IDP_CTL0);          /* N_SET = 6 */
r0 = BSET r0 BY 0;
r0 = BSET r0 BY 1;
r0 = BSET r0 BY 2;
r0 = BCLR r0 BY 3;
dm(IDP_CTL0) = r0;

r0 = dm(DAI_IRPTL_RE);      /* Unmask for rising edge */
r0 = BSET r0 BY IDP_FIFO_GTN_INT;
dm(DAI_IRPTL_RE) = r0;

r0 = dm(DAI_IRPTL_FE);      /* Mask for falling edge */
r0 = BCLR r0 BY IDP_FIFO_GTN_INT;
dm(DAI_IRPTL_FE) = r0;

r0 = dm(DAI_IRPTL_PRI);     /* Map to high priority in core */
r0 = BSET r0 BY IDP_FIFO_GTN_INT;
dm(DAI_IRPTL_PRI) = r0;

r0 = dm(IDP_CTL0);          /* Start the IDP */
r0 = BSET r0 BY IDP_ENABLE;
dm(IDP_CTL0) = r0;

initIDP.end:

IDP_ISR:
    i0 = OutBuffer;
    m0 = 1;
    LCNTR = 5, DO RemovedFromFIFO UNTIL LCE;
    r0 = dm(IDP_FIFO);
    dm(i0,m0) = r0;
RemovedFromFIFO:
    RTI;
IDP_ISR.end:

```

Input Data Port Programming Example

8 PULSE WIDTH MODULATION

Pulse width modulation (PWM) is a technique for controlling analog circuits with a microprocessor's digital outputs. PWM is employed in a wide variety of applications, ranging from measurement to communications to power control and conversion.

One of the advantages of PWM is that the signal remains digital all the way from the processor to the controlled system; no digital-to-analog conversion is necessary. By maintaining a digital signal throughout a system, noise effects are minimized.

PWM Implementation

The PWM modules in the ADSP-21367/8/9 and ADSP-2137x SHARC processors are flexible, programmable, PWM waveform generators that produce switching patterns for various purposes related to motor control, electronic valve control, or audio power control. The PWM module is comprised of four identical groups that contain four PWM outputs each, allowing 16 PWM outputs in total. [Figure 8-1](#) shows a single group. The PWM generator can produce either center-aligned or edge-aligned PWM waveforms. In addition, it can generate complementary signals on two outputs in paired mode or independent signals in non-paired mode.

PWM Waveforms

The PWM module can generate waveforms that are either edge-aligned (left-justified) or center-aligned. Each waveform is described in detail in the following sections.

PWM Implementation

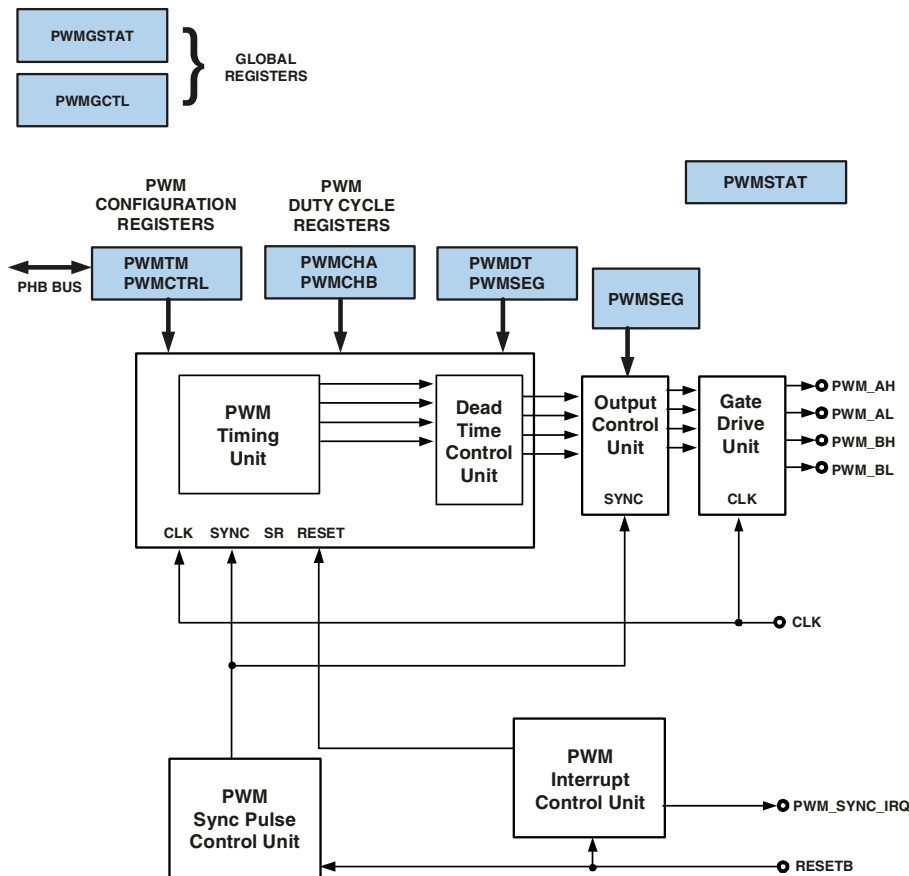


Figure 8-1. Example PWM Module Block Diagram

Edge-Aligned Mode

In edge-aligned mode, the PWM waveform is left-justified in the period window. A duty value of zero, programmed through the PWM_{AX} registers, produces a PWM waveform with 50% duty cycle. For even values of period, the PWM pulse width is exactly $\text{period} \div 2$, whereas for odd values

of period, it is equal to $\text{period} \div 2$ (rounded up). Therefore, for a duty value programmed in two's-complement, the PWM pulse width is given by:

$$\text{Width} = \lceil (\text{period}) \div 2 \rceil + \text{duty}$$

To generate constant logic high on PWM output, program the duty register with the value $\geq + \text{period} \div 2$.

To generate constant logic low on PWM output, program the duty register with the value $\geq - \text{period} \div 2$.

For example, using an odd period of $p = 2n + 1$, the counter within the PWM generator counts as $(-n \dots 0 \dots +n)$. If the period is even ($p = 2n$), then the counter counts as $(-n+1 \dots 0 \dots n)$.

For more information, see “PWM Channel Duty Control Registers (PWMAx, PWMBx)” on page A-84.

Center-Aligned Mode

Most of the following description applies to paired mode, but it can also be applied to non-paired mode, the difference being that each of the four outputs from a PWM group is independent. Within center-aligned mode, there are several options to choose from.

Center-aligned single update mode. Duty cycle values are programmable only once per PWM period, so the resultant PWM patterns are symmetrical about the midpoint of the PWM period.

Center-aligned, double-update mode. Duty cycle values are programmable only twice per PWM period. This second updating of the PWM registers is implemented at the midpoint of the PWM period, producing asymmetrical PWM patterns that produce lower harmonic distortion in three-phase PWM inverters.

PWM Implementation

Center-aligned, paired mode. Generates complementary signals on two outputs.

Center-aligned, non-paired mode. Generates complementary signals on independent signals.

In paired mode, the two's-complement integer value in the 16-bit read/write duty cycle registers, (PWMAx and PWMBx), control the duty cycles of the four PWM output signals on the `pwm_a`, `pwm_ah`, `pwm_b1`, and `pwm_bh` respectively. The duty cycle registers are programmed in two's-complement integer counts of the fundamental time unit, t_{PCLK} (the peripheral clock of the ADSP-21367/8/9 and ADSP-2137x processors) and define the desired on-time of the high-side PWM signal over half the PWM period. The duty cycle register range is from $(-PWMPERIOD \div 2 - PWMDT)$ to $(+PWMPERIOD \div 2 + PWMDT)$, which, by definition, is scaled such that a value of 0 represents a 50% PWM duty cycle.

Each group in the PWM module (0–3) has its own set of registers which control the operation of that group. The operating mode of the PWM block (single or double-update mode) is selected by the `PWM_UPDATE` bit (bit 2) in the PWM control (`PWMCTRL3-0`) registers. Status information about each individual PWM group is available to the program in the PWM status (`PWMSTAT3-0`) registers. Apart from the local control and status registers for each PWM group, there is a single PWM global control register (`PWMGCTL`) and a single PWM global status register (`PWMGSTAT`). The global control register allows programs to enable or disable the four groups in any combination, which provides synchronization across the four PWM groups. The global status register shows the period completion status of each group.

On period completion, the corresponding bit in the `PWMGSTAT` register is set and remains sticky. In the interrupt service routine (ISR), the program should first read the global status register and clear all the intended bits by explicitly writing 1. This also clears the `PWM_INT` bit. Interrupts from individual groups can be disabled by the `PWM_IRQEN` bit in the local `PWMCTRLx`

registers of that group. The period completion status bits in the `PWM_GSTAT` register are set independently of the corresponding `PWM_IRQEN` bit, but interrupt generation depends on the `PWM_IRQEN` bit.

Switching Frequencies

The 16-bit read/write PWM period registers, (`PWMPERIOD3-0`), control the PWM switching frequency. The fundamental timing unit of the PWM controller is t_{PCLK} . Therefore, for a 100 MHz peripheral clock ($PCLK$), the fundamental peripheral clock increment, $PCLK$, is 10 ns. The value written to the `PWMPERIODx` register is effectively the number of t_{PCLK} clock increments in half a PWM period. The required `PWMPERIODx` value as a function of the desired PWM switching frequency (f_{PWM}) is given by:

$$PWMPERIOD = \frac{f_{PCLK}}{2 \times f_{PWM}}$$

Therefore, the PWM switching period, T_s , can be written as:

$$T_s = 2 \times PWMPERIOD \times PCLK$$

For example, for a 100 MHz f_{PCLK} and a desired PWM switching frequency of 10 kHz ($T_s = 100 \mu s$), the correct value to load into the `PWMPERIODx` register is:

$$PWMPERIOD = \frac{100 \times 10^6}{2 \times 10 \times 10^3} = 5000$$

PWM Implementation

The largest value that can be written to the 16-bit `PWMPERIODx` register is `0xFFFF` = 65,535 which corresponds to a minimum PWM switching frequency of:

$$f_{(PWM),min} = \frac{100 \times 10^6}{2 \times 65535} = 763Hz$$

Also note that `PWMPERIOD` values of 0 and 1 are not defined and should not be used when the PWM outputs or PWM sync is enabled.

Dead Time

The second important parameter that must be set up in the initial configuration of the PWM block is the switching dead time. This is a short delay time introduced between turning off one PWM signal (say `AH`) and turning on the complementary signal, (`AL`). This short time delay is introduced to permit the power switch to turn off (`AH` in this case) to completely recover its blocking capability before the complementary switch is turned on. This time delay prevents a potentially destructive short-circuit condition from developing across the dc link capacitor of a typical voltage source inverter.

The 10-bit, read/write `PWMDT3-0` registers control the dead time. The dead time, T_d , is related to the value in the `PWMDTx` registers by:

$$T_d = PWMDT \times 2 \times t_{CLK}$$

Therefore, a `PWMDT` value of `0x00A` (= 10), introduces a 200 ns delay between when the PWM signal (for example `AH`) is turned off and its complementary signal (`AL`) is turned on. The amount of dead time can therefore be programmed in increments of $2 t_{CLK}$ (or 20 ns for a 100 MHz peripheral clock). The `PWMDTx` registers are 10-bit registers, and the

maximum value they can contain is 0x3FF (= 1023) which corresponds to a maximum programmed dead time of:

$$T_{d,max} = 1023 \times 2 \times t_{CLK} = 1023 \times 2 \times 10 \times 10^{-9} = 20.5\mu s$$

This equates to an f_{CLK} rate of 100 MHz. Note that the dead time can be programmed to be 0 by writing 0 to the PWMDTx registers (see “PWM Dead Time Registers (PWMDTx)” on page A-85).

Duty Cycles

The two 16-bit read/write duty cycle registers, PWMA and PWMB control the duty cycles of the four PWM output signals on the PWM pins when not in switch reluctance mode. The two’s-complement integer value in the PWMA register controls the duty cycle of the signals on pwm_ah and pwm_al. The two’s-complement integer value in the PWMB register controls the duty cycle of the signals on pwm_bh and pwm_bl. The duty cycle registers are programmed in two’s-complement integer counts of the fundamental time unit, t_{CLK} , and define the desired on-time of the high-side PWM signal produced by the three-phase timing unit over half the PWM period. The duty cycle register range is from:

$$(-PWPERIOD \div 2 - PWMDT) \text{ to } (+PWPERIOD \div 2 + PWMDT)$$

which, by definition, is scaled such that a value of 0 represents a 50% PWM duty cycle. The switching signals produced by the three-phase timing unit are also adjusted to incorporate the programmed dead time value in the PWMDT register. The three-phase timing unit produces active low signals so that a low level corresponds to a command to turn on the associated power device.

Duty Cycles and Dead Time

A typical pair of PWM outputs (in this case for `pwm_ah` and `pwm_al`) from the timing unit are shown in [Figure 8-2](#) for operation in single-update mode. All illustrated time values indicate the integer value in the associated register and can be converted to time by simply multiplying by the fundamental time increment, (t_{PCLK}) and comparing this to the two's-complement counter. Note that the switching patterns are perfectly symmetrical about the midpoint of the switching period in single-update mode since the same values of the `PWMAx`, `PWMPERIODx`, and `PWMDTx` registers are used to define the signals in both half cycles of the period. Further, the programmed duty cycles are adjusted to incorporate the desired dead time into the resulting pair of PWM signals. As shown in [Figure 8-2](#), the dead time is incorporated by moving the switching instants of both PWM signals (`pwm_ah` and `pwm_al`) away from the instant set by the `PWMAx` registers. Both switching edges are moved by an equal amount ($PWMDT \times t_{PCLK}$) to preserve the symmetrical output patterns. Also shown is the `PWM_PHASE` bit of the `PWMSTAT` register that indicates whether operation is in the first or second half cycle of the PWM period.

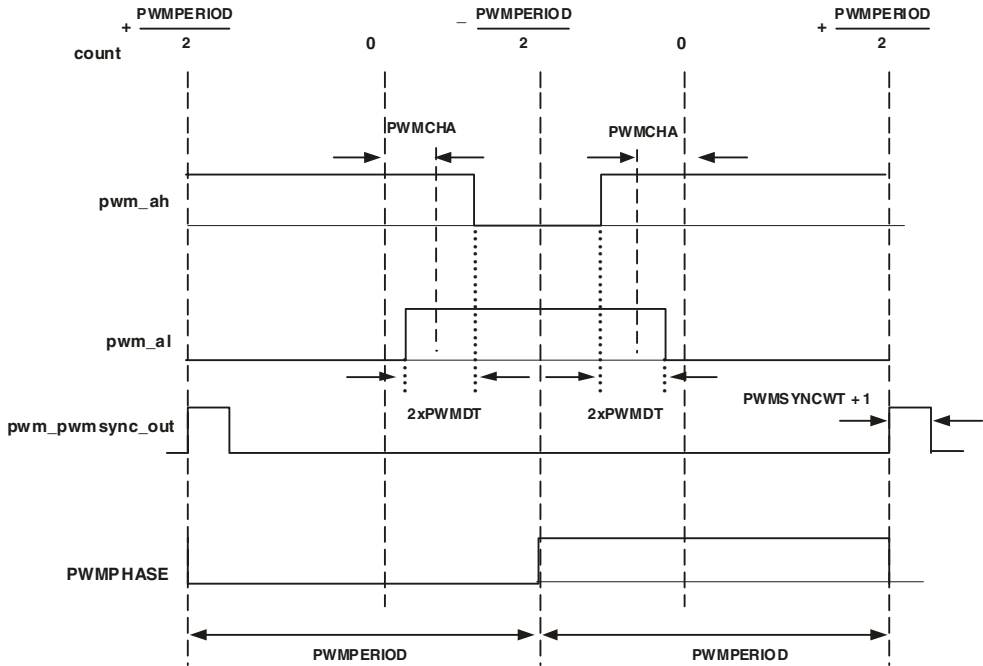


Figure 8-2. Center-Aligned Paired PWM in Single-Update Mode, Low Polarity

The resulting on-times (active low) of the PWM signals over the full PWM period (two half periods) produced by the PWM timing unit and illustrated in [Figure 8-3 on page 8-11](#) may be written as:

The range of T_{AH} is:

$$[0 - 2 \times PWMPERIOD \times t_{PCLK}]$$

and the corresponding duty cycles are:

$$T_{AH} = (PWMPERIOD - 2 \times (PWMCHA + PWMDT) \times t_{PCLK}$$

PWM Implementation

The range of T_{AL} is:

$$[0 - 2 \times PWMPERIOD \times t_{PCLK}]$$

and the corresponding duty cycles are:

$$d_{AH} = \frac{t_{AH}}{T_S} = \frac{1}{2} + \frac{PWMCHA - PWMDT}{PWMPERIOD}$$

$$d_{AL} = \frac{t_{AL}}{T_S} = \frac{1}{2} + \frac{PWMCHA - PWMDT}{PWMPERIOD}$$

The minimum permissible value of T_{AH} and T_{AL} is zero, which corresponds to a 0% duty cycle, and the maximum value is T_S , the PWM switching period, which corresponds to a 100% duty cycle. Negative values are not permitted.

The output signals from the timing unit for operation in double-update mode are shown in [Figure 8-3](#). This illustrates a general case where the switching frequency, dead time, and duty cycle are all changed in the second half of the PWM period. The same value for any or all of these quantities can be used in both halves of the PWM cycle. However, there is no guarantee that a symmetrical PWM signal will be produced by the timing unit in this double-update mode. Additionally, [Figure 8-3](#) shows that the dead time is inserted into the PWM signals in the same way as in single-update mode.

In general, the on-times (active low) of the PWM signals over the full PWM period in double-update mode can be defined as:

$$T_S = (PWMPERIOD_1 + PWMPERIOD_2) \times t_{PCLK}$$

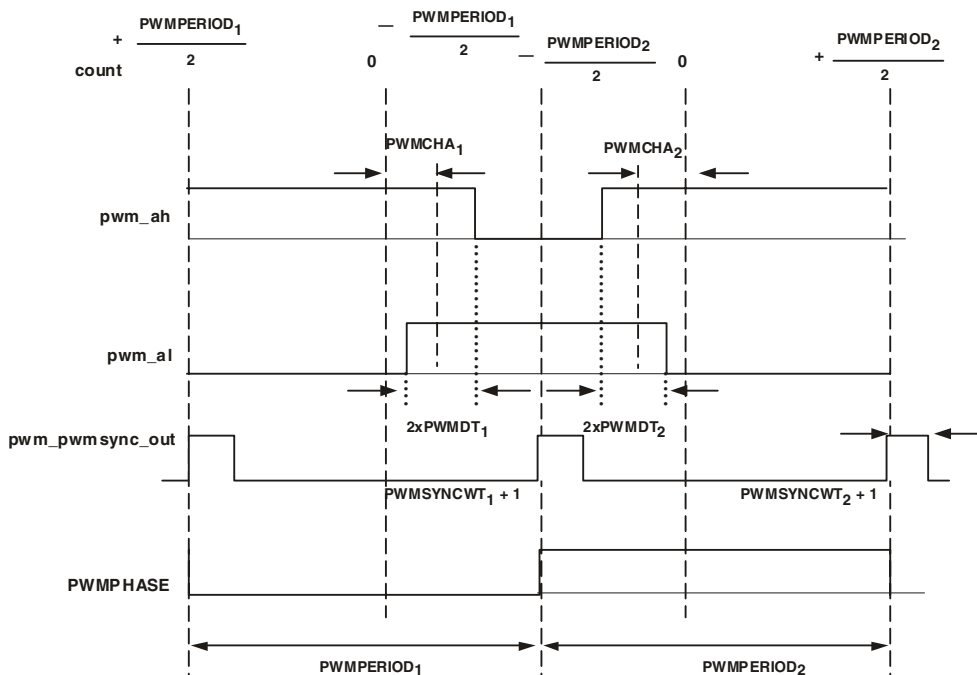


Figure 8-3. Center-Aligned Paired PWM in Double-Update Mode, Low Polarity

$$T_{AH} = \left(\frac{PWMPERIOD_1}{2} + \frac{PWMPERIOD_2}{2} + PWMCHA_1 + PWMCHA_2 - PWMDT_1 - PWMDT_2 \right) \times t_{PCLK}$$

$$T_{AL} = \left(\frac{PWMPERIOD_1}{2} + \frac{PWMPERIOD_2}{2} - PWMCHA_1 - PWMCHA_2 - PWMDT_1 - PWMDT_2 \right) \times t_{PCLK}$$

PWM Implementation

where subscript 1 refers to the value of that register during the first half cycle and subscript 2 refers to the value during the second half cycle. The corresponding duty cycles are:

$$d_{AH} = \frac{T_{AH}}{T_H} = \frac{1}{2} + \frac{(PWMCHA_1 + PWMCHA_2 - PWMDT_1 - PWMDT_2)}{(PWMPERIOD_1 + PWMPERIOD_2)}$$

$$d_{AL} = \frac{T_{AL}}{T_S} = \frac{1}{2} - \frac{(PWMCHA_1 + PWMCHA_2 + PWMDT_1 + PWMDT_2)}{(PWMPERIOD_1 + PWMPERIOD_2)}$$

since for the general case in double- update mode, the switching period is given by:

$$T_S = (PWMPERIOD_1 + PWMPERIOD_2) \times t_{CLK}$$

Again, the values of T_{AH} and T_{AL} are constrained to lie between zero and T_S . Similar PWM signals to those illustrated in [Figure 8-2](#) and [Figure 8-3](#) can be produced on the BH and BL outputs by programming the $PWMBx$ registers in a manner identical to that described for the $PWMAx$ registers.

Over Modulation

The PWM timing unit is capable of producing PWM signals with variable duty cycle values at the PWM output pins. At the extreme side of the modulation process, settings of 0% and 100% modulation are possible. These two modes are termed full off and full on respectively. Settings that fall between the extremes are considered normal modulation. These settings are explained in further detail below.

- **Full on.** The PWM for any pair of PWM signals is said to operate in full on when the desired high side output of the three-phase timing unit is in the on state (low) between successive `PWMSYNC` rising edges. This state may be entered by virtue of the commanded duty cycle values in conjunction with the setting in the `PWMDTx` registers.
- **Full off.** The PWM for any pair of PWM signals is said to operate in full off when the desired high side output of the three-phase timing unit is in the off state (high) between successive `PWMSYNC` pulses. This state may be entered by virtue of the commanded duty cycle values in conjunction with the setting in the `PWMDTx` registers.
- **Normal modulation.** The PWM for any pair of PWM signals is said to operate in normal modulation when the desired output duty cycle is other than 0% or 100% between successive `PWMSYNC` pulses.

There are certain situations, when transitioning either into or out of either full on or full off, where it is necessary to insert additional *emergency dead time* delays to prevent potential *shoot through conditions* in the inverter.

The use of *crossover*, described in [“Crossover” on page 8-16](#), can also cause outputs to violate the shoot through conditions criteria. These transitions are detected automatically and, if appropriate, the emergency dead time is inserted to prevent the shoot through conditions.

Inserting additional emergency dead time into one of the PWM signals of a given pair during these transitions is only needed if both PWM signals would otherwise be required to toggle within a dead time of each other. The additional emergency dead time delay is inserted into the PWM signal that is toggling into the on state. In effect, the turn on (if turning on during this dead time region) of this signal is delayed by an amount $2 \times PWMDT \times t_{CLK}$ from the rising edge of the opposite output. After this delay, the PWM signal is allowed to turn on, provided the desired output is still scheduled to be in the on state after the emergency dead time delay.

PWM Implementation

Figure 8-4 illustrates two examples of such transitions. In Figure 8-4 (A), when transitioning from normal modulation to full on at the half cycle boundary in double-update mode, no special action is needed. However, in Figure 8-4 (B), when transitioning into full off at the same boundary, an additional emergency dead time is necessary. This inserted dead time is a little different to the normal dead time as it is impossible to move one of the switching events back in time because this would move the event into the previous modulation cycle. Therefore, the entire emergency dead time is inserted by delaying the turn on of the appropriate signal by the full amount.

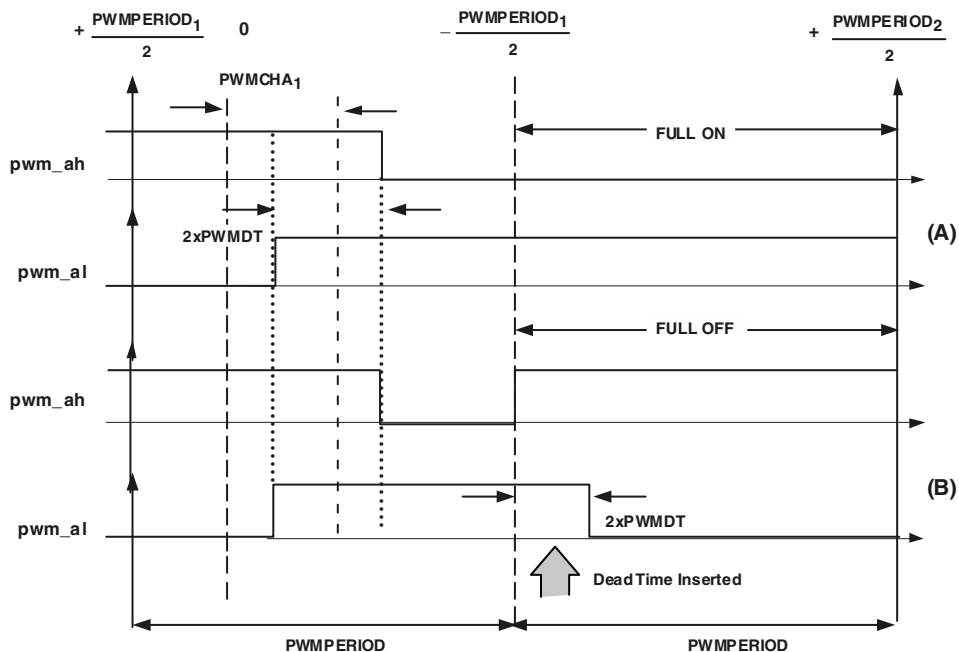


Figure 8-4. Full On to Full Off Transition

Update Modes

Update modes determine the frequency with which the waveforms are sampled.

Single Update

In this mode, duty cycle values are programmable only once per PWM period, so that the resultant PWM patterns are symmetrical about the midpoint of the PWM period.

Double Update

In this mode, a second updating of the PWM registers is implemented at the midpoint of the PWM period. In this mode, it is possible to produce asymmetrical PWM patterns that produce lower harmonic distortion in three-phase PWM inverters. This technique also permits closed-loop controllers to change the average voltage applied to the machine windings at a faster rate, and so permits faster closed-loop bandwidths to be achieved.

Configurable Polarity

The polarity of the generated PWM signals is programmed using the `PWMPOLARITY3-0` registers (see [“PWM Polarity Select Registers \(PWM-POLx\)” on page A-83](#)), so that either active high or active low PWM patterns can be produced. The polarity values can be changed on-the-fly if required, provided the change is done a few cycles before the next period change.

PWM Pins and Signals

The entire PWM module has four groups of four PWM outputs, for a total of 16 PWM outputs. The modules are controlled by the `PWM_AH` and `PWM_BH` pins which produce high side drive signals and the `PWM_AL` and `PWM_BL` pins which produce low side drive signals. These are shown in [Figure 8-1 on page 8-2](#).

Each PWM group is able to generate complementary signals on two outputs in paired mode or each group can provide independent outputs in non-paired mode.

The switching frequency and dead time of the generated PWM patterns are programmable using the `PWMPERIODx` and `PWMDTx` registers. In addition, two duty cycle control registers (`PWMAX` and `PWMBx`) directly control the duty cycles of the two pairs of PWM signals. In non-paired mode the low side signals can have different duty cycles programmed through another pair of registers (`PWMALx` and `PWMBLx`). It should be further noted that the choice of center or edge-aligned mode applies to a single group of four PWM waveforms. Each of the four PWM output signals can be enabled or disabled by separate output enable bits in the `PWMSEG0-3` register (see [“PWM Output Disable Registers \(PWMSEGx\)” on page A-82](#)). Additionally, in center-aligned paired mode, an emergency dead time insertion circuit enforces a dead time defined by the `PWMDT0-3` registers between the high and low side drive signals of each PWM channel. This ensures that the correct dead time occurs at the power inverter. In many applications, there is a need to provide an isolation barrier in the gate-drive circuits that turn on the power devices of the inverter.

Crossover

The `PWMSEG3-0` registers contain four bits, one for each PWM output (see [Table A-28 on page A-82](#)). If crossover mode is enabled for any pair of PWM signals, the high side PWM signal from the timing unit (for example `AH`) is diverted to the associated low side output of the output control

unit so that the signal ultimately appears at the AL pin. The corresponding low side output of the timing unit is also diverted to the complementary high side output of the output control unit so that the signal appears at the AH pin. Following a reset, the three crossover bits are cleared so that the crossover mode is disabled on all three pairs of PWM signals. Even though crossover is considered an output control feature, dead time insertion occurs after crossover transitions as necessary to eliminate shoot through safety issues.

PWM Accuracy

The PWM has 16-bit resolution but accuracy is dependent on the PWM period. In single-update mode, the same values of $PWMA$ and $PWMB$ are used to define the on-times in both half cycles of the PWM period. As a result the effective accuracy of the PWM generation process is $2t_{PCLK}$ (or 20 ns for a 100 MHz clock). Incrementing one of the duty cycle registers by one changes the resultant on-time of the associated PWM signals by t_{PCLK} in each half period (or $2t_{PCLK}$ for the full period). In double-update mode, improved accuracy is possible since different values of the duty cycles registers are used to define the on-times in both the first and second halves of the PWM period. As a result, it is possible to adjust the on-time over the whole period in increments of t_{PCLK} . This corresponds to an effective PWM accuracy of t_{PCLK} in double-update mode (or 10 ns for a 100 MHz clock). The achievable PWM switching frequency at a given PWM accuracy is tabulated in [Table 8-1](#).

Table 8-1. PWM Accuracy in Single and Double-Update Modes

Resolution (bits)	Single-Update Mode PWM Frequency (kHz)	Double-Update Mode PWM Frequency (kHz)
8	195.3	390.6
9	97.7	195.3
10	48.8	97.7

PWM Registers

Table 8-1. PWM Accuracy in Single and Double-Update Modes (Cont'd)

Resolution (bits)	Single-Update Mode PWM Frequency (kHz)	Double-Update Mode PWM Frequency (kHz)
11	24.4	48.8
12	12.2	24.4
13	6.1	12.2
14	3.05	6.1

PWM Registers

The registers described below control the operation and provide the status of pulse width modulation on the ADSP-21367/8/9 and ADSP-2137x processors. [For more information, see “Pulse Width Modulation Registers” on page A-78.](#)

- **PWM global control register.** The `PWMGCTL` register enables or disables the four PWM groups in any combination. This provides synchronization across the four PWM groups. This 16-bit, read/write register is located at address `0x3800`.
- **PWM global status register.** The `PWMGSTAT` register provides the status of each PWM group and is located at address `0x3801`. The bits in this register are W1C-type (write one-to-clear).
- **PWM control registers.** The `PWMCTL3-0` registers are used to set the operating modes of each PWM block. These registers also allow programs to disable interrupts from individual groups.
- **PWM status registers.** The `PWMSTAT3-0` registers are 16-bit read-only registers report the status of the phase and mode for each PWM group.

- **PWM period registers.** The `PWMPERIOD3-0` registers are 16-bit, read/write registers that control the period of the four PWM groups.
- **PWM dead time registers.** The `PWMDT3-0` registers are 16-bit, read/write registers that are used to set the switching dead time.
- **PWM channel A and B duty control registers.** The `PWMA3-0` and `PWMBx` registers directly control the duty cycles of the two pairs of PWM output signals on the `pwm_ah` to `pwm_cl` pins when not in switch reluctance mode.
- **PWM output enable registers.** The `PWMSEG3-0` registers are 16-bit read/write registers that are used to control the output signals of the four PWM groups.
- **PWM channel A and B low side duty control registers.** In non-paired mode, the `PWMAL3-0` and `PWMBL3-0` registers are used to program the low side duty cycle of the two pairs of PWM output signals. These can be different for the high side cycles.
- **PWM output polarity select registers.** The `PWMPOL3-0` registers are 16-bit read/write registers that are used to determine whether the polarity of the generated PWM signals are active high or active low. The polarity values can be changed on-the-fly if required, provided the change is done a few cycles before the next period change.

Duty Cycles

The `PWMAx` and `PWMBx` registers directly control the duty cycles of the two pairs of PWM output signals on the `pwm_ah` to `pwm_cl` pins when not in switch reluctance mode.

PWM Registers

- The two's-complement integer value in the `PWMAx` registers controls the duty cycle of the signals on the `pwm_ah` and `pwm_al` pins.
- The two's-complement integer value in the `PWMBx` registers control the duty cycle of the signals on `pwm_bh` and `pwm_bl` pins.

The duty cycle registers are programmed in two's-complement integer counts of the fundamental time unit, t_{PCLK} , and define the desired on-time of the high side PWM signal produced by the three-phase timing unit over half the PWM period. The duty cycle register range is from

$$(-PWMPERIOD \div 2 - PWMDT) \text{ to } (+PWMPERIOD \div 2 + PWMDT)$$

which, by definition, is scaled such that a value of 0 represents a 50% PWM duty cycle. The switching signals produced by the three-phase timing unit are also adjusted to incorporate the programmed dead time value in the `PWMDT` register. The three-phase timing unit produces active low signals so that a low level corresponds to a command to turn on the associated power device.

Output Enable

The `PWMSEG` register contains six bits (0 to 5) that can be used to individually enable or disable each of the six PWM outputs. If the associated bit of the `PWMSEG` register is set (=1), then the corresponding PWM output is disabled, regardless of the value of the corresponding duty cycle register. This PWM output signal remains disabled as long as the corresponding enable/disable bit of the `PWMSEGx` register is set. Programs should implement this function after the crossover function. After reset, all six enable bits of the `PWMSEG` register are cleared so that all PWM outputs are enabled (default). In the same manner as the duty cycle registers, the `PWMSEG` register is latched on the rising edge of the `pwm_pwmsync_out` signal. Therefore, in single-update mode, changes to this register only become effective at the start of each PWM cycle. In double-update mode, the `PWMSEG` register can also be updated at the midpoint of the PWM cycle.

Programming Example

The following program shows the four steps used to configure a PWM module.

Listing 8-1. Generic PWM Configuration Example

```
#include "def21369.h"
.global start;

/* define for PWM frequency used in PWMPERIOD0 */
#define fPWM 0x1388;      /* 200MHz/2(20kHz) => 50us */
.section/pm_seg_pmco;
start:
call default_int_enable;
call PWM_setup;
call PWM_enables;
nop;
finish: jump finish;
start.end: nop;

/* enable interrupts */
default_int_enable:
    LIRPTL = 0;
    IRPTL = 0;
    bit set MODE1 IRPTEN;      /* Global interrupt enable
    bit set LIRPTL P13IMSK; /* Enable PWM default interrupt -
                                location 13 */
default_int_enable.end: rts;
/* PWM setup registers */
PWM_setup:
```

Programming Example

```
/* 1. Configure frequency */

ustat3=fPWM;          /* fPCLK/2xfPWM */
dm(PWMPERIOD0)=ustat3; /* PWM Period Register for switching
                        frequency (unsigned integer) */

/* 2. Configure duty cycles Width=[period/2] + duty program in
the two's-complement of the high side width for individual con-
trol this only programs AH signal. If PWMAL0 is not programmed
then the AL and AH signals have the same duty cycle */

ustat3=0;             /* PWM Channel A Duty Control
                        (two's-complement integer) */

dm(PWMA0)=ustat3;     /* Set up the duty cycle register to 0
                        (50% duty cycle) */

ustat3 = 0x63C;       /* two's-complement of 0x9C4 = 0x63C - 80%
                        high - 20% low */

dm(PWMAL0)=ustat3; /* PWM Channel AL Duty Control */

/* 3. Configure Dead Time */

ustat3=0x0; /* PWM Dead Time Register (unsigned integer) */
dm(PWMDT0)=ustat3;

/* 4. Configure Polarity (this can be changed on the fly
after the PWM port is enabled) */
ustat3=0; /* PWM Polarity Select Register */

bit set ustat3 PWM_POL1AL | PWM_POL1AH; /* Enables high polarity
A output */

dm(PWMPOL0)=ustat3;
PWM_setup.end: nop;
```

```
/* PWM enables */

PWM_enables:
ustat3=dm(SYSCTL); /* System Control Register */
bit set ustat3 PWM0EN | PPFLGS;

dm(SYSCTL)=ustat3; /* Selects AD11-8 in PWM0 mode instead
                  of PP mode */

ustat3=dm(PWMSEG0); /* PWM Output Enable. Should probably be
                  changed to PWM Output Disable since you
                  write it to disable it. */

bit set ustat3 PWM_BH | PWM_BL; /* disables B outputs */
dm(PWMSEG0)=ustat3;
ustat3=dm(PWMCTL0); /* PWM0 Control Register
ustat3=0;
dm(PWMCTL0)=ustat3; /* Enables edge-aligned, individual pair
                  mode with single update and no
                  interrupt */

ustat3=dm(PWMGCTL); /* PWM General Control Register */

bit set ustat3 PWM_EN0 | PWM_DIS1 | PWM_DIS2 | PWM_DIS3 |
PWM_SYNCEN0 | PWM_SYNCDIS1 | PWM_SYNCDIS2 | PWM_SYNCDIS3;

dm(PWMGCTL)=ustat3; /* Enables only PWM 0 and its internal
                  timer; Disables other PWMs globally. The
                  write to PWMGCTL will kick off the
                  transfer */

PWM_enables.end: rts;

idle;
```

Programming Example

9 S/PDIF

TRANSMITTER/RECEIVER

S/PDIF (Sony/Philips Digital Interface) is a standard audio data transfer format that allows the transfer of digital audio signals from one device to another without having to be converted to an analog signal. The ADSP-21367/8/9 and ADSP-2137x processors have AES3-compliant S/PDIF receivers/transmitters that allow programs to interface to other S/PDIF devices.

This chapter provides information on the function of the S/PDIF module in the ADSP-21367/8/9 and ADSP-2137x SHARC processors. It is important to be familiar with the serial digital audio interface standards IEC-60958, EIAJ CP-340, AES3, and AES11.

The S/PDIF receiver and transmitter reside in the digital audio interface (DAI). This allows applications to use the serial ports and/or the external DAI pins to interface to other S/PDIF devices. This can include using the receiver to decode incoming bi-phase encoded audio streams and passing them through the SPORTs to internal memory for processing or using the transmitter to encode audio or digital data and transfer it to another S/PDIF receiver in the audio system. Other features include:

- Managing user status information and providing error-handling capabilities in both the receiver and transmitter.
- Transmitting a bi-phase encoded signal that may contain any number of audio channels (compressed or linear PCM) or encoded data streams.

AES3/SPDIF Stream Format

- In regards to handling channel status information, the SPDIF transmitter can operate in standalone and full serial modes. In standalone mode, channel status bits, user bits, and validity bits are taken from the corresponding buffers or control register. In full serial mode, all these bits are transferred with data bits from the SDATA pin. This allows programs to have control over the status fields and the ability to pass on status information which may have been supplied from an encoder, for instance.

AES3/SPDIF Stream Format

The data carried by the SPDIF interface is transmitted serially. In order to identify the assorted bits of information the data stream is divided into frames, each of which are 64 time slots (or 128 unit intervals¹) in length (Figure 9-1). Since the time slots correspond with the data bits, the frame is often described as being 64 bits in length.

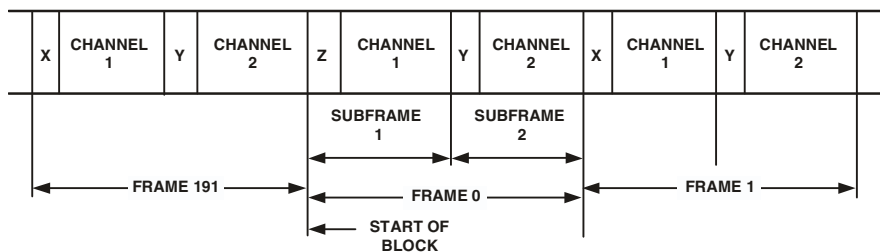


Figure 9-1. S/PDIF Block Structure

¹ The unit interval is the minimum time interval between condition changes of a data transmission signal.

A frame is uniquely composed of two subframes. The first subframe normally starts with preamble X. However, the preamble changes to preamble Z once every 192 frames. This defines the block of frames structure used to organize the channel status information. The second subframe always starts with preamble Y.

Subframe Format

Each frame consists of two subframes. [Figure 9-2](#) shows an illustration of a subframe, which consists of 32 time slots numbered 0 to 31. A subframe is 64 unit intervals in length. The first four time slots of each subframe carry the preamble information. The preamble marks the subframe start and identifies the subframe type. The next 24 time slots carry the audio sample data, which is transmitted in a 24-bit word with the least significant bit (LSB) first. When a 20-bit coding range is sufficient, time slots 8 to 27 carry the audio sample word with the LSB in time slot 8. Time slots 4 to 7 may be used for other applications. Under these circumstances, the bits in time slots 4 to 7 are designated auxiliary sample bits. If the source provides fewer bits than the interface allows (either 20 or 24), the unused LSBs are set to logic 0.

This functionality is important when using the SPDIF receiver in common applications where there are multiple types of data to handle. If there are PCM audio data streams as well as encoded data streams, for example a CD audio stream and a DVD audio stream with encoded data, there is a danger of incorrectly passing the encoded data directly to the DAC. This results in the ‘playing’ of encoded data as audio, causing loud odd noises to be played. The non-audio flag provides an easy method to mark the this type of data.

After the audio sample word, there are four final time slots which carry:

1. **Validity bit (time slot 28).** The validity bit is logic 0 if the audio sample word is suitable for conversion to an analog audio signal, and logic 1 if it is not. This bit is set if the `CHST_BUF_ENABLE` bit

AES3/SPDIF Stream Format

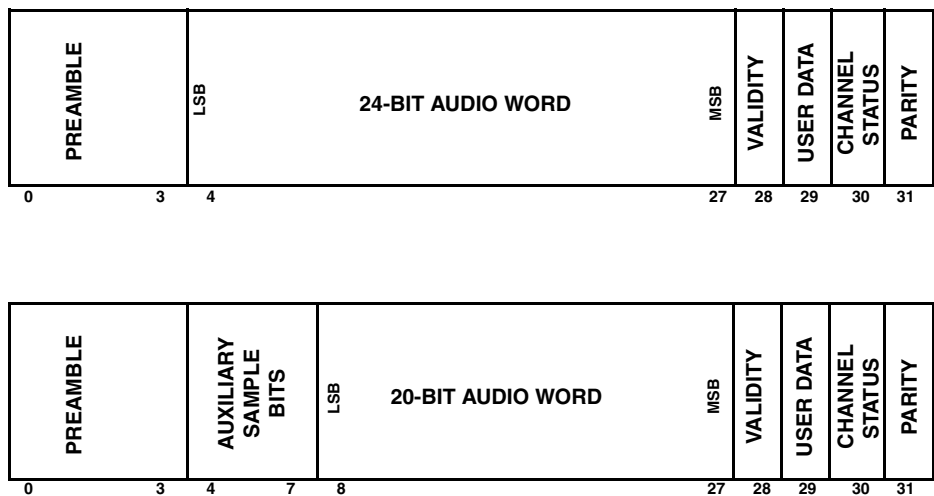


Figure 9-2. Subframe Format

and the `VALIDITY_A` (`VALIDITY_B` for channel 2) bit is set in the `SPDIF_TX_CTL` register. This bit is also set if the corresponding bit given with the sample is set.

2. **User data bit (time slot 29).** This bit carries user-specified information that may be used in any way. This bit is set if the corresponding bit given with the left/right sample is set.
3. **Channel status bit (time slot 30).** The channel status for each audio signal carries information associated with that audio signal, making it possible for different channel status data to be carried in the two subframes of the digital audio signal. Examples of information to be carried in the channel status are: length of audio sample words, number of audio channels, sampling frequency, sample address code, alphanumeric source and destination codes, and emphasis.

Channel status information is organized in 192-bit blocks, subdivided into 24 bytes. The first bit of each block is carried in the frame with preamble Z.

For convenience, the first five bytes of the channel status may be written all at once to control registers for both channels A and B (SPDIF_TX_CTL, SPDIF_TX_CHSTA, and SPDIF_TX_CHSTB). If the CHST_BUF_ENABLE bit is set in the SPDIF_TX_CTL register, the appropriate CS bit is ORed into the channel status bit of the 192-word frame. In addition, the CS bit can also be provided along with the data samples to be transmitted.

4. **Parity bit (time slot 31).** The parity bit indicates that time slots 4 to 31 inclusive will carry an even number of ones and an even number of zeros (even parity). The parity bit is automatically generated for each subframe and inserted into the encoded data.

The two subframes in a frame can be used to transmit two channels of data (channel 1 in subframe 1, channel 2 in subframe 2) with a sample rate equal to the frame rate. Alternatively, the two subframes can carry successive samples of the same channel of data, but at a sample rate that is twice the frame rate. This is called single-channel, double-frequency (SCDF). [For more information, see “Single-Channel, Double-Sampling Frequency Mode” on page 9-21.](#)

Channel Coding

To minimize the direct-current (dc) component on the transmission line, to facilitate clock recovery from the data stream, and to make the interface insensitive to the polarity of connections, time slots 4 to 31 are encoded in bi-phase mark.

AES3/SPDIF Stream Format

Each bit to be transmitted is represented by a symbol comprising two consecutive binary states. The first state of a symbol is always different from the second state of the previous symbol. The second state of the symbol is identical to the first if the bit to be transmitted is logic 0. However, it is different if the bit is logic 1.

Figure 9-3 shows that the ones in the original data end up with mid cell transitions in the bi-phase mark encoded data, while zeros in the original data do not. Note that the bi-phase mark encoded data always has a transition between bit boundaries.

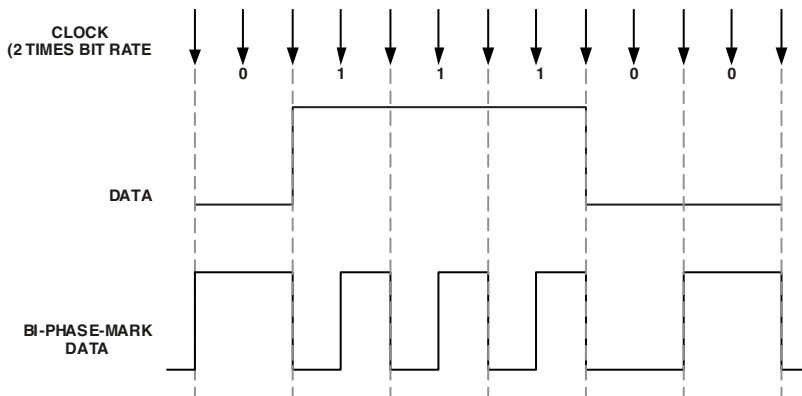


Figure 9-3. Bi-phase Mark Encoding

Preambles

Preambles are specific patterns that provide synchronization and identify the subframes and blocks. To achieve synchronization within one sampling period and to make this process completely reliable, these patterns violate the bi-phase mark code rules, thereby avoiding the possibility of data imitating the preambles.

A set of three preambles, shown in [Table 9-1](#), are used. These preambles are transmitted in the time allocated to four time slots at the start of each subframe (time slots 0 to 3) and are represented by eight successive states. The first state of the preamble is always different from the second state of the previous symbol (representing the parity bit).

Table 9-1. Preambles

Preamble	Preceding state 0	Preceding state 1	Description
X	11100010	00011101	Subframe 1
Y	11100100	00011011	Subframe 2
Z	11101000	00010111	Subframe 1 and block start

Like bi-phase code, the preambles are dc free and provide clock recovery. They differ in at least two states from any valid bi-phase sequence.

S/PDIF Transmitter

The S/PDIF transmitter resides within the DAI, and its inputs and outputs can be routed through the signal routing unit (SRU1). It receives audio data in serial format, encloses the specified user status information, and converts it into the bi-phase encoded signal. The serial data input to the transmitter can be formatted as left-justified, I²S, or right-justified with word widths of 16, 18, 20 or 24 bits.

The serial data, clock, external sync signal, and frame sync inputs to the S/PDIF transmitter are routed through SRU1. They can come from a variety of sources such as the SPORTs, external pins, the precision clock generators (PCG), or the sample rate converters (SRC). The signal routing is selected in the SRU1 control registers. [For more information, see “DAI/SRU1 Connection Groups” on page 4-18.](#)

S/PDIF Transmitter

The S/PDIF transmitter output may be routed to an output pin through SRU1 and then routed to another S/PDIF receiver or to components for off-board connections to other S/PDIF receivers. The output is also available to the S/PDIF receiver for loop-back testing through SRU1.

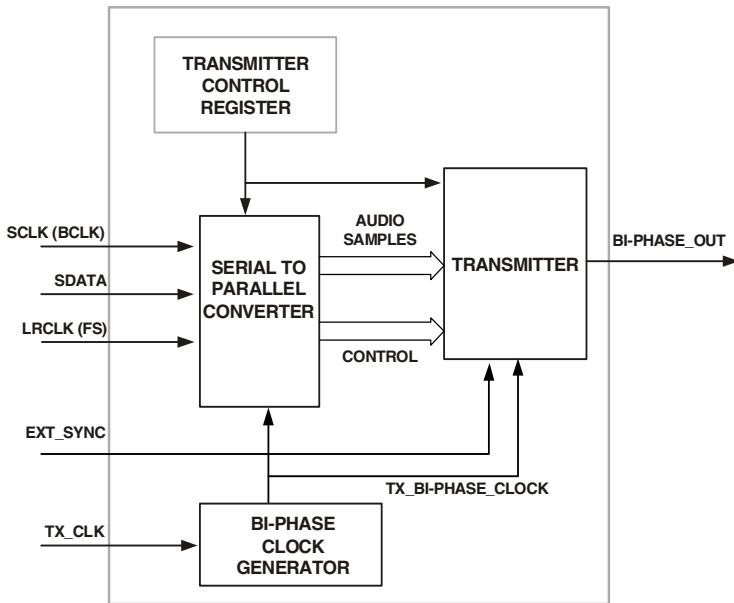


Figure 9-4. S/PDIF Transmitter Block Diagram

Two output data formats are supported by the transmitter: *two channel mode* and *single channel double frequency* (SCDF) mode. The output format is determined by the transmitter control register (DITCTL). [For more information, see “Transmitter Control Register \(DITCTL\)” on page A-86.](#)

In two channel mode, the left channel (channel A) is transmitted when the LRCLK is high and the right channel (channel B) is transmitted when the LRCLK is low.

In SCDF mode, the transmitter sends successive audio samples of the same signal across both subframes, instead of channel A and B. This mode also allows programs to select which channel is sent when using bits in the S/PDIF transmit control register. The channel status bits are set to provide the downstream receiver with information about which channel is used.

Channel Status

In addition to encoding the audio data in the bi-phase format, the transmitter also provides a way to easily add the channel status information to the outgoing bi-phase stream. There are status registers in the transmitter that correspond to each channel or subframe. Byte 0 for each channel A and B reside in the `DITCTL` register, and bytes 1–4 reside in the `DITCHANxx` registers. [For more information, see “Transmitter Control Register \(DITCTL\)” on page A-86, “Left Channel Status for Subframe A Registers \(DITCHANAx\)” on page A-89 and “Right Channel Status for Subframe B Registers \(DITCHANBx\)” on page A-90.](#)

The first five bytes of the channel status may be written all at once to the control registers for both A and B channels. As the data is serialized and transmitted, the appropriate bit is inserted into the channel status area of the 192-word frame. Validity bits for both channels may also be controlled by the transmitter control register. Optionally, the user bit, validity bit, and channel status bit are sent to the transmitter with each left/right sample. For each subframe the parity bit is automatically generated and inserted into the bi-phase encoded data. A mute control and support for double-frequency single-channel mode are also provided. The serial data input format may be selected as left-justified, I^2S , or right-justified with 16-, 18-, 20- or 24-bit word widths. The over sampling clock is also selected by the transmitter control register.

SRU1 Signals for the S/PDIF Transmitter

To use the transmitter, route the five required inputs using SRU1 as described below. Also, use SRU1 to connect the two outputs, bi-phase encoded output, and block start to the desired DAI (digital audio interface) pin.

DIT_CLK_I is the serial clock. It controls the rate at which serial data enters the S/PDIF module. This is typically 64 time slots¹. The **SCLK** input to the S/PDIF transmitter is controlled by the 5-bit clock routing (SRU1 group A) register field **SRU_CLK2[14:10]** (**DIT_CLK_I**). This clock input can come from the SPORTs, the PCG, external pins, or from the S/PDIF receiver. By default it is connected to **LOGIC_LEVEL_LOW**. [For more information, see “Group A Connections—Clock Signals” on page 4-19.](#)

DIT_DAT_I provides serial data. The format of the serial data can be I²S and right/left-justified. The **SDATA** input to the S/PDIF transmitter is controlled by the 6-bit serial data routing (SRU1 group B) register field **SRU_DAT4[5:0]** (**DIT_DAT_I**). The data input can come from the SPORTs, the SRC, external pins, or from the S/PDIF receiver. By default, **SDATA** is connected to external pin 0. [For more information, see “Group B Connections—Data Signals” on page 4-25.](#)

DIT_FS_I is the frame sync input to the S/PDIF transmitter. It is controlled by the 5-bit **LRCLK** routing (SRU1 group C) register field **SRU_FS2[14:10]**. The frame sync input can come from the SPORTs, the PCGs, external pins, or from the S/PDIF receiver. By default the frame sync is connected to **LOGIC_LEVEL_LOW**. [For more information, see “Group C Connections—Frame Sync Signals” on page 4-31.](#)

¹ Timing for the S/PDIF format consists of time slots, unit intervals, subframes and frames. For a complete explanation of S/PDIF timing, see one of the digital audio interface standards listed at the beginning of this chapter.

DIT_HFCLK_I is the oversampling clock. This clock is divided down according to the **FREQMULT** bit in the transmitter control register to generate the bi-phase clock. It can also be selected from various sources since it is routed through **SRU1**. The **TX_CLK** input to the S/PDIF transmitter is controlled by the 5-bit clock routing (**SRU1** group A) register field **SRU_CLK3[29:25]** (**DIT_HFCLK_I**). This clock input can come from the **SPORTS**, the **PCG**, external pins, or from the S/PDIF receiver. By default **TX_CLK** is connected to **LOGIC_LEVEL_LOW**. [For more information, see “Group A Connections—Clock Signals” on page 4-19.](#)

DIT_EXTSYNC_I is a clock input to the SPDIF transmitter and is controlled by the 5-bit clock routing (**SRU1** group A) register field **SRU_CLK4[19:15]** (**SPDIF_TX_EXT_SYNC**). This clock can come from any of 20 external DAI pins. By default it is connected to **LOGIC_LEVEL_LOW**.

DIT_O is the bi-phase encoded data stream. It can be routed to any of the external pins or to the S/PDIF receiver for loop-back testing through **SRU1**. The **SRU_PINx** registers control this routing. [For more information, see “Group D Connections—Pin Signal Assignments” on page 4-36.](#)

BLK_START indicates the last frame of the current block. This is high for the entire duration of the frame. The **BLK_START** output can be routed to any of the external pins controlled by the **SRU_PINx** registers. This can also be connected to the DAI interrupts [31–22] using the **SRU_MISCx** registers. Use of this signal is optional.

S/PDIF Transmitter Registers

The SPDIF transmitter contains registers that are used to enable/disable the transmitter, to manage its operation, and to report status. The registers are described below.

- **DITCTL** is the S/PDIF transmit control register. This 32-bit read/write register is located at address 0x24A0. It is used to enable the transmitter, control mute, over sampling, mode and data format. This register is described in detail in [“Transmitter Control Register \(DITCTL\)” on page A-86](#).
- **DITCHANAx** and **DITCHANBx** are the S/PDIF channel A and B transmit status registers. These 32-bit read/write registers, located at addresses 0x24A1 and 0x24A2, provide status information for transmitter subframe A and B. These registers are described in detail in [“Left Channel Status for Subframe A Registers \(DITCHANAx\)” on page A-89](#) and [“Right Channel Status for Subframe B Registers \(DITCHANBx\)” on page A-90](#). These registers are used in standalone mode only.
- **DITUSRBITAx** and **DITUSRBITBx** are the user bit buffers. Once programmed, they are used only for the next block of data. This allows programs to change the user bit information with every block of data. After writing to the appropriate registers to change the user bits for the next block, **DITUSRBITAx** and **DITUSRBITBx** must be written to enable the use of these bits. These registers are used in standalone mode only.

Modes of Operation

The SPDIF transmitter can operate in standalone and full serial modes. The following sections describe these modes in detail.

Standalone Mode

This mode is selected by setting bit 9 in the `DITCTL` register. In this mode, the block start bit (indicating start of a frame) is generated internally. The channel status bits come from the channel status buffer registers (`DITCHANAx` and `DITCHANBx`). The user status bits come from the user bits buffers (`DITUSRBITAx` and `DITUSRBITBx`). The channel status buffer must be programmed before the SPDIF transmitter is enabled and used for all the successive blocks of data.

Once the user bits buffer registers (`DITUSRBITA0-5` and `DITUSRBITB0-5`) are programmed, they are used only for the next block of data. This allows programs to change the user bit information in every block of data. After writing to the required user bit buffer registers to change the user bits for the next block, these registers must be rewritten to enable the use of these bits in the next block. The validity bit for channel A and B are taken from bit 10 and bit 11 of the `DITCTL` register. In this mode only audio data comes from the `SDATA` pin. All other data, including the status bit and block start bit is either generated internally or taken from the internal register.

Full Serial Mode

This mode is selected by clearing bit 9 in the `DITCTL` register. In this mode all the status bits, audio data and the block start bit (indicating start of a frame), come through the `SDATA` pin. The transmitter should be enabled after or at the same time as all of the other control bits.

Structure of the Serial Input Data

Figure 9-5 shows the format of data that is sent to the S/PDIF transmitter using a 24-bit I²S interface. The upper 24 bits (bits 8 through 31) contain the audio data. Bits 3–7 are used to transmit status information and to generate preambles and or headers. Bits 0 through 3 are empty.

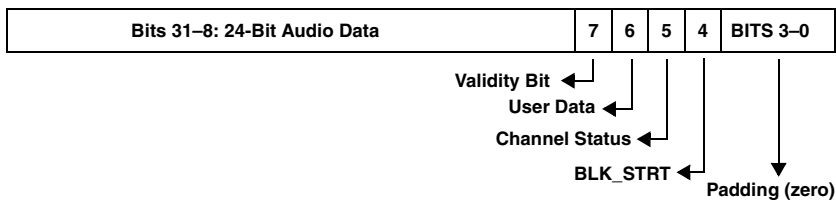


Figure 9-5. Data Packing for I²S and Left-Justified Format



When I²S format is used with 20-bit or 16-bit data, the audio data should be placed from the MSB of the 24-bit audio data.

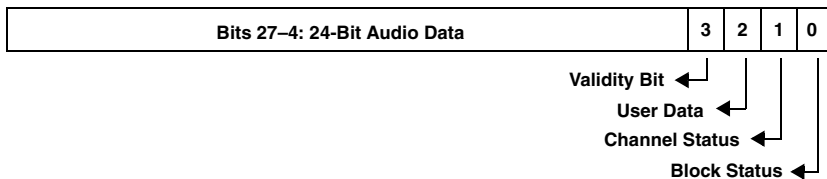


Figure 9-6. Data Packing for Right-Justified Format, 24 Bits

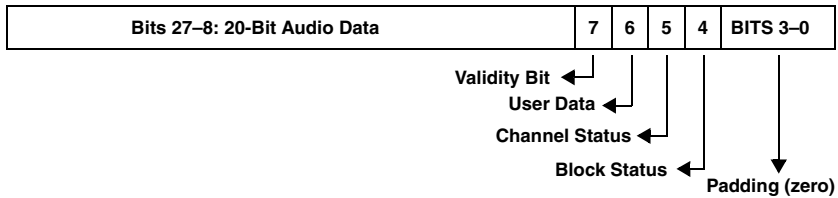


Figure 9-7. Data Packing for Right-Justified Format, 20 Bits

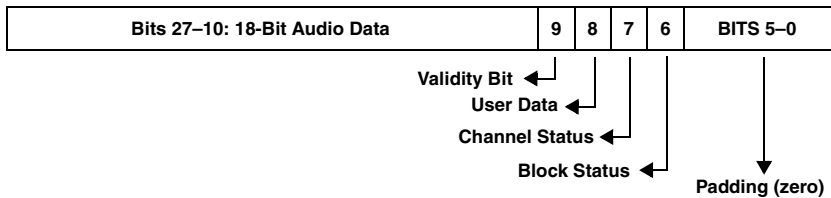


Figure 9-8. Data Packing for Right-Justified Format, 18 Bits

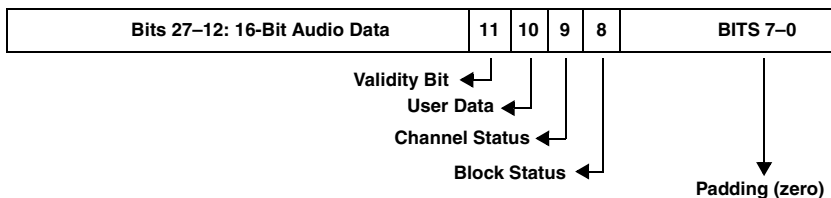


Figure 9-9. Data Packing for Right-Justified Format, 16 Bits

S/PDIF Receiver

The S/PDIF receiver is compliant with all common serial digital audio interface standards including IEC-60958, IEC-61937, AES3, and AES11. These standards define a group of protocols that are commonly associated with the S/PDIF interface standard defined by AES3, which was developed and is maintained by the Audio Engineering Society. AES3 effectively defines the data and status bit structure of an S/PDIF stream. AES3-compliant data is sometimes referred to as AES/EBU compliant. This term highlights the adoption of the AES3 standard by the European Broadcasting Union. The S/PDIF receiver in the ADSP-21367/8/9 and ADSP-2137x processors receives an S/PDIF bi-phase encoded stream and decodes it into an I²S serial data format, and provides the programmer with several methods of managing the incoming status bit information.

The input to the receiver is a bi-phase encoded signal that may contain two audio channels (compressed or linear PCM) or non-audio data. The receiver decodes the single bi-phase encoded stream, producing an I²S-compatible serial data output that consists of a serial clock (SCLK), a left/right clock (FS), and data (channel A/B).

The receiver can recover the clock from the bi-phase encoded stream using either a dedicated on-chip digital phased-locked loop (PLL) or an external analog PLL. (The dedicated on-chip digital PLL is separate from the PLL that supplies the core clock to the ADSP-213xx processor core.)

The S/PDIF receiver input stream can be selected from any of the DAI pins, DAI_P20-1, or from the output of the S/PDIF transmitter for loop-back testing using signal routing group C. The first five bytes of the channel status are identified and stored in dedicated status registers for both A and B channels. The registers are DIR_CHANL and DIR_CHANR. As the serial data is received, the appropriate bits (first five bytes – bit 0 through bit 39) are updated from the 192-word frame. If the channel status bits change, an interrupt may optionally be generated to notify the core.

The receiver also detects errors in the S/PDIF stream. These error bits are stored in the status register, which can be read by the core. Optionally, an interrupt may be generated to notify the core on error conditions. The extracted serial data is transmitted on the data pin in I²S format. The extracted clock, frame sync, and data are routed through SRU1.

The S/PDIF receiver receives any S/PDIF stream with a sampling frequency range of 32 kHz – 15% to 192 kHz + 15% range.

The serial output from the receiver is 24-bit I²S serial data which conforms to the format shown in [Figure 9-5 on page 9-14](#).

S/PDIF Receiver Registers

The S/PDIF receiver uses the registers described in the following sections. More detail can be found in [“Sony/Philips Digital Interface Registers” on page A-86](#).

DIRCTL is the receiver control register. It is a 32-bit, read/write register located at address 0x24A8 and is used to enable the receiver, control mute, PLL and SCDF mode. This register is described in detail in [“Receiver Control Register \(DIRCTL\)” on page A-92](#).

DIRSTAT is the receiver status register. It is a 32-bit, read-only register that is used to store the error bits. The error bits are sticky on read. Once they are set, they remain set until the register is read. This register also contains the lower byte of the 40-bit channel status information.

DIRCHANL, **DIRCHANR** are the channel status for the subframes registers. They provide status information for subframe A (left channel) bytes 1, 2, 3, and 4 and subframe B (right channel) bytes 1, 2, 3, and 4. These 32-bit, read/write registers are located at address 0x24AA and 0x24AB. See also [“Left Channel Status for Subframe A Registers \(DITCHANAx\)” on page A-89](#) and [“Right Channel Status for Subframe B Registers \(DITCHANBx\)” on page A-90](#).

SRU1 Receiver Signals

The bi-phase encoded data and the external PLL clock inputs to the receiver are routed through the signal routing unit (SRU1). The extracted clock, frame sync, and data are also routed through SRU1.

The SRU1 inputs to the S/PDIF receiver are configured through the following signals.

- **SPDIF_PLLCLK_I** is the external 512 x FS (frame sync) PLL clock input. This signal is controlled by the 5-bit clock routing register field `SRU_CLK4[14:10]` (Figure 4-16 on page 4-22). This clock input can come from the external pins connected to the external PLL.
- **DIR_I** is the bi-phase encoded data input. This signal is controlled by the 5-bit frame sync routing register field `SRU_FS3[29:25]` (Figure 4-16 on page 4-22). This data may be received from external pins or from the S/PDIF receiver.

The SRU1 outputs from the S/PDIF receiver are configured through the following signals.

- **DIR_DAT_O** is the extracted audio data output. This signal can be routed to any of the external pins or to one of the serial receivers (SPORT, input data port) through SRU1.
- **DIR_CLK_O** is the extracted receiver sample clock output. This signal can be routed to any of the external pins or to one of the serial receivers (SPORT, input data port) through SRU1.
- **DIR_FS_O** is the extracted receiver frame sync out. This signal can be routed to any of the external pins or to one of the serial receivers (SPORT, input data port) through SRU1.

- **DIR_TDMCLK_O** is the receiver TDM clock output. This signal can be routed to any of the external pins or to one of the serial receivers (SPORT, input data port) through SRU1.
- **DIR_LRCLK_FB** is the external PLL feedback point connection.
- **DIR_LRCLK_REF** is the external PLL reference point connection.

Phase-Locked Loop

The phase-locked loop for the AES3/SPDIF receiver is intended to recover the clock that generated the AES3/SPDIF bi-phase encoded stream. This clock is used by the receiver to clock in the bi-phase encoded data stream and also to provide clocks for either the serial ports, sample rate converter, or AES3/SPDIF transmitter. The recovered clock may also be used externally to the chip for clocking D/A and A/D converters.

In order to maintain performance, jitter on the clock is sourced to several peripherals. Jitter on the recovered clock must be less than 200 ps and, if possible, less than 100 ps across all the sampling frequencies ranging from 27.2 kHz to 220.8 kHz (32 kHz – 15% and 192 kHz + 15%). Furthermore, once the PLL achieves lock, it is able to vary $\pm 15\%$ in frequency over time. This allows for applications that do not use PLL unlocking.

The receiver can be used with the on-chip digital PLL or with an external analog PLL. There are various performance characteristics to consider when configuring for analog PLL mode, and more information can be found on the Analog Devices Web site.

Channel Status Decoding

The S/PDIF receiver processes compressed as well as non-linear audio data according to the supported standards. The following sections describe how the S/PDIF receiver handles different data.

Compressed or Non-Linear Audio Data

The AES3/SPDIF receiver is required to detect compressed or non-linear audio data according to the AES3, IEC60958, and IEC61937 standards. Bit 1 of byte 0 in the `DIR_BOCHAN` register indicates whether the audio data is linear PCM, (bit 1 = 0), or non-PCM audio, (bit 1 = 1). If the channel status indicates non-PCM audio, the `DIR_NOAUDIO` bit flag is set. (This bit can be used to generate an interrupt.) The `DIR_VALID` bit (bit 3 in the `DIRSTAT` register) when set (=1) may indicate non-linear audio data as well. Whenever this bit is set, the `VALIDITY` bit flag in the `SPDIF_RX_STAT` register is also set.

The MPEG-2, AC-3, DTS, and AAC compressed data may be transmitted without setting either the `DIR_VALID` bit or bit 1 of byte 0. To detect this data, the IEC61937 and SPMTE 337M standards dictate that there be a 96-bit sync code in the 16-, 20- or 24-bit audio data stream. This sync code consists of four words of zeros followed by a word consisting of 0xF872 and another word consisting of 0x4E1F. When this sync code is detected, the `DIR_NOAUDIO` bit flag is set. If the sync code is not detected again within 4096 frames, the `DIR_NOAUDIO` bit flag is deasserted.

The last two words of the sync code, 0xF872 and 0x4E1F, are called the preamble-A and preamble-B of the burst preamble. Preamble-C of the burst preamble contains burst information and is captured and stored by the receiver. Preamble-D of the burst preamble contains the length code and is captured by the receiver. Even if the validity bit or bit 1 of byte 0 has been set, the receiver still looks for the sync code in order to record the preamble-C and D values. Once the sync code has not been detected in 4096 frames, the preamble-C and D registers are set to zero.



The SPDIF receiver supports the DTS stream. The DTS specifications support frame sizes of 256, 512, 1024, 2048 and 4096. The on-chip SPDIF receiver supports the 256, 512 and 1024 DTS frames. The DTS test kit frames with 2048 and 4096 frame sizes

can be detected by adding the sync detection logic in software by using a software counter to check for the DTS header every 2048 and 4096 frames respectively.

Emphasized Audio Data

The receiver must indicate to the program whether the received audio data is emphasized using the channel status bits as detailed below.

- In professional mode, (bit 0 of byte 0 = 1), channel status bits 2–4 of byte 0 indicate the audio data is emphasized if they are equal to 110 or 111.
- In consumer mode, (bit 0 of byte 0 = 0), channel status bits 3–5 indicate the audio data is emphasized if they are equal to 100, 010 or 110.

If emphasis is indicated in the channel status bits, the receiver asserts the `EMPHASIS` bit flag. This bit flag is used to generate an interrupt as shown in [“Error Handling” on page 9-22](#).

Single-Channel, Double-Sampling Frequency Mode

Single-channel, double-frequency mode (SCDF) is selected with the `DIR_SCDF` and `DIR_SCDF_LR` bits in the `DIRCTL` register. The `DIR_BOCHANL/R` bits in the `DIRSTAT` register also contain information about the SCDF mode. When the `DIR_BOCHANL/R` indicates single-channel, double-frequency mode, the two subframes of a frame carry successive audio samples of the same signal.

Error Handling

Bits 0–3 of channel status byte 1 are decoded by the receiver to determine one of the following:

- 0111 = single-channel, double-frequency mode
- 1000 = single-channel, double-frequency mode – stereo left
- 1001 = single-channel, double-frequency mode – stereo right

Error Handling

The following five types of errors can occur in the receiver and are reported on the error flag bits.

1. **Lock Error.** When bit 4 in the `DIRSTAT` register is set (=1), the PLL is locked.
2. **Bi-phase Error.** When bit 7 in the `DIRSTAT` register is set (=1), it indicates that a bi-phase error has occurred and the data sampled from the bi-phase stream may not be correct.
3. **Parity Error.** When bit 6 in the `DIRSTAT` register is set (=1), it indicates that the AES3/SPDIF stream was received with the correct even parity. When the `DIR_PARITYERROR` bit is low (=0), it indicates that an error has occurred and the parity is odd.
4. **CRCC Error.** The `CRCCERROR` bit is asserted high whenever the CRCC check of the `DIR_BOCHANL/R` bits fails. The CRCC check is only performed if the channel status bit 0 of byte 0 is high, indicating professional mode.
5. **No Stream Error.** The `DIR_NOSTREAM` bit is asserted whenever the AES3/SPDIF stream is disconnected.

When the `DIR_NOSTREAM` bit is asserted and the audio data in the stream is linear PCM, the receiver performs a soft mute of the last valid sample from the AES3/SPDIF stream. A soft mute consists of taking the last valid

audio sample and slowly and linearly decrementing it to zero, over a period of 4096 frames. During this time, the PLL three-states the charge pump until the soft mute has been completed. If non-linear PCM audio data is in the AES3/SPDIF stream when the `NOSTREAM` bit is asserted, the receiver sends out zeros after the last valid sample.

When the `DIR_LOCK` bit is deasserted, it means that the PLL has become unlocked and the audio data is handled according to the `DIR_NOAUDIO[1:0]` bits in the `DIRCTL` register. When this happens, the receiver functions as follows.

- 00 = no action is taken with the audio data.
- 01 = the last valid audio sample is held.
- 10 = zeros are sent out after the last valid sample.
- 11 = soft mute of the last valid audio sample is performed (as if `DIR_NOSTREAM` is asserted).

This is valid only when linear PCM audio data is in the stream. When non-linear audio data is in the stream, this mode defaults to the case of `DIR_NOAUDIO[1:0]` bits = 10.

When a parity or bi-phase error occurs, the audio data is handled according to the `DIR_BIPHASEERROR_CTL[1:0]` bits in the following manner.

- 00 = no action is taken with the audio data.
- 01 = the last valid sample is held.
- 10 = the invalid sample is replaced with zeros.

The `VALIDITY`, `NONAUDIO`, `NOSTREAM`, `BIPHERR`, `PARITY` and `LOCK` bits are also stored in the receiver status register as `W1C` bits.

Interrupts

The following error/status bits can be used to interrupt the processor core.

- The `DIR_LOCK`, `DIR_VALID`, `DIR_NOSTREAM` and `DIR_NOAUDIO` bits can generate interrupts. Parity errors and bi-phase errors are ORed together to form a `PARITY_BIPHASE_ERROR` interrupt. Whenever there is a change in channel status information, a `CHANNEL_STAT_CHANGE` interrupt occurs.
- The `DIR_BIPHASEERROR` and `DIR_VALID` interrupts are one `PCLK` pulse wide.
- All interrupts are processed through the interrupt controller which can generate an interrupt on the rising or falling edge of the signal.

DAI Programming Examples

The following examples show how the S/PDIF receiver and transmitter are programmed using the digital audio interface/SRU1.

S/PDIF Transmitter Programming Guidelines

The following guidelines are intended to help in programming the S/PDIF transmitter.

Control Register

The `DITCTL` register contains control parameters for the S/PDIF transmitter. The control parameters include transmitter enable, mute information, oversampling clock division ratio, SCDF mode select and enable, serial data input format select and validity, and channel status buffer selects. By default, all the bits in this register are zero.

SRU1 Programming for Input and Output Streams

Signal routing unit 1 (SRU1) is used to connect the S/PDIF transmitter bi-phase data out to the output pins or to the S/PDIF receiver. The serial data input and the over sampling clock input also needs to be routed through SRU1. See [For more information, see “Group A Connections—Clock Signals” on page 4-19](#) and [“Group B Connections—Data Signals” on page 4-25](#).

Control Register Programming and Enable

After SRU1 programming is complete, if the channel status or validity buffer needs enabling, write to the buffers first with the required data and then enable the buffers using the `DIT_CHANBUF` bit in the `DITCTL` register. Also write other control values such as `DIT_SMODEIN`, and `DIT_FREQ` and enable the transmitter by setting the `DIT_EN` bit.

S/PDIF Receiver Programming Guidelines

The following guidelines are intended to help in programming the S/PDIF receiver.

Control Register

The S/PDIF receiver is enabled at default to receive in two channel mode. Therefore, if the receiver is not used, programs should disable the digital PLL to avoid unnecessary switching. This is accomplished by writing into the `DIR_PLLDIS` bit (bit 7) in the `DIRCTL` register. In most cases, when the S/PDIF receiver is used, this register does not need to be changed. For a detailed description of this register, see [“Receiver Control Register \(DIRCTL\)” on page A-92](#).

The `DIRCTL` register contains control parameters for the S/PDIF receiver. The control parameters include mute information, error controls, SCDF mode select and enable, and digital PLL disable.

SRU1 Programming

The SRU1 needs to be programmed in order to connect the S/PDIF receiver to the output pins or any other peripherals and also for the input bi-phase stream.

Program the corresponding SRU1 registers to connect the above outputs to the required destination. See [“SRU1 Receiver Signals” on page 9-18](#), [“Group A Connections—Clock Signals” on page 4-19](#), and [“Group C Connections—Frame Sync Signals” on page 4-31](#).

Control Register Programming

After SRU1 programming is complete, write to the `SPDIF_RX_CTL` register with control values, and enable the internal digital PLL by clearing the `DIR_PLLDIS` bit if it was cleared initially. At this point, the receiver attempts to lock.

Receiver Locking

Once the receiver is locked, the corresponding `LOCK` bit in the `DIRSTAT` register is set. This bit can be polled to detect the `LOCK` condition. Another option is to use the `SPDIF_RX_LOCK_START` interrupt in the `DAI_IRPTL_H/L` register. This triggers the `DAI_INTH/L` interrupt once the receiver is locked. From this point on, the S/PDIF starts producing extracted output serial data. The data is guaranteed to be correct only after the `LOCK` goes high.

Status Bits

After the receiver is locked, the other status bits in the receiver status (`DIRSTAT`) and the channel status (`DIRCHANL/R`) registers can be read. Interrupts can also be used with some status bits.

Interrupted Data Streams on the Receiver

When using the SPDIF receiver with data streams that are likely to be interrupted, (in other words unplugged and reconnected), it is necessary to take some extra steps to ensure that the SPDIF receiver's digital PLL will re lock to the stream. The steps to accomplish this are described below.

1. Setup interrupts within the DAI so that the SPDIF RX can generate an interrupt when the stream is reconnected.
2. Within the interrupt service routine (ISR), stop and restart the digital PLL. This is accomplished by setting and then clearing bit 7 of the SPDIF receiver control register.
3. Return from the ISR and continue normal operation.

This method of resetting the digital PLL has been shown to provide extremely reliable performance when SPDIF inputs that are interrupted or unplugged momentarily occur.

The following procedure and the example code show how to reset the digital PLL. Note that all of the SPDIF receiver interrupts are handled through the DAI interrupt controller. [For more information, see “DAI/DPI Interrupt Controller” on page 4-65.](#)

1. Initialize the No Stream Interrupt

```
/* Enable interrupts (globally) */
BIT SET MODE1 IRPTEN;
/* unmask DAI Hi=Priority Interrupt */
bit set imask DAIHI;
ustat1 = DIR_NOSTREAM_INT;

/* Enable no-stream Interrupt on Falling Edge. Interrupt
occurs when the stream is reconnected */
dm(DAI_IRPTL_FE) = ustat1;
```

DAI Programming Examples

```
/* Enable Hi-priority DAI interrupt */
dm(DAI_IRPTL_PRI) = ustat1;

/* If more than 1 DAI interrupt is being used, it is necessary to determine which interrupt occurred here */

/* Interrupt Service Routine for the DAI Hi-Priority Interrupt. This ISR triggered when the DIR sets no_stream bit */
_DAIisrH:
```

2. Reset the Digital PLL Inside of the ISR

```
r8=dm(DAI_IRPTL_H);          /* Reading DAI_IRPTL_H
                               clears interrupt */
ustat2=dm(DIRCTL);
bit set ustat2 DIR_PLLDIS; /* bit_7 disables Dpll only */
dm(DIRCTL)=ustat2;
bit clr ustat2 DIR_PLLDIS; /*reenable the digital pll */
dm(DIRCTL)=ustat2;
```


10 ASYNCHRONOUS SAMPLE RATE CONVERTER

Sample rate conversion is the process of converting a digital signal from one sampling rate to another, while modifying the information that is carried by the signal as little as possible.

Sample rate conversion is useful because different systems use different sampling rates for a variety of reasons. Because the physics of sampling require a minimum sampling rate, other factors determine the actual rates used. For example, different audio systems can use rates of 44.1, 48, and 96 kHz and programs need to transfer source material between these systems. Just replaying the existing data at the new rate does not work — it introduces large changes in pitch (for audio) and movement as well (for video), plus it cannot be done in real time.

The asynchronous sample rate converter (SRC), as it is implemented in the ADSP-21367/8/9 and ADSP-2137x processors, is used to perform asynchronous sample rate conversion across independent stereo channels without using any internal processor resources. Furthermore, the SRC blocks can be configured to operate together to convert multichannel audio data without phase mismatches. Finally, the SRC is used to clean up audio data from jittery clock sources such as the S/PDIF receiver.

The SRC contains four blocks (SRC0–3). It also is the same core that is used in the Analog Devices AD1896 192 kHz Stereo Asynchronous Sample Rate Converter. The top-level block diagram of the SRC module is shown in [Figure 10-4 on page 10-9](#).

The SRC has a 3-wire interface for the serial input and output ports that supports left-justified, I²S, and right-justified (16-, 18-, 20-, 24-bit) modes. Additionally, the serial interfaces support time-division

multiplexing (TDM) mode for daisy-chaining multiple SRCs to a processor. The serial output data is dithered down to 20, 18, or 16 bits when 20-, 18-, or 16-bit output data is selected.

Theory of Operation

The SRC sample rate converts the data from the serial input port to the sample rate of the serial output port. The sample rate at the serial input port can be asynchronous with respect to the output sample rate of the output serial port.

Conceptually, the SRC interpolates the serial input data by a rate of 2^{20} and samples the interpolated data stream by the output sample rate. In practice, a 64-tap FIR filter with 2^{20} poly phases, a FIFO, a digital-servo loop that measures the time difference between the input and output samples within 5 ps, and a digital circuit to track the sample rate ratio are used to perform the interpolation and output sampling. The digital-servo loop and sample rate ratio circuit automatically track the input and output sample rates.

The digital-servo loop measures the time difference between the input and output sample rates within 5 ps. This is necessary in order to select the correct polyphase filter coefficient. The digital-servo loop has excellent jitter rejection for both input and output sample rates as well as the master clock. The jitter rejection begins at less than 1 Hz. This requires a long settling time whenever the SRC is enabled or when the input or output sample rate changes.

To reduce the settling time when the SRC is enabled or there is a change in the sample rate, the digital-servo loop enters the fast settling mode. When the digital-servo loop has adequately settled in the fast mode, it switches into the normal or slow settling mode and continues to settle until the time difference measurement between input and output sample rates is within 5 ps. During fast mode, the MUTE_OUT signal is asserted high.

Normally, the `MUTE_OUT` signal is connected to the `MUTE_IN` signal. The `MUTE_IN` signal is used to softly mute the SRC upon assertion and softly stop muting the SRC when it is deasserted.

The sample rate ratio circuit is used to scale the filter length of the FIR filter for decimation. Hysteresis in measuring the sample rate ratio is used to avoid oscillations in the scaling of the filter length, which would cause distortion on the output.

However, when multiple SRCs are used with the same serial input port clock and the same serial output port clock, the hysteresis causes different group delays between multiple SRCs. A phase-matching mode feature was added to the SRC to address this problem. In phase-matching mode, one SRC, (the master), transmits its sample rate ratio to the other SRCs, (the slaves), so that the group delay between the multiple SRCs remains the same.

Asynchronous sample rate conversion is converting data from one clock source at a sample rate to another clock source at the same or a different sample rate. The simplest approach to an asynchronous sample rate conversion is the use of a zero-order hold between the two samplers shown in [Figure 10-1](#). In the figure, f_{S_IN} and f_{S_OUT} are the input and output sampling frequencies, respectively, and $T1$ and $T2$ are the time periods that correspond to the input and output. In an asynchronous system, $T2$ is never equal to $T1$ nor is the ratio between $T2$ and $T1$ rational. As a result, samples at f_{S_OUT} are repeated or dropped, thus producing an error in the resampling process. The frequency domain shows the wide side lobes that result from this error when the sampling of f_{S_OUT} is convolved with the attenuated images from the $\sin(x)/x$ nature of the zero-order hold. The images at f_{S_IN} , (dc signal images), of the zero-order hold are infinitely attenuated. Since the ratio of $T2$ to $T1$ is an irrational number, the error resulting from the resampling at f_{S_OUT} can never be eliminated. However, the error can be significantly reduced through interpolation of the input data at f_{S_IN} . The SRC is conceptually interpolated by a factor of 2^{20} .

Theory of Operation

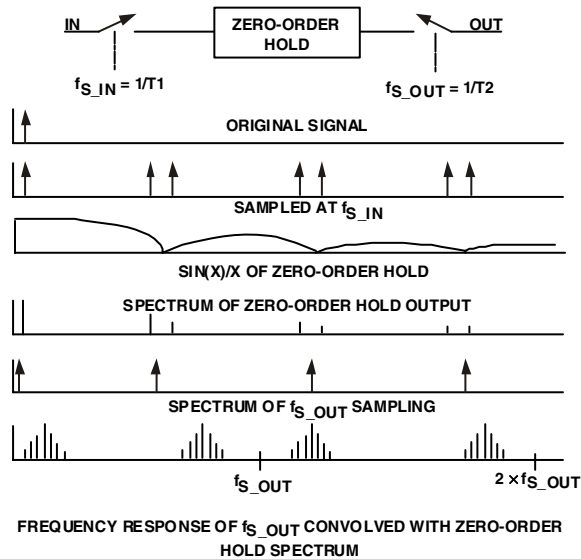


Figure 10-1. Zero-Order Hold Used by f_{S_OUT} to Resample Data From f_{S_IN}

Conceptual Model

In SRC theory, interpolation of the input data by a factor of 2^{20} involves placing $(2^{20} - 1)$ samples between each f_{S_IN} sample. [Figure 10-2](#) and [Figure 10-3 on page 10-6](#) shows both the time domain and the frequency domain of interpolation by a factor of 2^{20} . Conceptually, interpolation by 2^{20} involves the steps of zero-stuffing $(2^{20} - 1)$ samples between each f_{S_IN} sample and convolving this interpolated signal with a digital low-pass filter to suppress the images. In the time domain, it can be seen that f_{S_OUT} selects the closest $f_{S_IN} \times 2^{20}$ sample from the zero-order hold as opposed to the nearest f_{S_IN} sample in the case of no interpolation. This significantly reduces the resampling error.

Asynchronous Sample Rate Converter

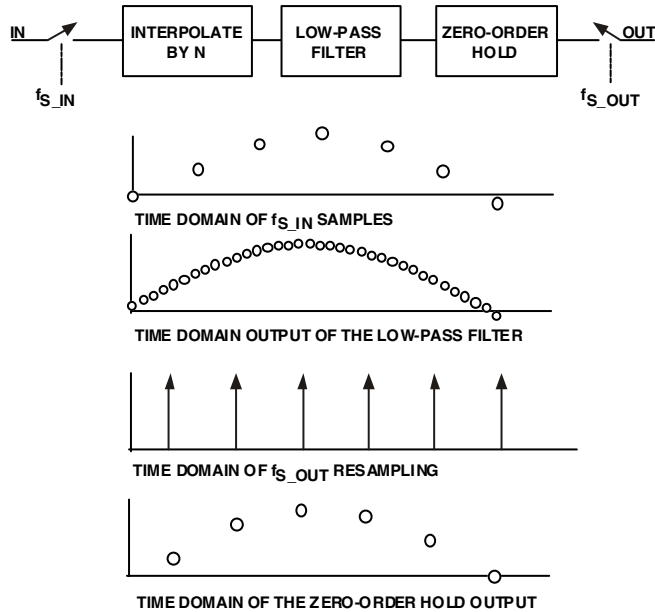


Figure 10-2. Time Domain of the Interpolation and Resampling

In the frequency domain shown in [Figure 10-3 on page 10-6](#), the interpolation expands the frequency axis of the zero-order hold. The images from the interpolation can be sufficiently attenuated by a good low-pass filter. The images from the zero-order hold are now pushed by a factor of 2^{20} closer to the infinite attenuation point of the zero-order hold, which is $f_{S_IN} \times 2^{20}$. The images at the zero-order hold are the determining factor for the fidelity of the output at f_{S_OUT} . The worst-case images that can be computed from the zero-order hold frequency response are:

$$Maximum\ Image = \frac{\sin\left(\frac{\pi \times F}{f_{S_INTERP}}\right)}{\left(\frac{\pi \times F}{f_{S_INTERP}}\right)}$$

Theory of Operation

F is the frequency of the worst-case image which is:

$$2^{20} \times f_{S_IN} \pm \frac{f_{S_IN}}{2}$$

and f_{S_INTERP} is

$$f_{S_IN} \times 2^{20}$$

The following worst-case images would appear for $f_{S_IN} = 192$ kHz:

Image at $f_{S_INTERP} - 96$ kHz = -125.1 dB

Image at $f_{S_INTERP} + 96$ kHz = -125.1 dB

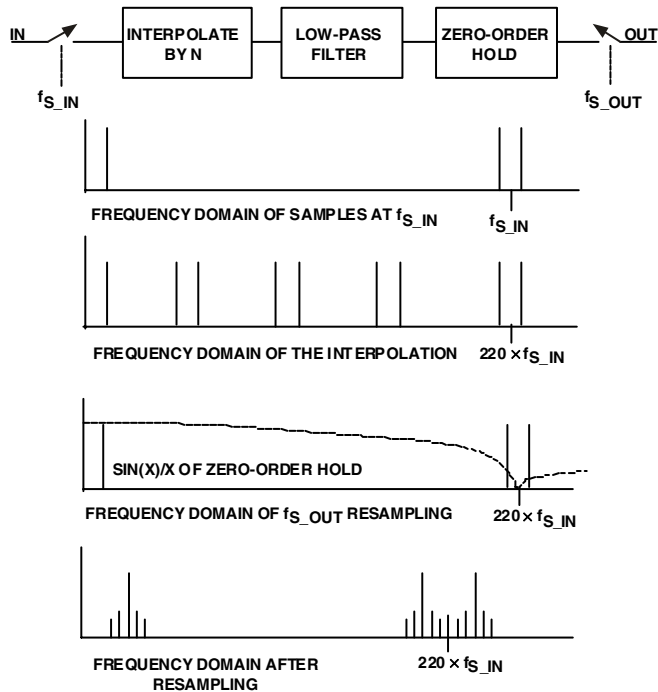


Figure 10-3. Frequency Domain of the Interpolation and Resampling

Hardware Model

The output rate of the low-pass filter of [Figure 10-2](#) is the interpolation rate, $2^{20} \times 192 \text{ kHz} = 201.3 \text{ GHz}$. Sampling at a rate of 201.3 GHz is clearly impractical, not to mention the number of taps required to calculate each interpolated sample. However, since interpolation by 2^{20} involves zero-stuffing $2^{20} - 1$ samples between each f_{S_IN} sample, most of the multiplies in the low-pass FIR filter are by zero. A further reduction can be realized by the fact that since only one interpolated sample is taken at the output at the f_{S_OUT} rate, only one convolution needs to be performed per f_{S_OUT} period instead of 2^{20} convolutions. A 64-tap FIR filter for each f_{S_OUT} sample is sufficient to suppress the images caused by the interpolation.

The difficulty with the above approach is that the correct interpolated sample needs to be selected upon the arrival of f_{S_OUT} . Since there are 2^{20} possible convolutions per f_{S_OUT} period, the arrival of the f_{S_OUT} clock must be measured with an accuracy of $1/201.3 \text{ GHz} = 4.96 \text{ ps}$. Measuring the f_{S_OUT} period with a clock of 201.3 GHz frequency is clearly impossible; instead, several coarse measurements of the f_{S_OUT} clock period are made and averaged over time.

Another difficulty with the above approach is the number of coefficients required. Since there are 2^{20} possible convolutions with a 64-tap FIR filter, there needs to be 2^{20} polyphase coefficients for each tap, which requires a total of 2^{26} coefficients. To reduce the amount of coefficients in ROM, the SRC stores a small subset of coefficients and performs a high-order interpolation between the stored coefficients. So far the above approach works for the case of $f_{S_OUT} > f_{S_IN}$. However, in the case when the output sample rate, f_{S_OUT} , is less than the input sample rate, f_{S_IN} , the ROM starting address, input data, and the length of the convolution must be scaled. As the input sample rate rises over the output sample rate, the anti-aliasing filter's cutoff frequency has to be lowered because the Nyquist frequency of the output samples is less than the Nyquist frequency of the input samples. To move the cutoff frequency of the

Sample Rate Converter Architecture

anti-aliasing filter, the coefficients are dynamically altered and the length of the convolution is increased by a factor of (f_{S_IN}/f_{S_OUT}) . This technique is supported by the Fourier transform property that if $f(t)$ is $F(\omega)$, then $f(k \times t)$ is $F(\omega/k)$. Thus, the range of decimation is simply limited by the size of the RAM.

Sample Rate Converter Architecture

Figure 10-4 shows a top level block diagram of the SRC module and Figure 10-5 shows architecture details. The sample rate converter's FIFO block adjusts the left and right input samples and stores them for the FIR filter's convolution cycle. The f_{S_IN} counter provides the write address to the FIFO block and the ramp input to the digital-servo loop. The ROM stores the coefficients for the FIR filter convolution and performs a high-order interpolation between the stored coefficients. The sample rate ratio block measures the sample rate by dynamically altering the ROM coefficients and scaling the FIR filter length and input data. The digital-servo loop automatically tracks the f_{S_IN} and f_{S_OUT} sample rates and provides the RAM and ROM start addresses for the start of the FIR filter convolution.



The master clock input (MCLK) shown in Figure 10-4 is peripheral clock (PCLK) divided by 4 (which is core clock divided by 8). Therefore, $MCLK = PCLK \div 4 = CCLK \div 8$

Asynchronous Sample Rate Converter

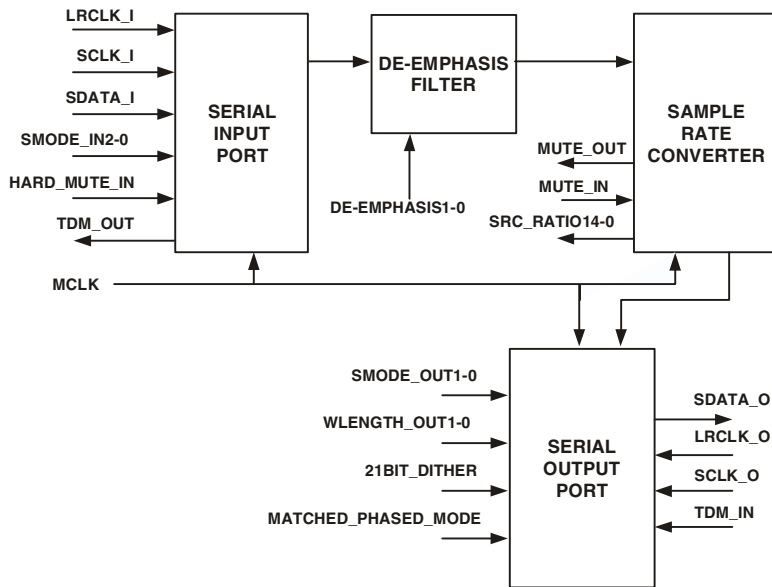


Figure 10-4. Sample Rate Converter Block Diagram

The FIFO receives the left and right input data and adjusts the amplitude of the data for both the soft muting of the SRC and the scaling of the input data by the sample rate ratio before storing the samples in RAM. The input data is scaled by the sample rate ratio because as the FIR filter length of the convolution increases, so does the amplitude of the convolution output. To keep the output of the FIR filter from saturating, the input data is scaled down by multiplying it by (f_{S_OUT}/f_{S_IN}) when $f_{S_OUT} < f_{S_IN}$. The FIFO also scales the input data to mute and stop muting the SRC.

The RAM in the FIFO is 512 words deep for both left and right channels. An offset to the write address, provided by the f_{S_IN} counter, is added to prevent the RAM read pointer from overlapping the write address. The offset is selectable by the `GRPDLYS` (group delay select) signal. A small offset (16) is added to the write address pointer when `GRPDLYS` is high, and a large offset, (64), is added to the write address pointer when `GRPDLYS` is

Sample Rate Converter Architecture

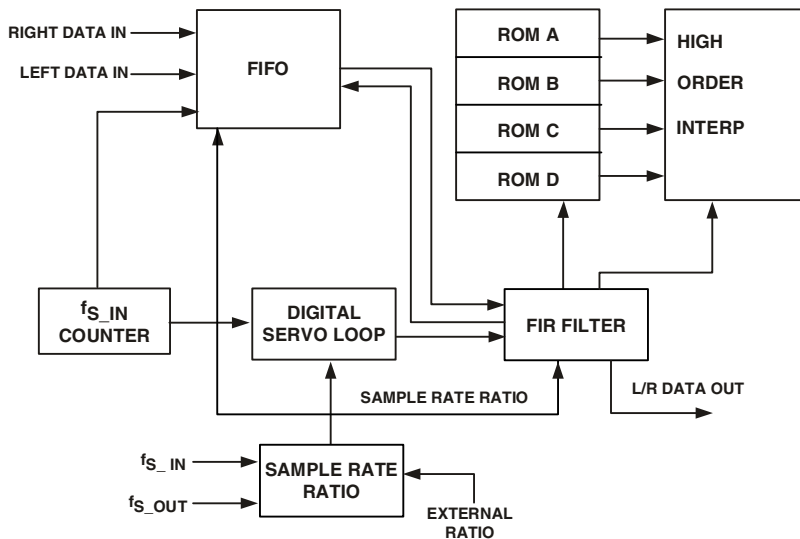


Figure 10-5. Sample Rate Converter Architecture

low. Increasing the offset of the write address pointer is useful for applications when small changes in the sample rate ratio between f_{S_IN} and f_{S_OUT} are expected. The maximum decimation rate can be calculated from the RAM word depth and **GRPDLYS** as $(512 - 16)/64$ taps = 7.75 for short group delay and $(512 - 64)/64$ taps = 7 for long group delay.

The digital-servo loop is essentially a ramp filter that provides the initial pointer to the address in RAM and ROM for the start of the FIR convolution. The RAM pointer is the integer output of the ramp filter while the ROM pointer is the fractional part. The digital-servo loop must be able to provide excellent rejection of jitter on the f_{S_IN} and f_{S_OUT} clocks as well as measure the arrival of the f_{S_OUT} clock within 4.97 ps. The digital-servo loop also divides the fractional part of the ramp output by the ratio of f_{S_IN}/f_{S_OUT} for the case when $f_{S_IN} > f_{S_OUT}$, to dynamically alter the ROM coefficients.

The digital-servo loop is implemented with a multi-rate filter. To settle the digital-servo loop filter quickly at startup or at a change in the sample rate, a *fast mode* has been added to the filter. When the digital-servo loop starts up or the sample rate is changed, the digital-servo loop kicks into fast mode to adjust and settle on the new sample rate. Upon sensing the digital-servo loop settling down to some reasonable value, the digital-servo loop kicks into *normal* or *slow mode*. During fast mode, the `MUTE_OUT` signal of the SRC is asserted to remind the user to mute the SRC which avoids clicks and pops.

The FIR filter is a 64-tap filter in the case of $f_{S_OUT} < f_{S_IN}$ and is $(f_{S_IN}/f_{S_OUT}) \times 64$ taps for the case when $f_{S_IN} > f_{S_OUT}$. The FIR filter performs its convolution by loading in the starting address of the RAM address pointer and the ROM address pointer from the digital-servo loop at the start of the f_{S_OUT} period. The FIR filter then steps through the RAM by decrementing its address by 1 for each tap, and the ROM pointer increments its address by the $(f_{S_OUT}/f_{S_IN}) \times 2^{20}$ ratio for $f_{S_IN} > f_{S_OUT}$ or 2^{20} for $f_{S_OUT} < f_{S_IN}$. Once the ROM address rolls over, the convolution is complete. The convolution is performed for both the left and right channels, and the multiply/accumulate circuit used for the convolution is shared between the channels.

The f_{S_IN}/f_{S_OUT} sample rate ratio circuit is used to dynamically alter the coefficients in the ROM for the case when $f_{S_IN} > f_{S_OUT}$. The ratio is calculated by comparing the output of an f_{S_OUT} counter to the output of an f_{S_IN} counter. If $f_{S_OUT} > f_{S_IN}$, the ratio is held at one. If $f_{S_IN} > f_{S_OUT}$, the sample rate ratio is updated if it is different by more than two f_{S_OUT} periods from the previous f_{S_OUT} to f_{S_IN} comparison. This is done to provide some hysteresis to prevent the filter length from oscillating and causing distortion.

However, the hysteresis of the f_{S_OUT}/f_{S_IN} ratio circuit can cause phase mismatching between two SRCs operating with the same input and output clocks. Since the hysteresis requires a difference of more than two f_{S_OUT} periods to update the f_{S_OUT}/f_{S_IN} ratio, two SRCs may have

SRC Operation

differences in their ratios from 0 to 4 f_{S_OUT} period counts. The f_{S_OUT}/f_{S_IN} ratio adjusts the filter length of the SRC, which corresponds directly with the group delay. Thus, the magnitude in the phase difference depends upon the resolution of the f_{S_OUT} and f_{S_IN} counters. The greater the resolution of the counters, the smaller the phase difference error.

Group Delay

When multiple SRCs are used with the same serial input port clock and the same serial output port clock, the hysteresis causes different group delays between multiple SRCs. The filter group delay of the SRC is given by the equations:

$$GDS = \frac{16}{f_{S_IN}} + \frac{32}{f_{S_IN}} \text{seconds for } f_{S_OUT} > f_{S_IN}$$

$$GDS = \frac{16}{f_{S_IN}} + \left(\frac{32}{f_{S_IN}} \right) \times \left(\frac{f_{S_IN}}{f_{S_OUT}} \right) \text{seconds for } f_{S_OUT} < f_{S_IN}$$

SRC Operation

The following sections provide details on the SRC's operation within the ADSP-21367/8/9 and ADSP-2137x processors.



The maximum data that is transferred through the SRCs is 24-bit. When 32-bit clock and data is provided to the 128 dB SRC, the least significant 8-bits are zero. However for a 140 dB SRC, the least significant 8-bits contain the ratio information. This information should be considered while interfacing the SRC to interfaces such as the S/PDIF.

Enabling the SRC

When the `SRCx_ENABLE` bit (bit 31 in the SRC control registers) is set (= 1), the SRC begins its initialization routine where all locations in the FIFO are initialized to zero, `MUTE_OUT` is cleared, and any output pins are enabled.

The `SRCx_ENABLE` bit should be held low for a minimum of five `PCLK` cycles when setting or clearing the bit. It is recommended that the SRC be disabled when changing modes.

When the `SRCx_ENABLE` bit is set or there is a change in the sample rate between `LRCLK_I` and `LRCLK_O`, the `MUTE_OUT` pin is cleared. The `MUTE_OUT` pin remains cleared until the digital-servo loop's internal fast settling mode is complete. When the digital-servo loop has switched to slow settling mode, the `MUTE_OUT` pin is set. While `MUTE_OUT` is cleared, the `MUTE_IN` pin should be cleared as well to prevent any major distortion in the audio output samples.

Serial Data Ports

The serial data ports provide the interface through which data is transferred into and out of the SRC modules. The following sections describe the various data formats and the available modes of operation.

Data Format

The serial data input port mode is set by the logic levels on the `SRCx_SMODEIN[0:2]` bits that are located in the `SRCCTLx` registers. The serial data input port modes available are left-justified, I²S, TDM and right-justified, 16, 18, 20, or 24 bits as defined in [Table 10-1](#).

The serial data output port mode is set by the logic levels on the `SRCx_SMODE_OUT[0:1]` bits. The serial mode can be changed to left-justified, I²S, right-justified, or TDM as defined in [Table 10-2](#). The output word width can be set by using the `SRCx_LENOUT[0:1]` bits as shown in

SRC Operation

Table 10-1. Serial Data Input Port Mode

SRCx_SMODE_0:2			Interface Format
2	1	0	
0	0	0	Left-justified
0	0	1	I ² S
0	1	0	TDM
0	1	1	RESERVED
1	0	0	Right-justified, 16 bits
1	0	1	Right-justified, 18 bits
1	1	0	Right-justified, 20 bits
1	1	1	Right-justified, 24 bits

Table 10-3. When the output word width is less than 24 bits, dither is added to the truncated bits. The right-justified serial data out mode assumes 64 SCLK₀ cycles per frame, divided evenly for left and right. Please note that 8 bits of each 32-bit subframe are used for transmitting matched-phase mode data as shown in [Figure 10-9 on page 10-18](#). The SRC also supports 16-bit, 32-clock packed input and output serial data in left-justified and I²S format.

Table 10-2. Serial Data Output Port Mode

SRCx_SMODEOUT_0:1		Interface Format
1	0	
0	0	Left-justified
0	1	I ² S
1	0	TDM
1	1	Right-justified

Table 10-3. Word Width

SRCx_LENOUT_0:1		Interface Format
1	0	
0	0	24 bits
0	1	20 bits
1	0	18 bits
1	1	16 bits

Time-Division Multiplex (TDM) Output Mode

In TDM output mode, several SRCs can be daisy-chained together and connected to the serial input port of an ADSP-21367/8/9 and ADSP-2137x processor or other processor ([Figure 10-6](#)). The SRC contains a 64-bit parallel load shift register. When the LRCLK_0 pulse arrives, each SRC parallel loads its left and right data into the 64-bit shift register. The input to the shift register is connected to the TDM_IN signal, while the output is connected to the SDATA_0 signal. By connecting the SDATA_0 signal to the TDM_IN signal of the next SRC, a large shift register is created, which is clocked by the SCLK_0 signal.

The number of SRCs that can be daisy-chained together is limited by the maximum frequency of SCLK_0 signals, which is about 25 MHz. For example, if the output sample rate, f_S , is 48 kHz, up to eight SRCs could be connected since $512 \times f_S$ is less than 25 MHz.

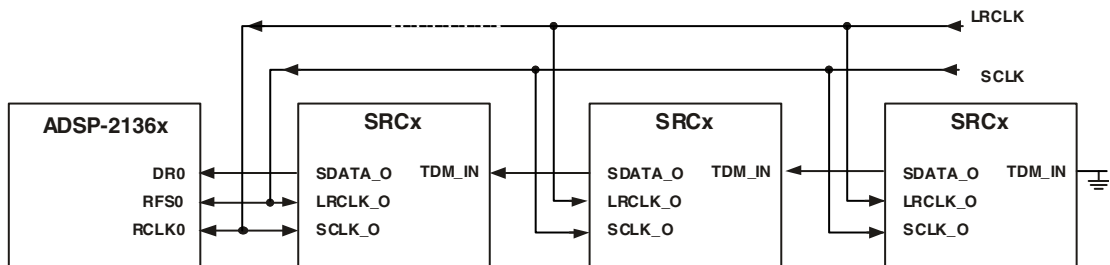


Figure 10-6. TDM Output Mode

TDM Input Mode

In TDM input mode, several SRCs can be daisy-chained together and connected to the serial input port of an ADSP-21367/8/9 and ADSP-2137x processor or other processor (Figure 10-7). The SRC contains a 64-bit parallel load shift register. When the LRCLK_I pulse arrives, each SRC parallel loads its left and right data into the 64-bit shift register. The input to the shift register is connected to the SDATA_IN, while the output is connected to the TDM_I signal. By connecting the SDATA_I signal to the TDM_I signal of the next SRC, a large shift register is created, which is clocked by the SCLK_I signal.

The number of SRCs that can be daisy-chained together is limited by the maximum frequency of the SCLK_0 signal, which is about 25 MHz. For example, if the output sample rate, f_S , is 48 kHz, up to eight SRCs could be connected since $512 \times f_S$ is less than 25 MHz.

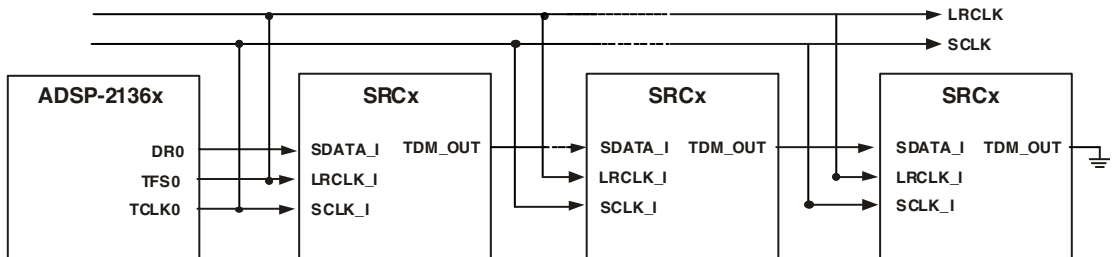


Figure 10-7. TDM Input Mode

Matched-Phase Mode

The matched-phase mode is the mode discussed in “[Theory of Operation](#)” on [page 10-2](#). This mode eliminates the phase mismatch between multiple SRCs. The master SRC device transmits its f_{S_OUT}/f_{S_IN} ratio through the SDATA_0 pin to the slave SRC’s TDM_IN pins. The slave SRCs receive the transmitted f_{S_OUT}/f_{S_IN} ratio and use the transmitted f_{S_OUT}/f_{S_IN} ratio instead of their own internally-derived f_{S_OUT}/f_{S_IN} ratio as shown

Asynchronous Sample Rate Converter

in [Figure 10-8](#). The master device can have both its serial ports in slave mode as depicted, or either one in master mode. The slave SRCs must have their `MATASE_2` bits set to 1, respectively. The `LRCLK_I` and `LRCLK_0` signals may be asynchronous with respect to each other in this mode.

There must be 32 `SCLK_0` cycles per subframe in matched-phase mode. The SRC supports the matched-phase mode for all serial output data formats: left-justified, I²S, right-justified, and TDM mode.

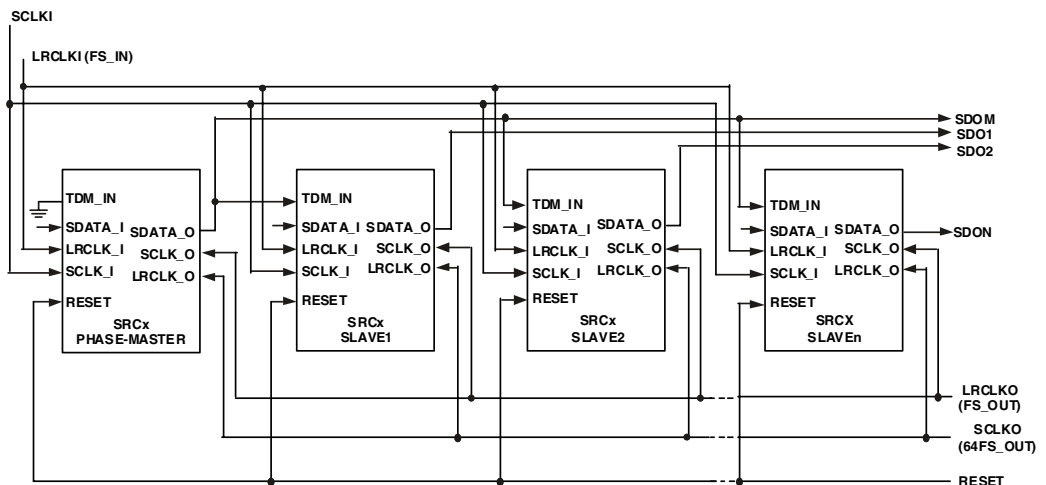


Figure 10-8. Typical Configuration for Matched-Phase Mode Operation

Note that in the left-justified, I²S, and TDM modes, the lower eight bits of each channel subframe are used to transmit the matched-phase data. In right-justified mode, the upper eight bits are used to transmit the matched-phase data. This is shown in [Figure 10-9](#).

SRC Operation

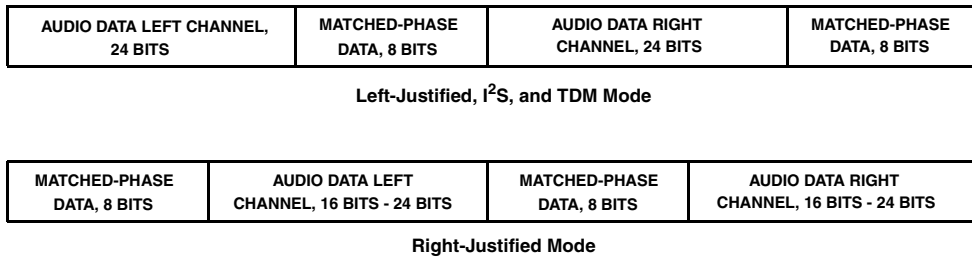


Figure 10-9. Matched-Phase Data Transmission

Bypass Mode

When the `BYPASS` bit is set (=1), the input data bypasses the sample rate converter and is sent directly to the serial output port. Dithering of the output data when the word length is set to less than 24 bits is disabled. This mode is ideal when the input and output sample rates are the same and the `LRCLK_I` and `LRCLK_O` signals are synchronous with respect to each other. This mode can also be used for passing through non-audio data since no processing is performed on the input data in this mode.

De-Emphasis Filter

As discussed, the serial input port generates a frame synchronization signal, `UN_fS_IN`, that derives its clock from the positive edge of the `PCLK` signal. The `UN_fS_IN` signal asserts when a new frame of left and right data is available for the de-emphasis filter and the SRC. The de-emphasis filter is used to de-emphasize audio data that has been emphasized. The type of de-emphasis filter is selected by the `SRCn_DEEMPHASIS1-0` bits and is based on the input sample rate as follows:

- 00 – No de-emphasis, audio data is passed directly to the SRC
- 01 – 32 kHz sample rate de-emphasis filter

- 10 – 44.1 kHz sample rate de-emphasis filter
- 11 – 48 kHz sample rate de-emphasis filter

After the audio data is passed from the de-emphasis filter to the SRC, the SRC converts the audio data from the input sample rate to the output sample rate. When the serial output port needs new data, the frame synchronization signal, ($UN_f_{S_OUT}$), is asserted from the serial output port. Like the $UN_f_{S_IN}$ signal, the $UN_f_{S_OUT}$ signal derives its clock from the positive edge of the $PCLK$ signal. The $UN_f_{S_OUT}$ and $UN_f_{S_IN}$ signals are used by the SRC to perform the sample rate conversion. The $SRCRAT$ register indicates the sample rate ratio of $UN_f_{S_OUT}/UN_f_{S_IN}$.

Mute Control

When the $SRCx_ENABLE$ bit is enabled (set = 1), or when the sample rate between the input and output $LRCLK$ changes, the SRC begins its initialization routine and the $MUTE_OUT$ signal is asserted. When $MUTE_OUT$ is asserted, the $MUTE_IN$ signal should also be asserted to avoid any unwanted output.

When the $MUTE_IN$ pin is asserted high, the $MUTE_IN$ control performs a soft mute by linearly decreasing the input data to the SRC FIFO to zero, (–144 dB attenuation). A 12-bit counter, clocked by the $LRCLK_I$ signal, is used to control the mute attenuation. Therefore, the time it takes from the assertion of the $MUTE_IN$ signal to –144 dB, (full mute attenuation) is 4096 $LRCLK$ clock cycles. Likewise, the time it takes to reach 0 dB mute attenuation from the deassertion of the $MUTE_IN$ signal is 4096 $LRCLK$ cycles.

The mute feature of the SRC can be controlled automatically in hardware using the $MUTE_IN$ signal by connecting it to the $MUTE_OUT$ signal. By default, the two signals for each SRC are connected. Automatic muting can be disabled using the $SRCx_MUTE_DIS$ bits in the $SRCMUTE$ register.

SRC Operation

Muting can also be controlled in software using the MUTE bits (SRCx_SOFTMUTE, SRCx_HARD_MUTE, SRCx_AUTO_MUTE) in the SRC control register (SRCCTL) as described below. For more information, see [“SRC Registers” on page 10-21](#).

Soft Mute

When the SRCx_SOFTMUTE bit in the SRCCTL register is set, the MUTE_IN signal is asserted, and the SRC performs a soft mute by linearly decreasing the input data to the SRC FIFO to zero, (–144 dB attenuation) as described for automatic hardware muting.

Hard Mute

When the SRCx_HARD_MUTE bit in the SRCCTL register is set, the SRC immediately mutes the input data to the SRC FIFO to zero, (–144 dB attenuation).

Auto Mute

When the SRCx_AUTO_MUTE bit in the SRCCTL register is set, the SRC communicates with the SPDIF receiver peripheral to determine when the input should mute. Each SRC is connected to the NOAUDIO bits in the output of the SPDIF receiver (see [“Receiver Status Register \(DIRSTAT\)” on page A-94](#)). When this signal is asserted, the SRC immediately mutes the input data to the SRC FIFO to zero, (–144 dB attenuation). This mode is useful for automatic detection of non-PCM audio data received from the S/PDIF receiver.

SRC Registers

The SRC uses five 32-bit registers to configure and operate the SRC module.

- **SRCCTL0, SRC control 0.** This read/write register is used to control the operating modes, filters, and data formats used in the SRC0 and SRC1 modules. This register is located at address 0x2490.
- **SRCCTL1, SRC control 1.** This read/write register is used to control the operating modes, filters, and data formats used in the SRC2 and SRC3 modules. This register is located at address 0x2491.
- **SRCMUTE, SRC mute.** This read/write register performs mute-out to mute-in control and provides status information for the SRC3–0 modules. This register is located at address 0x2492.
- **SRCRAT0, SRC output to input ratio 0.** This read-only register reports the mute and I/O sample ratio for SRC0 and SRC1. This register is located at address 0x2498.
- **SRCRAT1, SRC output to input ratio 1.** This read-only register reports the mute and I/O sample ratio for SRC2 and SRC3. This register is located at address 0x2499.

For complete register bit descriptions, see [“SRC Control Registers \(SRC-CTLx\)” on page A-97](#).

Programming the SRC Module

Use the following guidelines when developing programs that include the SRC module.

SRC Control Register Programming

Initially, programs configure the SRC control registers `SRCCTL0` and `SRCCTL1`. The `SRCCTL0` register contains control parameters for the SRC0 and SRC1 modules and the `SRCCTL1` register contains control values for the SRC2 and SRC3 modules. The control parameters include mute information, data formats for input and output ports, de-emphasis enable, dither enable, and matched-phase mode enable for multiple SRCs. Write the settings to the desired control register at least one cycle before setting the corresponding SRC module enable bit, `SRCx_ENABLE`.

SRU Programming

The SRU (signal routing unit) needs to be programmed in order to connect the SRCs to the output pins or any other peripherals.

For normal operation, the data, clock, and frame sync signals need to be routed as shown in [Table 10-4](#).

Table 10-4. SRC Signal Routing

Signal	Definition	
<code>SRCx_DAT_IP_I</code>	SRC module data input	
<code>SRCx_CLK_IP_I</code>	SRC module clock input	
<code>SRCx_FS_IP_I</code>	SRC module frame sync input	
<code>SRCx_DAT_OP_I</code>	SRC module data output	
<code>SRCx_CLK_OP_I</code>	SRC module clock output	
<code>SRCx_FS_OP_O</code>	SRC module frame sync output	

For information on using the SRU, see [“Making Connections in the SRUs” on page 4-15](#), and [“DAI/SRU1 Connection Groups” on page 4-18](#).

SRC Mute-Out Interrupt

Once the SRC is locked (after 4K input samples), the corresponding SRCx_MUTE_OUT bit in the DAI_IRPTL_H/L register is set. This generates the DAI_INTH/L interrupt. From this point, the SRC produces output serial data at the output sampling frequency. The SRC mute signals can be used to generate interrupts on their leading edge, falling edge, or both, depending on how the DAI_IRPT_RE/FE registers are programmed.

Sample Rate Ratio

Once the SRC mute interrupt has triggered, the SRCRAT0 and SRCRAT1 registers can be read to find the ratio of output to input sampling frequency. This ratio is reported in 4.11 (integer.fraction) format.

Programming Summary

Since the SRC data is not available to the core, programming the SRC peripheral involves simply connecting the SRU to the on-chip (serial ports or input data port) or off-chip (DAI pins) serial devices. These devices provide the clock and data to be converted, and select the desired operating mode in the SRC control register. This setup can be accomplished in two steps.

1. Connect each of the four SRCs to their two serial clocks (SRCx_CLK_IP_I, SRCx_CLK_OP_I) and frame sync inputs (SRCx_FS_IP_I, SRCx_FS_OP_I). Also connect one data input (SRCx_DAT_IP_I) and one data output (SRCx_DAT_OP_0) in the SRU.

Programming the SRC Module

In multichannel, or matched-phase modes, the TDM signals must also be connected. (See [Table 10-1 on page 10-14](#) and [Table 10-2 on page 10-14](#).)

2. Initialize the `SRCCTLx` register to enable the SRCs.

11 UART PORT CONTROLLER

The universal asynchronous receiver/transmitter (UART) is a full-duplex peripheral compatible with the PC-style, industry-standard UART. The UART converts data between serial and parallel formats. The serial format follows an asynchronous protocol that supports various word lengths, stop bits, and parity generation options. The UART includes interrupt handling hardware. Interrupts can be generated from 12 different events.

The ADSP-21367/8/9 and ADSP-2137x processors contain two UARTs which support multiprocessor communication using 9-bit address detection. This allows the units to be used in multi-drop networks using the RS-485 data interface standard.

The two independent UARTs are referred to as UART0 and UART1. Both are identical and each has its own set of control and status registers. The content in this chapter applies to both UARTs unless otherwise specified.

The UART is a DMA-capable peripheral with support for separate transmit and receive DMA master channels. It can be used in either DMA or programmed non-DMA modes of operation. The non-DMA mode requires software management of the data flow using either interrupts or polling. The DMA method requires minimal software intervention as the DMA engine itself moves the data. [For more information, see “UART DMA” on page 2-44.](#)

Either one of the peripheral timers can be used to provide a hardware-assisted autobaud detection mechanism for use with the UART. See the *ADSP-2136x SHARC Processor Programming Reference*, “Timers” chapter for more information.

Serial Communications

The UART follows an asynchronous serial communication protocol with these options:

- 5 – 8 data bits
- 1 or 2 stop bits
- None, even, or odd parity
- Baud rate = $PCLK / (16 \times \text{divisor})$, where $PCLK$ is the system clock frequency and the divisor can be a value from 1 to 65,536

All data words require a start bit and at least one stop bit. With the optional parity bit, this creates a 7- to 12-bit range for each word. The format of received and transmitted character frames is controlled by the line control register ($UARTxLCR$). Data is always transmitted and received least significant bit (LSB) first.

Figure 11-1 shows a typical physical bit stream measured on the transmit pin.

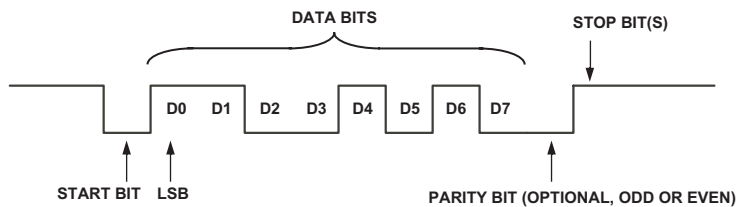


Figure 11-1. Bit Stream on the Transmit Pin

UART Control and Status Registers

The processor provides a set of PC-style, industry-standard control and status registers for each UART. These memory-mapped registers (MMRs) are byte-wide registers that are mapped as half-words with the most significant byte zero-filled.

Consistent with industry-standard interfaces, multiple registers are mapped to the same address location. The transmit holding, divisor latch low, and receive buffer registers share an address and the interrupt enable and divisor latch high registers share an address. The divisor latch access bit (UARTDLAB, bit 7) in the line control register (UARTxLCR) controls which set of registers is accessible at a given time.

Transmit and receive channels are both buffered. The UARTxTHR register buffers the transmit shift register (UARTxTSR) and the UARTxRBR register buffers the receive shift register (UARTxRSR). The shift registers are not directly accessible by software.

These registers are also shown and described in [“UART Control and Status Registers” on page A-118](#).


UARTxLCR Registers

The UART line control register (UARTxLCR) controls the format of received and transmitted character frames. The UARTSB bit (bit 6) functions even when the UART clock is disabled. Since the transmit pin normally drives high, it can be used as a flag output pin when the UART is not used. The parity and word length select controls are not valid in 9-bit mode. In 9-bit mode, the word length is always 8 and the 9th bit is transmitted instead of the parity bit.

UARTxLSR Register

The UART line status register (UARTxLSR) contains UART status information as shown in [Figure A-49 on page A-120](#).

The break interrupt (UARTBI), overrun error (UARTOE), parity error (UARTPE), and framing error (UARTFE) bits are cleared when the UART line status register (UARTxLSR) is read. The data ready (UARTDR) bit is cleared when the UART receive buffer register (UARTxRBR) is read.

 Because of the destructive nature of reading these registers, shadow registers are provided for reading the contents of the corresponding main registers. The shadow registers, UARTxIIRSH, return exactly the same contents as the main register, but without changing the register's status in any way. These registers are 32-bit registers located at address 0x3C0A (for UART0LSRSH) and 0x400A (for UART1LSRSH).

The UARTTHRE bit (bit 6) indicates that the UART transmit channel is ready for new data, and software can write to the UARTxTHR register. Writes to UARTxTHR clear the UARTTHRE bit. It is set again when data is copied from UARTxTHR to the transmit shift register (UARTxTSR). The UARTTEMT bit can be evaluated to determine whether a recently initiated transmit operation has been completed.

UARTxTHR Register

A write to the UART transmit holding register (UARTxTHR) initiates the transmit operation. The data is moved to the internal transmit shift register (UARTxTSR) where it is shifted out at a baud rate equal to $PCLK/(16 \times \text{Divisor})$ with start, stop, and parity bits appended as required. All data words begin with a 1-to-0 transition start bit. The transfer of data from the UARTxTHR register to the transmit shift register sets the transmit holding register empty status flag (UARTTHRE) in the UART line status register (UARTxLSR).

This 32-bit write only register uses only 18-bits. The other bits are filled with zeros during writes. In no-pack mode (default), only the lower byte is used—all other bits are zero filled. However in pack mode, both the high and low bytes are used ([Figure 11-2](#)). The TX9D_x bits are the ninth bit in 9-bit transmission mode. This register is mapped to the same address as the UART_xRBR and UART_xDLL registers. A write to the UART transmit holding register (UART_xTHR) initiates the transmit operation.

To access the UART_xTHR register, the UARTDLAB bit in the UART_xLCR register must be cleared. When the UARTDLAB bit is cleared, writes to this address target the UART_xTHR register, and reads from this address return the UART_xRBR register.

Note that data is transmitted and received by the least significant bit (LSB) first (bit 0) followed by the most significant bits (MSBs).

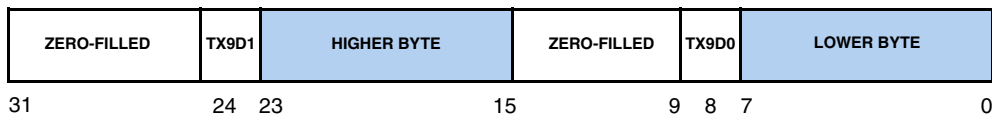


Figure 11-2. Transmit Holding Register (Packing Enabled)

UART_xRBR Register

The receive operation uses the same data format as the transmit configuration, except that the number of stop bits is always assumed to be 1. After detection of the start bit, the received word is shifted into the receive shift register (UART_xRSR) at a baud rate of $PCLK/(16 \times \text{Divisor})$. After the appropriate number of bits (including stop bit) is received, the data and any status are updated and the UART_xRSR register is transferred to the UART receive buffer register (UART_xRBR), shown in [Figure 11-3](#) and [Figure A-51 on page A-122](#). After the transfer of the received word to the UART_xRBR buffer and the appropriate synchronization delay, the data ready status flag (UARTDR) is updated.

UART Control and Status Registers

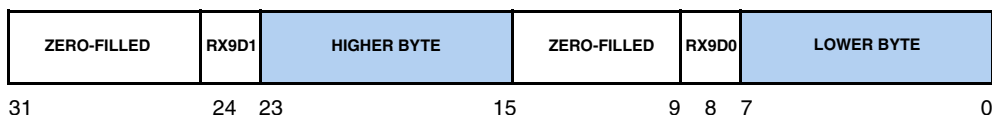



Figure 11-3. Receive Buffer Register (Packing Enabled)

A sampling clock equal to 16 times the baud rate samples the data as close to the midpoint of the bit as possible. Because the internal sample clock may not exactly match the asynchronous receive data rate, the sampling point drifts from the center of each bit. The sampling point is synchronized again with each start bit, so the error accumulates only over the length of a single word. A receive filter removes spurious pulses of less than two times the sampling clock period.

The 32-bit, read-only `UARTxRBR` register is mapped to the same address as the write-only `UARTxTHR` and `UARTxDLL` registers. In no pack mode (default), only the lower byte is used—all other bits are zero-filled. However in pack mode, both the high and low bytes are used. The `RX9Dx` bits are the 9th bit in 9-bit transmission mode. To access `UARTxRBR`, the `UARTDLAB` bit in the `UARTxLCR` register must be cleared. When the `UARTDLAB` bit is cleared, writes to this address target the `UARTxTHR` register, while reads from this address return the `UARTxRBR` register.

 Because of the destructive nature of reading these registers, shadow registers are provided for reading the contents of the corresponding main registers. The shadow registers, `UARTxRBRSH`, return exactly the same contents as the main register, but without changing the status in any way. These registers are 32-bit registers located at address `0x3C08` (for `UART0RBRSH`) and `0x4008` (for `UART1RBRSH`).

UARTxIER Register

The UART interrupt enable registers (UARTxIER) are used to enable requests for system handling of empty or full states of UART data registers. Unless polling is used as a means of action, the UARTBFIIE and/or UARTTBEIE bits in this register are normally set.

Setting these registers without enabling system DMA causes the UART to notify the processor of the state of the data inventory by means of interrupts. For proper operation in this mode, system interrupts must be enabled, and appropriate interrupt handling routines must be present. For backward compatibility, the UARTxIIR registers still reflect the correct interrupt status.

With system DMA enabled, the UART uses DMA to transfer data to or from the processor. Dedicated DMA channels are available to receive and transmit operations. Line error handling can be configured completely independently from the receive/transmit setup.

The UARTxIER register is mapped to the same address as the UARTxDLH register. To access the UARTxIER register, the UARDDLAB bit in the UARTxLCR register must be cleared.

The UART interrupts are all combined into the digital peripheral interface (DPI) interrupt. The DPI_IRPTL register determines whether an interrupt is for the transmitter or receiver. For DMA, the transmit interrupt is generated when a DMA in transmit mode is complete whereas the receive interrupt is generated when receive DMA is complete or when a receive error occurs. The UARTxRXSTAT register reports whether the interrupt is due to DMA completion or errors.

The UART receive and transmit interrupt can also be programmed through the peripheral interrupt control registers (PICR) as separate interrupts for DMA. (By default, these interrupts are not configured in the IRPTL register—the PICR register has to be programmed to configure them.) For I/O mode, both the transmit and receive interrupt can be

UART Control and Status Registers

programmed through the `PICR` register using the code select value for the UART receive interrupt (0x13 for UART0 interrupt and 0x14 for UART1 receive interrupt). Similar to I/O mode, both the transmit and receive interrupts are mapped to the receive interrupt. Then the `UARTxIER` register can be used to select the transmit or receive interrupt respectively for I/O mode. The following examples show two methods of enabling transmit DMA in I/O mode. The first method uses the DPI interrupt directly.

Listing 11-1. Enabling Transmit DMA (DPI Direct Method)

```
bit set mode1 IRPTEN;      /* enables global interrupts */
bit set imask DPII;        /* unmaskes DPI interrupt */
ustat1 = UART0_RX_INT;
dm(DPI_IRPTL_RE) = ustat1; /* enables transmit interrupt in
                           I/O mode */
```

The second method uses the DPI interrupt by programming the `PICR` with the code value of the UART0 receive interrupt (0x13).

Listing 11-2. Enabling Transmit DMA (DPI PICR Method)

```
#define MASKP14 (0x1f<<10)
#define UART0Rx (0x13<<10)

bit set mode1 IRPTEN;

/* Map the UART0 receive interrupt to P14 using the programmable
interrupt controller */
ustat1 = dm(PICR2);
bit clr ustat1 MASKP14;
bit set ustat1 UART0Rx;
dm(PICR2) = ustat1;

bit set IMASK P14I; /* Unmaskes the UART Receive interrupt */
```


For information on using the UART for DMA transfers, see “[UART DMA](#)” on page 2-44, “[DAI/DPI Interrupt Controller](#)” on page 4-65, and “[Peripheral Interrupt Priority Control Registers](#)” on page A-164.



Even though the UART has two interrupts for receive and transmit, in I/O mode, all interrupts are grouped as a single receive interrupt.

The `UARTLSIE` bit (bit 2) enables interrupt generation on an independent interrupt channel when any of the following conditions are raised by the respective bit in the UART line status register (`UARTxLSR`):

- Receive overrun error (`UARTOE`)
- Receive parity error (`UARTPE`)
- Receive framing error (`UARTFE`)
- Break interrupt (`UARTBI`)

When the `UARTTBEIE` bit is set in the `UARTxIER` register in I/O mode, the UART module immediately issues an interrupt. When initiating the transmission of a string, no special handling of the first character is required. Set the `UARTTBEIE` bit (bit 1) and let the interrupt service routine (ISR) load the first character from memory and write it to the `UARTxTHR` register in the normal manner. Accordingly, the `UARTTBEIE` bit should be cleared if the string transmission has completed.

UARTxIIR Register

For legacy reasons, the UART interrupt identification register (`UARTxIIR`) still reflects the UART interrupt status (see [Table 11-1](#)). Legacy operation may require bundling all UART interrupt sources to a single interrupt channel and servicing them all by the same software routine. This can be established by globally assigning all UART interrupts to the same

UART Control and Status Registers

interrupt priority using the peripheral interrupt priority control registers. For more information, see [“Peripheral Interrupt Priority Control Registers”](#) on page A-164.

Table 11-1. IIR Register in I/O Mode

Bit Status	NINT	Interrupt Priority	Interrupt Type	Cleared When...
000	1	–	No interrupt	–
011	0	1	Rx line status	LSR is read
100	0	2	Address detect	RBR is read
010	0	3	Rx data ready	RBR is read
001	0	4	THR empty	Write THR or Read IIR when priority = 4
000	0	5	THR and TSR empty	Write THR or Read IIR when priority = 5

When cleared, the pending interrupt bit (NINT) signals that an interrupt is pending. The STATUS field indicates the highest priority pending interrupt. The receive line status has the highest priority; the UARTTXFI interrupt has the lowest priority. In the case where both interrupts are signalling, the UARTxIIR register reads 0x06.

When a UART interrupt is pending, the interrupt service routine (ISR) needs to clear the interrupt latch explicitly. For information on how to clear the latches see [“Interrupt Identification Registers \(UARTxIIR\)”](#) on page A-124.

The transmit interrupt request is cleared by writing new data to the UARTxTHR register or by reading the UARTxIIR register. Please note the special role of the UARTxIIR register read in the case where the service routine does not want to transmit further data.

If software stops transmission, it must read the `UARTxIIR` register to reset the interrupt request. As long as the `UARTxIIR` register reads `0x04` or `0x06` (indicating that another interrupt of higher priority is pending), the `UARTxTHR` empty latch cannot be cleared by reading the `UARTxIIR` register.



The following restrictions should be noted.

1. If either the line status interrupt or the receive data interrupt has been assigned a lower interrupt priority by the interrupt controller, a deadlock condition can occur. To avoid this, always assign the lowest priority of the enabled UART interrupts to the `UARTxTHR` empty event.
2. Because of the destructive nature of reading these registers, shadow registers are provided for reading the contents of the corresponding main registers. The shadow registers, `UARTxIIRSH`, return exactly the same contents as the main register, but without changing the status in any way. These registers are 32-bit registers located at address `0x3C09` (for `UART0IIRSH`) and `0x4009` (for `UART1IIRSH`).

UARTxDLL and UARTxDLH Registers


The bit rate is characterized by the peripheral clock ($PCLK = 2 \times CCLK$) and the 16-bit divisor. The divisor is split into the UART divisor latch low byte register (`UARTxDLL`) and the UART divisor latch high byte register (`UARTxDLH`). These registers form a 16-bit divisor. The baud clock is divided by 16 so that:

Divisor = 1 when `UARTxDLL = UARTxDLH = 1`

Divisor = 65,535 when `UARTxDLL = UARTxDLH = FF`

The `UARTxDLL` register is mapped to the same address as the `UARTxTHR` and `UARTxRBR` registers. The `UARTxDLH` register is mapped to the same address as the interrupt enable register (`UARTxIER`). The `DLAB` bit in the `UARTxLCR` register must be set before the UART divisor latch registers can be accessed.


UART Control and Status Registers

 The 16-bit divisor formed by the `UARTxDLH` and `UARTxDLL` registers resets to `0x0001`, resulting in the highest possible clock frequency by default. If the UART is not used, disabling the UART clock saves power (see bits 13 and 14 in the “[Power Management Control Register \(PMCTL\)](#)” on page A-170). The `UARTxDLH` and `UARTxDLL` registers can be programmed by software before or after turning on the clock.

[Table 11-2](#) provides example divide factors required to support most standard baud rates.

Table 11-2. UART Baud Rate Examples With 100 MHz PCLK

Baud Rate	DL	Actual	% Error
2400	2604	2400.15	0.006
4800	1302	4800.31	0.007
9600	651	9600.61	0.006
19200	326	19,171.78	0.147
38400	163	38,343.56	0.147
57600	109	57,339.45	0.452
115200	54	115,740.74	0.469
921,600	7	892,857.14	3.119
6,250,000	1	6,250,000	—

 Careful selection of PCLK frequencies, that is, even multiples of desired baud rates, can result in lower error percentages.

UARTxSCR Register

The contents of the 8-bit UART scratch register (`UARTxSCR`) is reset to `0x00`. It is used for general-purpose data storage and does not control the UART hardware in any way.

For information on UART DMA registers, see [“UART DMA” on page 2-44](#).

UARTxMODE Register

The UART mode register controls miscellaneous settings such as packing and address detection. [For more information, see “Mode Registers \(UARTxMODE\)” on page A-126](#).

I/O Mode

In I/O mode, data is moved to and from the UART by the processor core. To transmit a character, load it into the `UARTxTHR` register. Received data can be read from the `UARTxRBR` register. The processor must write and read one character at time.

To prevent any loss of data and misalignments of the serial data stream, the UART line status register (`UARTxLSR`) provides two status flags for handshaking—`UARTTHRE` and `UARTDR`.

The `UARTTHRE` flag is set when the `UARTxTHR` register is ready for new data and cleared when the processor loads new data into the `UARTxTHR` register. Writing this register when it is not empty overwrites the register with the new value and the previous character is never transmitted.

The `UARTDR` flag signals when new data is available in the `UARTxRBR` register. This flag is cleared automatically when the processor reads from this register. Reading the `UARTxRBR` register when it is not full returns the previously received value. When the `UARTxRBR` register is not read in time, newly received data overwrites the `UARTxRBR` register and the overrun (`UARTOE`) flag is set.

With interrupts disabled, these status flags can be polled to determine when data is ready to move. Note that because polling is processor-intensive, it is not typically used in real-time signal processing environments.

I/O Mode

Software can write up to two words into the `UARTxTHR` register before enabling the UART clock. As soon as the UART DMA engine is enabled, those two words are sent.

Alternatively, UART writes and reads can be accomplished by interrupt service routines (ISRs). Separate interrupt lines are provided for the transmit, receive, and error signals. The independent interrupts can be enabled individually by the `UARTxIER` register. In I/O mode, the receive interrupt is generated for the following cases.

- When `UARTxRBR` is full
- On a receive overrun error
- On a receive parity error
- On a receive framing error
- On a break interrupt (`RXSIN` held low)
- When `UARTxTHR` is empty
- A transmit complete (`UARTTXFI`) interrupt
- An address detect (`UARTADI`) interrupt (for 9-bit mode)

The ISRs can evaluate the status bit field within the UART interrupt identification register (`UARTxIIR`) to determine the signalling interrupt source. If more than one source is signalling, the status field displays the one with the highest priority. Interrupts also must be assigned and unmasked by the processor's interrupt controller. The ISRs must clear the interrupt latches explicitly. See [Figure A-54 on page A-125](#) and [“Interrupt Priorities” on page B-4](#).

Packing Mode

The UART provides packed and unpacked modes of data transfer to and from the internal memory of the ADSP-21367/8/9 and ADSP-2137x processors. This mode is set using the `UARTPACK` bit (bit 0) in the `UARTxMODE` register. In unpacked mode, the data word is appended to the left with 24 zeros during transmission or reception. In packed mode, two words of data are transmitted or received with their corresponding higher bytes filled with zeros. For example, consecutive data words 0xAB and 0xCD are packed as 0x00CD 00AB in the receiver, and 0x00CD 00AB is transmitted as two words of 0xAB and 0xCD successively from the transmitter. The packed feature is provided to use the internal memory of the processor in a more efficient manner. Packing is available in both I/O and DMA modes. A control bit, `UARTPKSYN`, can be used to re-synchronize the packing. For information on using the UART for DMA transfers, see [“UART DMA” on page 2-44](#).

Note that in packed mode, both the transmitter and receiver operate with an even number of words. A transmit-buffer-empty or receive-buffer-full interrupt is generated only after an even number of words are transferred. In 9-bit mode, the address detect interrupt can be generated whenever the receiver gets an address word, irrespective of the packing mode. This helps programs respond to an address word immediately. The program is expected to take into account these features when using packed mode.



Programs must use care when using the packing feature in 9-bit mode.

Programs should write the `UARTPKSYN` bit (bit 1) with a 1 each time an address is received. This starts the reception of the following data from the lower half-word of the `UARTxRBR` register.

The address-detect interrupt is generated whenever the UART receiver receives an address, irrespective of the packing. The `DR` bit in the `UARTxLSR` register can be used to discover whether the address is in the lower (`DR = 0`) or higher half-word (`DR = 1`).

Packing Mode

The `LSR` register must be read before reading the `UARTxRBR` register, because the latter clears the `DR` bit. Reading the `UARTxRBR` register clears both the address-detect and the data-ready interrupts. In non-packed mode, when the address-detect interrupt is generated, it means that the data is ready in the `RBR` buffer while in packed mode, this is not the case.

12 TWO WIRE INTERFACE CONTROLLER

The two wire interface (TWI) controller allows a device to interface to an inter-IC bus as specified by the *Philips I²C Bus Specification version 2.1* dated January 2000.

Overview

The TWI is fully compatible with the widely used I²C bus standard. It was designed with a high level of functionality and is compatible with multimaster, multislave bus configurations. To preserve processor bandwidth, the TWI controller can be set up and a transfer initiated with interrupts only. This allows the processor to service FIFO buffer data reads and writes. Protocol-related interrupts are optional.

The TWI moves 8-bit data externally while maintaining compliance with the I²C bus protocol. The TWI controller includes the following features.

- Simultaneous master and slave operation on multiple device systems
- Support for multimaster data arbitration
- 7-bit addressing
- 100K bits/second and 400K bits/second data rates
- General call address support
- Master clock synchronization and support for clock low extension

Architecture

- Separate multiple-byte receive and transmit FIFOs
- Low interrupt rate
- Individual override control of data and clock lines in the event of a bus lockup
- Input filter for spike suppression

Table 12-1 shows the pins for the TWI. Two bidirectional pins externally interface the TWI controller to the I²C bus. The interface is simple and no other external connections or logic are required.

Table 12-1. TWI Pins

Pin	Description
SDA	In/Out TWI serial data, high impedance reset value
SCL	In/Out TWI serial clock, high impedance reset value

Architecture

Figure 12-1 illustrates the overall architecture of the TWI controller.

The peripheral interface supports the transfer of 32-bit wide data and is used by the processor in the support of register and FIFO buffer reads and writes.

The register block contains all control and status bits and reflects what can be written or read as outlined by the programmer's model. Status bits can be updated by their respective functional blocks.

The FIFO buffer is configured as a 1-byte-wide, 2-deep transmit FIFO buffer and a 1-byte-wide, 2-deep receive FIFO buffer.

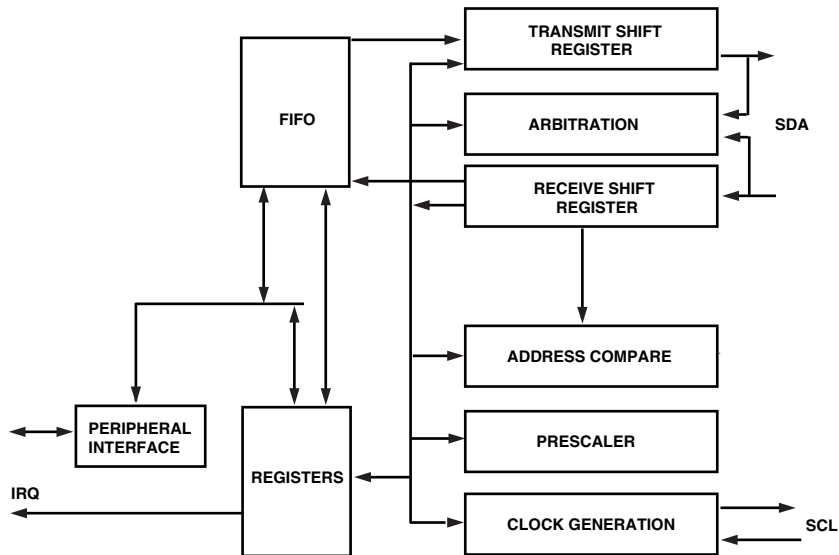


Figure 12-1. TWI Block Diagram

The transmit shift register serially shifts its data out externally off chip. The output can be controlled to generate acknowledgements or it can be manually overwritten.

The receive shift register receives its data serially from off chip. The receive shift register is 1 byte wide and data received can either be transferred to the FIFO buffer or used in an address comparison.

The address compare block supports address comparison in the event the TWI controller module is accessed as a slave.

The prescaler block must be programmed to generate a 10 MHz time reference relative to the peripheral clock. This time base is used for filtering data and timing events specified by the electrical parameters in the data sheet (see the I²C bus specification from Philips), as well as for SCL clock generation.

Register Descriptions

The clock generation module is used to generate an external serial clock (SCL) when in master mode. It includes the logic necessary for synchronization in a multimaster clock configuration and clock stretching when configured in slave mode.

Register Descriptions

The TWI controller has 16 registers which are described in the following sections. More information on these registers can be found in [“Two Wire Interface Registers” on page A-130](#).

TWI Master Internal Time Register

The TWI control register (TWIMTR) is used to enable the TWI module as well as to establish a relationship between the peripheral clock (PCLK) and the TWI controller’s internally-timed events. The internal time reference is derived from PCLK using a prescaled value.

$$\text{PRESCALE} = f_{\text{PCLK}}/10 \text{ MHz}$$

Additional information for the TWIMTR register bits includes:

TWI Enable (TWIEN). This bit must be set for slave or master mode operation. It is recommended that this bit be set and remain set at the time PRESCALE is initialized. This guarantees accurate operation of bus busy detection logic.

Prescale (PRESCALE). This value should be set to the number of peripheral clock periods that equals the time period corresponding to a 10 MHz frequency. This number is represented as a 7-bit binary value.

TWIDIV Register

During master mode operation, the serial clock divider register (TWIDIV) values are used to create the high and low durations of the serial clock (SCL). Serial clock frequencies can vary from 400 kHz to less than 20 kHz. The resolution of the generated clock is 1/10 MHz or 100 ns.

$$\text{CLKDIV} = \text{TWI SCL period} \div 10 \text{ MHz time reference}$$

For example, for an SCL of 400 kHz (period = 1/400 kHz = 2500 ns) and an internal time reference of 10 MHz (period = 100 ns):

$$\text{CLKDIV} = 2500 \text{ ns} \div 100 \text{ ns} = 25$$

For an SCL with a 30% duty cycle, then CLKLOW = 17 and CLKHI = 8.

Note that CLKLOW and CLKHI add up to CLKDIV.

Additional information for the TWIDIV register bits can be found in [“Clock Divider Register \(TWIDIV\)” on page A-132](#).

Slave Mode Control Register

The TWI slave mode control register (TWISCTL) controls the logic associated with slave mode operation. Settings in this register do not affect master mode operation and should not be modified to control master mode functionality.

Additional information for the TWISCTL register bits can be found in [“Slave Mode Control Register \(TWISCTL\)” on page A-133](#).

Slave Mode Address Register

The TWI slave mode address register (TWISADDR) holds the slave mode address, which is the valid address that the slave-enabled TWI controller responds to. The TWI controller compares this value with the received address during the addressing phase of a transfer. [For more information, see “Slave Address Register \(TWISADDR\)” on page A-135.](#)

Slave Mode Status Register

During and at the conclusion of slave mode transfers, the TWI slave mode status register (TWISSTAT) holds information on the current transfer. Generally, slave mode status bits are not associated with the generation of interrupts. Master mode operation does not affect slave mode status bits. [For more information, see “Slave Status Register \(TWISSTAT\)” on page A-135.](#)

Master Mode Control Register

The TWI master mode control register (TWIMCTL) controls the logic associated with master mode operation. Bits in this register do not affect slave mode operation and should not be modified to control slave mode functionality. [For more information, see “Master Control Register \(TWIMCTL\)” on page A-136.](#)

Master Mode Address Register

During the addressing phase of a transfer, the TWI controller, with its master enabled, transmits the contents of the TWI master mode address register (TWIMADDR). When programming this register, omit the read/write bit. That is, only the upper 7 bits that make up the slave address should be written to this register. For example, if the slave address is 1010000, then the TWI_MASTER_ADDR register is programmed with 1010000, which corresponds to 0x50. When sending out the address on the bus, the TWI

controller appends the read/write bit as appropriate based on the state of the `MDIR` bit in the master mode control register. [For more information, see “Master Address Register \(TWIMADDR\)” on page A-139.](#)

Master Mode Status Register

The TWI master mode status register (`TWIMSTAT`) holds information during master mode transfers and at their conclusion. Generally, master mode status bits are not directly associated with the generation of interrupts but offer information on the current transfer. Slave mode operation does not affect master mode status bits. This is a read-only register. [For more information, see “Master Status Register \(TWIMSTAT\)” on page A-140.](#)

FIFO Control Register

The TWI FIFO control register (`TWIFIOCTL`) affect only the FIFO and is not tied in any way with master or slave mode operation. [For more information, see “FIFO Control Register \(TWIFIOCTL\)” on page A-143.](#)

FIFO Status Register

The fields in the TWI FIFO status register (`TWIFIFOSTAT`) indicate the state of the FIFO buffers’ receive and transmit contents. The FIFO buffers do not discriminate between master data and slave data. By using the status and control bits provided, the FIFO can be managed to allow simultaneous master and slave operation. All bits in this register are read-only. [For more information, see “FIFO Status Register \(TWIFIFOSTAT\)” on page A-145.](#)

Interrupt Source Register

The TWI interrupt source register (`TWIIIRPTL`) contains information about functional areas requiring servicing. Many of the bits in this register serve as an indicator to further read and service various status registers.

Register Descriptions

After servicing the interrupt source associated with a bit, programs must clear that interrupt source bit. All bits are sticky and W1C-type. [For more information, see “Interrupt Source Register \(TWIIRPTL\)” on page A-147.](#)

Interrupt Enable Register

The TWI interrupt enable register (TWIIMASK) allows interrupt sources to assert the interrupt output. Each enable bit corresponds with one interrupt source bit in the TWI interrupt source register (TWIIRPTL). Reading and writing the TWI interrupt enable register does not affect the contents of the TWI interrupt source register. For all bits, 0 = interrupt generation disabled and 1 = interrupt generation enabled. [For more information, see “Interrupt Enable Register \(TWIIMASK\)” on page A-150.](#)

8-Bit Transmit FIFO Register

The TWI 8-bit transmit FIFO register (TXTWI8) holds an 8-bit data value written into the FIFO buffer. Transmit data is entered into the corresponding transmit buffer in a first-in, first-out order. Although peripheral bus writes are 32 bits, a write access to the TXTWI8 register adds only one transmit data byte to the FIFO buffer. With each access, the transmit status (TWITXS) field in the TWIFIFOSTAT register is updated. If an access is performed while the FIFO buffer is full, the core waits until there is at least one byte space in the transmit FIFO buffer and then completes the write access. The bits in this register are write-only. [For more information, see “8-Bit Transmit FIFO Register \(TXTWI8\)” on page A-152.](#)

16-Bit Transmit FIFO Register

The TWI 16-bit FIFO transmit register (TXTWI16) holds a 16-bit data value written into the FIFO buffer. Although peripheral bus writes are 32 bits, a write access to the TXTWI16 register adds only two transmit data bytes to the FIFO buffer. To reduce interrupt output rates and peripheral

bus access times, a double byte transfer data access can be performed. Two data bytes can be written, effectively filling the transmit FIFO buffer with a single access.

The data is written in little-endian byte order as shown in [Figure 12-2](#), where byte 0 is the first byte to be transferred and byte 1 is the second byte to be transferred. With each access, the transmit status (TWITXS) field in the TWIFIFOSTAT register is updated. If an access is performed while the FIFO buffer is not empty, the core waits until the FIFO buffer is completely empty and then completes the write access. All bits in this register are write-only. This register always reads as 0x00000000. [For more information, see “16-Bit Transmit FIFO Register \(TXTWI16\)” on page A-153.](#)

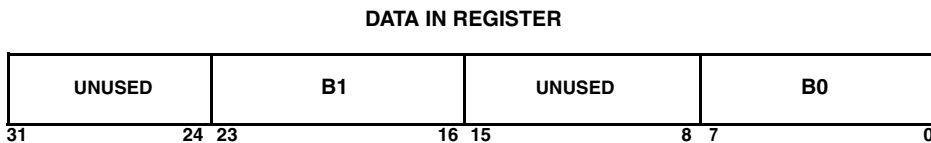


Figure 12-2. Little-Endian Byte Order

8-Bit Receive FIFO Register

The TWI 8-bit FIFO receive register (RXTWI8) holds an 8-bit data value read from the FIFO buffer. Receive data is read from the corresponding receive buffer in a first-in, first-out order. Although peripheral bus reads are 32 bits, a read access to the RXTWI8 register can only access one receive data byte from the FIFO buffer. With each access, the receive status (TWIRXS) field in the TWIFIFOSTAT register is updated. If an access is performed while the FIFO buffer is empty, the core waits until there is at least one byte in the receive FIFO buffer and then completes the read access. All bits in this register are read-only. [For more information, see “8-Bit Receive FIFO Register \(RXTWI8\)” on page A-154.](#)

16-Bit Receive FIFO Register

The TWI 16-bit FIFO receive register (RXTWI16) holds a 16-bit data value read from the FIFO buffer. Although peripheral bus reads are 32 bits, a read access to the RXTWI16 register can only access two receive data bytes from the FIFO buffer. To reduce interrupt output rates and peripheral bus access times, a double-byte receive data access can be performed. Two data bytes can be read, effectively emptying the receive FIFO buffer with a single access.

The data is read in little-endian byte order, as shown in [Figure 12-2 on page 12-9](#), where byte 0 is the first byte received and byte 1 is the second byte received. With each access, the receive status (TWIRXS) field in the TWIFIFOSTAT register is updated to indicate it is empty. If an access is performed while the FIFO buffer is not full, the core waits until the receive FIFO buffer is full and then completes the read access. All bits in this register are write-only. [For more information, see “16-Bit Receive FIFO Register \(RXTWI16\)” on page A-154.](#)

Data Transfer Mechanics

The TWI controller follows the transfer protocol of the *Philips I²C Bus Specification version 2.1* dated January 2000. A simple complete transfer is diagrammed in [Figure 12-3](#).

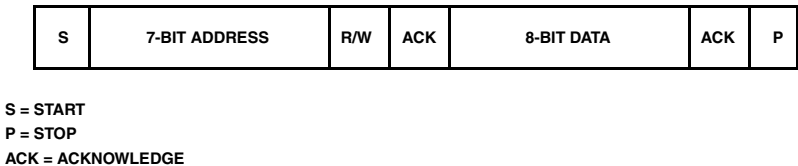
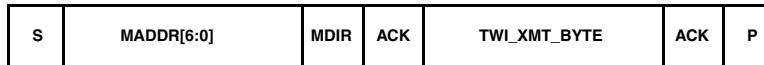


Figure 12-3. Basic Data Transfer

To better understand the mapping of TWI controller register contents to a basic transfer, [Figure 12-4](#) details the same transfer as above noting the corresponding TWI controller bit names. In this illustration, the TWI controller successfully transmits one byte of data. The slave has acknowledged both address and data.



S = START

P = STOP

ACK = ACKNOWLEDGE

Figure 12-4. Data Transfer With Bit Illustration

Clock Generation and Synchronization

The TWI controller only issues a clock during master mode operation and only at the time a transfer has been initiated. If arbitration for the bus is lost, the serial clock output immediately three-states. If multiple clocks attempt to drive the serial clock line, the TWI controller synchronizes its clock with the other remaining clocks. This is illustrated in [Figure 12-5](#).

The TWI controller's serial clock (SCL) output follows these rules:

- Once the clock high (CLKHI) count is complete, the serial clock output is driven low and the clock low (CLKLOW) count begins.
- Once the clock low count is complete, the serial clock line is three-stated and the clock synchronization logic enters into a delay mode (shaded area) until the SCL line is detected at a logic 1 level. At this time, the clock high count begins.

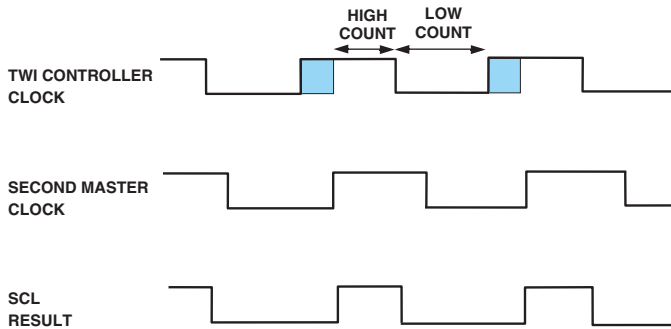


Figure 12-5. TWI Clock Synchronization

Bus Arbitration

The TWI controller initiates a master mode transmission (TWIMEN) only when the bus is idle. If the bus is idle and two masters initiate a transfer, arbitration for the bus begins. This is illustrated in [Figure 12-6](#).

The TWI controller monitors the serial data bus (SDA) while SCL is high. If SDA is determined to be an active logic 0 level while the internal TWI controller's data is a logic 1 level, the TWI controller has lost arbitration and ends generation of clock and data. Note that arbitration is performed not only at serial clock edges, but also during the entire time SCL is high.

Start and Stop Conditions

Start and stop conditions involve serial data transitions while the serial clock is at logic 1 level. The TWI controller generates and recognizes these transitions. Typically, start and stop conditions occur at the beginning and at the conclusion of a transmission, with the exception of repeated start “combined” transfers, as shown in [Figure 12-7](#).

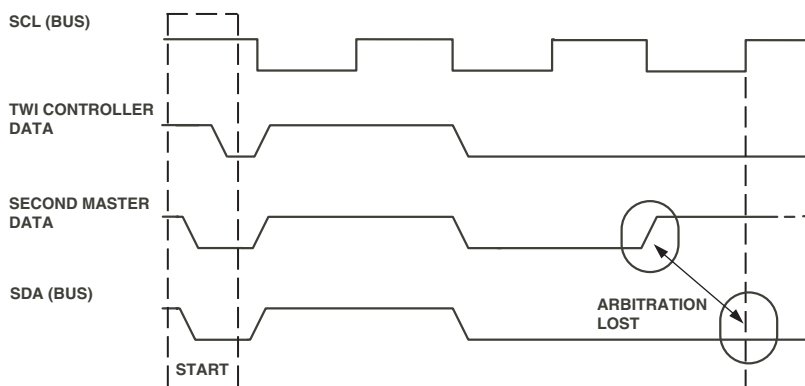


Figure 12-6. TWI Bus Arbitration

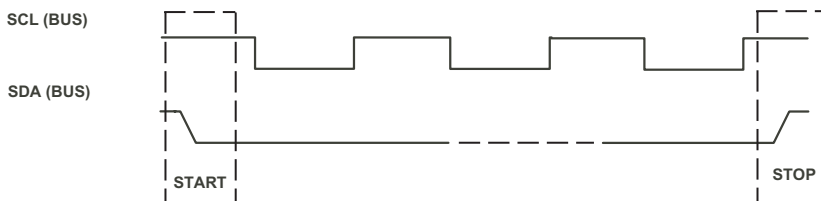


Figure 12-7. TWI Start and Stop Conditions

The TWI controller's special-case start and stop conditions include:

- TWI controller addressed as a slave-receiver

If the master asserts a stop condition during the data phase of a transfer, the TWI controller concludes the transfer (TWISCOMP).

Data Transfer Mechanics

- TWI controller addressed as a slave-transmitter

If the master asserts a stop condition during the data phase of a transfer, the TWI controller concludes the transfer (TWISCOMP) and indicates a slave transfer error (TWISERR).

- TWI controller as a master-transmitter or master-receiver

If the stop bit is set during an active master transfer, the TWI controller issues a stop condition as soon as possible to avoid any error conditions (as if data transfer count had been reached).

General Call Support

The TWI controller always decodes and acknowledges a general call address if it is enabled as a slave (TWISEN) and if general call is enabled (TWIGC). General call addressing (0x00) is indicated by the setting of the GCALL bit, and by the nature of the transfer, the TWI controller is a slave-receiver. If the data associated with the transfer is to be not acknowledged (NAKed), the TWINAK bit can be set.

If the TWI controller is to issue a general call as a master-transmitter, the appropriate address and transfer direction can be set along with loading transmit FIFO data.

Fast Mode

Fast mode essentially uses the same mechanics as standard mode. It is the electrical specifications and timing that are different. When fast mode is enabled (TWIFAST), the following timings are modified to meet the electrical requirements.

- Serial data rise times before arbitration evaluation (t_r)
- Stop condition setup time from serial clock to serial data (t_{SUSTO})
- Bus free time between a stop and start condition (t_{BUF})

Programming Examples

The following sections include programming examples for general setup, slave mode, and master mode, as well as guidance for repeated start conditions. For an example of programming the TWI using the digital peripheral interface and SRU2, see [“Configuring the Two Wire Interface” on page 4-73](#).

General Setup

General setup refers to register writes that are required for both slave mode and master mode operation. General setup should be performed before either the master or slave enable bits are set.

Programs should enable the TWI controller through the `TWIMITR` register and set the prescale value. Program the prescale value to the binary representation of $f_{\text{PCLK}}/10 \text{ MHz}$.

All values should be rounded up to the next whole number. The `TWIEN` enable bit must be set. Note that once the TWI controller is enabled, a bus busy condition may be detected.

Slave Mode

When enabled, slave mode supports both receive and transmit data transfers. It is not possible to enable only one data transfer direction and not acknowledge (NAK) the other. This is reflected in the following setup.

1. Program the `TWISADDR` register. The appropriate 7 bits are used in determining a match during the address phase of the transfer in case of 7-bit addressing.

Programming Examples

2. Program the `TXTWI8` or `TXTWI16` register. These are the initial data values to be transmitted in the event the slave is addressed as a transmitter. This is an optional step. If no data is written and the slave is addressed and a transmit is required, the serial clock (`SCL`) is stretched and an interrupt is generated.
3. Program the `TWIFIFOCTL` register. Indicate if transmit (or receive) FIFO buffer interrupts should occur with each byte transmitted (received) or with each 2 bytes transmitted (received).
4. Program the `TWIIMASK` register. Enable bits associated with the desired interrupt sources. As an example, programming the value `0x000F` results in an interrupt output to the processor when a valid address match is detected, a valid slave transfer completes, a slave transfer has an error, or a subsequent transfer has begun but the previous transfer has not been serviced.
5. Program the `TWISCTL` register. This prepares and enables slave mode operation. As an example, programming the value `0x0005` enables slave mode operation, requires 7-bit addressing, and indicates that data in the transmit FIFO buffer is intended for slave mode transmission.

Table 12-2 shows what the interaction between the TWI controller and the processor might look like when the slave is addressed as a receiver.

Table 12-2. Slave Mode Setup Interaction (Slave Addressed as Receiver)

TWI Controller Master	Processor
Interrupt: <code>TWISINIT</code> – Slave transfer has been initiated.	Acknowledge: Clear interrupt source bits.
Interrupt: <code>TWIRXS</code> – Receive buffer has 1 or 2 bytes (according to <code>TWIRXINT</code>).	Read receive FIFO buffer. Acknowledge: Clear interrupt source bits.
...	...
Interrupt: <code>TWISCOMP</code> – Slave transfer complete.	Read receive FIFO buffer. Acknowledge: Clear interrupt source bits.

Master Mode Clock Setup

Master mode operation is set up and executed on a per-transfer basis. An example of programming steps for a receive and for a transmit are given separately in following sections. The clock setup programming step listed here is common to both transfer types.

Program the `TWIDIV` register. This defines the clock high duration and clock low duration.

Master Mode Transmit

Follow these programming steps for a single master mode transmit:

1. Program the `TWIMADDR` register. This defines the address transmitted during the address phase of the transfer.
2. Program the `TXTWI8` or `TXTWI16` registers. This is the initial data transmitted. It is considered an error to complete the address phase of the transfer and not have data available in the transmit FIFO buffer.
3. Program the `TWIFIFOCTL` register. Indicate if transmit FIFO buffer interrupts should occur with each byte transmitted (8 bits) or with each 2 bytes transmitted (16 bits).
4. Program the `TWIIMASK` register. Enable bits associated with the desired interrupt sources. As an example, programming the value `0x0030` results in an interrupt output to the processor in the event that the master transfer completes, or if the master transfer has an error.

Programming Examples

5. Program the `TWIMCTL` register. This prepares and enables master mode operation. As an example, programming the value `0x0201` enables master mode operation, generates a 7-bit address, sets the direction to master-transmit, uses standard mode timing, and transmits 8 data bytes before generating a stop condition.

Table 12-3 shows what the interaction between the TWI controller and the processor might look like using this example.

Table 12-3. Master Mode Transmit Setup Interaction

TWI Controller Master	Processor
Interrupt: <code>TWITXINT</code> – Transmit buffer has 1 or 2 bytes empty (according to <code>XMTINTLEN</code>).	Write transmit FIFO buffer. Acknowledge: Clear interrupt source bits.
...	...
Interrupt: <code>TWIMCOMP</code> – Master transfer complete.	Acknowledge: Clear interrupt source bits.

Master Mode Receive

Follow these programming steps for a single master mode transmit:

1. Program the `TWIMADDR` register. This defines the address transmitted during the address phase of the transfer.
2. Program the `TWIFIFOCTL` register. Indicate if receive FIFO buffer interrupts should occur with each byte received (8 bits) or with each 2 bytes received (16 bits).
3. Program the `TWIIMASK` register. Enable bits associated with the desired interrupt sources. For example, programming the value `0x0030` results in an interrupt output to the processor in the event that the master transfer completes, and the master transfer has an error.

4. Program the `TWIMCTL` register. Ultimately this prepares and enables master mode operation. As an example, programming the value `0x0201` enables master mode operation, generates a 7-bit address, sets the direction to master-receive, uses standard mode timing, and receives 8 data bytes before generating a stop condition.

Table 12-4 shows what the interaction between the TWI controller and the processor might look like using this example.

Table 12-4. Master Mode Receive Setup Interaction

TWI Controller Master	Processor
Interrupt: <code>TWIRXINT</code> – Receive buffer has 1 or 2 bytes (according to <code>RCVINTLEN</code>).	Read receive FIFO buffer. Acknowledge: Clear interrupt source bits.
...	...
Interrupt: <code>TWIMCOMP</code> – Master transfer complete.	Read receive FIFO buffer. Acknowledge: Clear interrupt source bits.

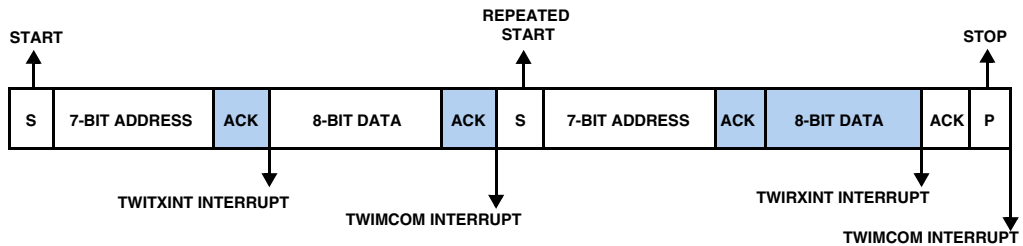
Repeated Start Condition

In general, a repeated start condition is the absence of a stop condition between two transfers initiated by the same master. The two transfers can be of any direction type. Examples include a transmit followed by a receive, or a receive followed by a transmit. During a repeated start transfer, each interrupt must be serviced correctly to avoid errors. The following sections are intended to assist the programmer with service routine development.

Transmit/Receive Repeated Start Sequence

Figure 12-8 illustrates a repeated start data transmit followed by a data receive sequence.

Programming Examples



Shaded region indicates slave to master transmission.

Figure 12-8. Transmit/Receive Data Repeated Start

The tasks performed at each interrupt are:

- TWITXINT interrupt

This interrupt is generated every time the transmit FIFO has one or two byte locations available to be written. To service this interrupt, write a byte or word into the transmit FIFO registers (TXTWI8 or TXTWI16). During one of these interrupts (preferably the first time), do the following:

- Set the RSTART bit (or earlier when TWIMCTL register is programmed first).
- Set the TWIMDIR bit to indicate the next transfer direction is receive. This should be done before the addressing phase of the next transfer begins.

- TWIMCOMP interrupt

This interrupt is generated because all data has been transferred (DCNT = 0). If no errors were generated, a start condition is initiated. At this time, program the following bits of TWI_MASTER_CTRL register:

- Clear RSTART (if this is the last transfer).

- Re-program `DCNT` with the desired number of bytes to receive.
- `TWISERR` interrupt

This interrupt is generated due to the arrival of a byte into the receive FIFO. Simple data handling is all that is required.

Receive/Transmit Repeated Start Sequence

Figure 12-9 illustrates a repeated start data receive followed by a data transmit sequence. The shading in the figure indicates the slave has the bus.

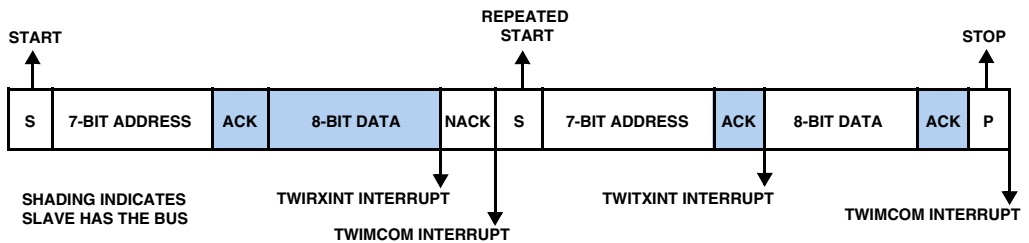


Figure 12-9. Receive/Transmit Data Repeated Start

The tasks performed at each interrupt are:

- `TWIRXINT` interrupt

This interrupt is generated due to the arrival of one or two data bytes into the receive FIFO. The `TWIRSTART` bit should be set at this time (or earlier) and `MDIR` should be cleared to reflect the change in direction of the next transfer. The `TWIMDIR` bit must be cleared before the addressing phase of the subsequent transfer begins.

Electrical Specifications

- TWIMCOMP interrupt

This interrupt has occurred due to the completion of the data receive transfer. At this time the data transmit transfer begins. The TWIDCNT field should be set to reflect the number of bytes to be transmitted. Clear the TWIRSTART bit if this is the last transfer.

- TWITXINT interrupt

This interrupt is generated when there is one or two bytes of empty space in the FIFO. Simple data handling is all that is required.

- TWIMCOM interrupt

The transfer is complete.

Electrical Specifications

All logic complies with the electrical specification outlined in the *Philips I²C Bus Specification version 2.1* dated January, 2000.

13 PRECISION CLOCK GENERATORS

The precision clock generators (PCG) consist of four units, each of which generates a pair of signals (clock and frame sync) derived from a clock input signal. The units, A B, C, and D, are identical in functionality and operate independently of each other. The two signals generated by each unit are normally used as a serial bit clock/frame sync pair.

Note the definitions of various clock periods that are a function of $CLKIN$ and the appropriate ratio control ([Table 13-1](#)).

Table 13-1. Clock Periods

Timing Requirements	Description
t_{CK}	CLKIN clock period
t_{CCLK}	Processor core clock period
t_{PCLK}	Peripheral clock period = $2 \times t_{CCLK}$

The unit that generates the bit clock is relatively simple, since digital clock signals are usually regular and symmetrical. The unit that generates the frame sync output, however, is designed to be extremely flexible and capable of generating a wide variety of framing signals needed by many types of peripherals that can be connected to the signal routing unit (SRU1).

[For more information, see “Signal Routing Units” on page 4-8.](#)

The core phase-locked loop (PLL) has been designed to provide clocking for the processor core. Although the performance specifications of this PLL are appropriate for the core, they have not been optimized or specified for precision data converters where jitter directly translates into time quantization errors and distortion.

As shown in [Figure 13-1](#), the PCGs can accept clock inputs either directly from the external oscillator (or discrete crystal) connected to the `CLKIN` pin, from the peripheral clock (`PCLK`), or from any of the 20 DAI pins. This allows a design to contain an external clock with performance specifications appropriate for the application target.

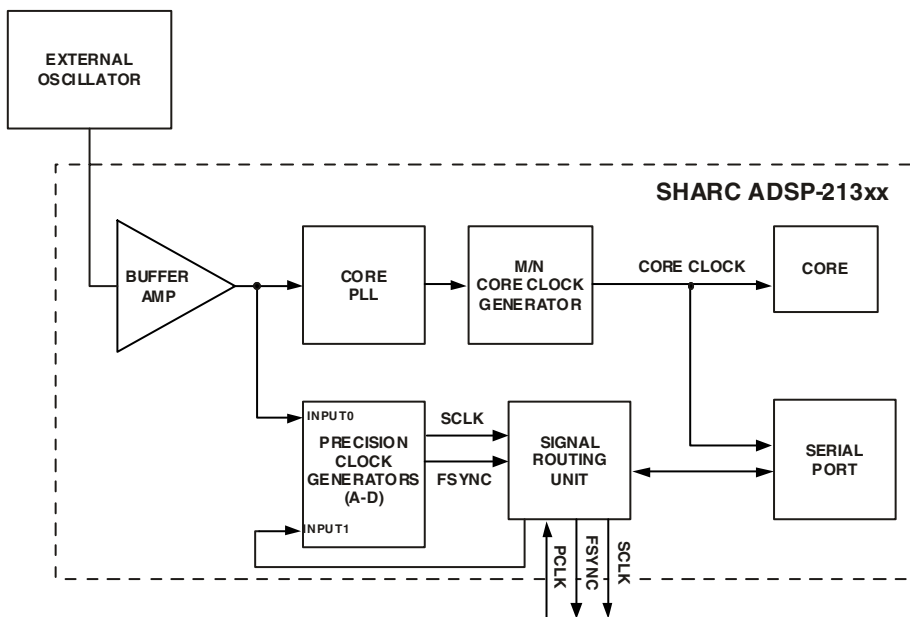



Figure 13-1. Clock Inputs

Note that clock and frame sync signals generated by the serial ports are also subject to these jitter problems because the SPORT clock is generated from the core clock. However, a SPORT can produce data output while

being a clock and frame sync slave. The clock generated by the SPORT is sufficient for most of the serial communications, but it is suboptimal for analog/digital conversion. Therefore, all precision data converters should be synchronized to a clock generated by the PCG or to a clean (low jitter) clock that is fed into SRU1 off-chip through a pin.

 Any clock or frame sync unit should be disabled (have its enable bit cleared) before changing any of the associated parameters.

Clock Outputs

Each of the four units (A, B, C, and D) produces a clock output and a frame sync output. The clock output is derived from the input to the PCG with a 20-bit divisor as shown in the following equation.

$$\text{Frequency of Clock Output} = \frac{\text{Frequency of Clock Input}}{\text{Clock Divisor}}$$

If the divisor is zero or one, the PCG's clock generation unit is bypassed, and the clock input is connected directly to the clock output. Otherwise, the PCG unit clock output frequency is equal to the input clock frequency, divided by a 20-bit integer. This integer is specified in bits 19–0 of the `PCG_CTLx1` registers for units A, B, C and D respectively. These registers and bits are also described in [Table A-63 on page A-156](#) and [Table A-64 on page A-157](#).

The clock outputs have four other control bits that enable the A, B, C, and D units, `ENCLKA`, `ENCLKB`, `ENCLKC`, and `ENCLKD` respectively (bits 31 of the `PCG_CTLx0` registers). These bits enable (= 1) and disable (= 0) the clock output signal for units A, B, C, and D respectively. When disabled, clock output is held at logic low.

Frame Sync Outputs

The `CLKASOURCE` bit (bit 31 in the `PCG_CTLA1` registers) specifies the input source for the clock of the respective units (A, B, C, and D). When this bit is cleared ($= 0$), the input is sourced from the external oscillator, as shown in [Figure 13-1](#). When set ($= 1$), the input is sourced from `SRU1`, as specified in the `PCG_EXT*_I` bits in the `SRU_CLK4` register. The `CLKASOURCE` bit is overridden if `CLKA_SOURCE_IOP` bit (bit 2) in the `PCG_SYNC` register is set. If the `CLKA_SOURCE_IOP` bit is set, the input is sourced from the peripheral clock. See “[Group A Connections—Clock Signals](#)” on page 4-19.

The PCG units B, C, and D function identically, except that the `PCG_CTLB1`, `PCG_CTLC1`, and `PCG_CTLD1` bits (bit 31) indicate that the external source for these units is specified in `PCG_EXTB_I`, `PCG_EXTC_I`, and `PCG_EXTD_I` bits in the `SRU_CLK4` and `SRU_CLK5` registers. See [Figure 4-16](#) on page 4-22 and [Figure 4-17](#) on page 4-23.

Note that the clock output is always set (as closely as possible) to a 50% duty cycle. If the clock divisor is even, the duty cycle of the clock output is exactly 50%. If the clock divisor is odd, then the duty cycle is slightly less than 50%. The low period of the output clock is one input clock period more than the high period of the output clock. For higher values of an odd divisor, the duty cycle is close to 50%.



A PCG clock output cannot be fed to its own input. Setting `SRU_CLK4[4:0] = 0x1C` connects `PCG_EXT*_I` to logic low, not `PCG_CLK*_0`. Setting `SRU_CLK4[9:5] = 0x1D` connects `PCG_EXTB_I` to logic low, not `PCG_CLKB_0`.

Frame Sync Outputs

Each of the four units (A through D) also produces a synchronization signal for framing serial data. The frame sync outputs are much more flexible since they need to accommodate the wide variety of serial protocols used by peripherals.

Frame sync generation from a unit is independently enabled and controlled. Sources for the frame sync generation can be either from the crystal buffer output, PCLK, or an external pin source. There is only one external source pin for both frame sync and clock output for a unit. If an external source is selected for both frame sync and clock output for a unit, then they operate on the same input signal. Apart from enable and source select control bits, frame sync generation is controlled by a 20-bit divisor, a 16-bit pulse width control, and a 20-bit phase control.

There are two modes of operation for the PCG frame sync. The divisor field determines if the frame sync operates in normal mode (divisor > 1) or bypass mode (divisor = 0 or 1).

Normal Mode

In normal mode, the frequency of the frame sync output is determined by the divisor where:

$$\text{Frequency of Frame Sync Output} = \left(\frac{\text{Frequency of Clock Input}}{\text{Frame Sync Divisor}} \right)$$

The high period of the frame sync output is controlled by the value of the pulse width control. The value of the pulse width control should be less than the value of the divisor.

The phase of the frame sync output is determined by the value of the phase control. If the phase is zero, then the positive edges of the clock and frame sync coincide, provided the divisors of the clock and frame sync are the same, the source for the clock and frame sync is also the same, and if clock and frame sync are enabled at the same time using a single instruction.

The number of input clock cycles that have already elapsed before the frame sync is enabled is equal to the difference between the divisor and the phase values. If the phase is a small fraction of the divisor, then the frame

Frame Sync Outputs

sync appears to lead the clock. If the phase is only slightly less than the divisor, then the frame sync appears to lag the clock. The frame sync phase should not be greater than the divisor.

Bypass Mode

In bypass mode, the frame sync divisor is either 0 or 1. There are two ways the bypass mode operates, depending on the STROBEA, STROBEB, STROBEC and STROBED bits of the PCG_PW and PCG_PW2 registers of the pulse width control register (PCG_PW_x, see [Table A-66 on page A-159](#)). This is shown below.

- **Direct bypass.** If the STROBEA/B/C/D of the pulse width control register (PCG_PW, PCG_PW2) is reset to 0, then the input is directly passed to the frame sync output, either not inverted or inverted, depending on the INVFS_A, INVFS_B, INVFS_C and INVFS_D bits of the PCG_PW and PCG_PW2 registers.
- **One-shot.** In the bypass mode, if the least significant bit (LSB) of the PCG_PW register is set to 1, then a one-shot pulse is generated. This one-shot-pulse has a duration equal to the period of MISCA2_I for unit A, MISCA3_I for unit B, MISCA4_I for unit C, and MISCA5_I for unit D (see [“Group E Connections—Interrupts and Miscellaneous Signals” on page 4-43](#)). This pulse is generated either at the rising or at the falling edge of the input clock, depending on the value of the INVFS_A, INVFS_B, INVFS_C, and INVFS_D bits of the PCG_PW and PCG_PW2 registers.

Frame Sync Output Synchronization With an External Clock

The frame sync output may be synchronized with an external clock by programming the `PCG_SYNC` and `PCG_SYNC2` registers (shown in [Figure A-77 on page A-160](#)) and the PCG control registers (`PCG_CTLA0-1`, `PCG_CTLB0-1`, `PCG_CTLC0-1`, and `PCG_CTLD0-1`) appropriately. In this mode, the rising edge of the external clock is aligned with that of the frame sync output (shown in [Figure 13-2](#)). The external clock is routed to the PCG block from any of the SRU1 group A source signals through the `SRU_CLK4` and `SRU_CLK5` registers (described in [Table 4-4 on page 4-23](#)).

The synchronization with the external clock is enabled by setting bits 0 and 16 of the `PCG_SYNC` register for frame sync A or B and the `PCG_SYNC2` register for C or D output. The phase must be programmed to 3, so that the rising edge of the external clock is in sync with the frame sync.

Programming should occur in the following order.

1. Program the `PCG_SYNC` and `PCG_SYNC2` and the `PCG_CTLA0-1`, `PCG_CTLB0-1`, `PCG_CTLC0-1`, and `PCG_CTLD0-1` registers appropriately.
2. Enable clock or frame sync, or both.

Since the rising edge of the external clock is used to synchronize with the frame sync, the frame sync output is not generated until a rising edge of the external clock is sensed.

The clock output cannot be aligned with the rising edge of the external clock as there is no phase programmability. Once `CLKA` through `CLKD` have been enabled (by programming bit 1 and bit 17 of the `PCG_SYNC` register for `CLKA` and `CLKB` respectively and, bit 1 and 17 of `PCG_SYNC2` register for `CLKC` and `CLKD` respectively) these outputs are activated when a low-to-high transition is sensed in the external clock (`MISCA4_I`, `MISCA5_I`).

Frame Sync Output Synchronization With an External Clock

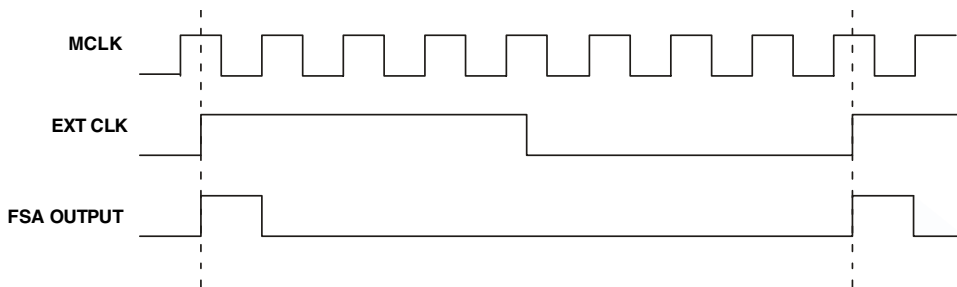


Figure 13-2. Clock Output Synchronization With External Clock

Frame Sync

For a given frame sync, the output is determined by the following:

- **Divisor.** A 20-bit divisor of the input clock that determines the period of the frame sync. When set to 0 or 1, the frame sync operates in bypass mode, otherwise it operates in normal mode.
- **Phase.** A 20-bit value that determines the phase relationship between the clock output and the frame sync output. Settings for phase can be anywhere between 0 to $DIV - 1$.
- **Pulse width.** A 16-bit value that determines the width of the framing pulse. Settings for pulse width can be 0 to $DIV - 1$. If the pulse width is equal to 0 or the frame sync is even, then the actual pulse width of the output frame sync is:

$$Pulse\ Width = \frac{FrameSyncDivisor}{2}$$

For odd divisors the actual pulse width of the output frame sync is:

$$Pulse\ Width = \frac{FrameSyncDivisor - 1}{2}$$

The frequency of the frame sync output is determined by:

$$\frac{\text{Frequency of Clock Input}}{\text{Frame Sync Divisor}}$$

When the divisor is set to any value *other* than 0 or 1, the processors operate in normal mode.

The frame sync divisors ($FS \times DIV$ bits) are specified in bits 19–0 of the corresponding PCG control registers (PCG_CTLx0). The pulse width of the frame sync output is equal to the number of input clock periods specified in the 16-bit field of the corresponding PCG pulse width register (PCG_PW for A and B, PCG_PW2 for C and D). Bits 15–0 specify the pulse width of frame sync A and C, and bits 31–16 specify the pulse width of frame sync B and D.

Phase Shift

Phase shift is a frame sync parameter that defines the phase shift of the frame sync with respect to the clock of the same unit. This feature allows shifting of the frame sync signal in time relative to clock signals. Frame sync phase shifting is often required by peripherals that need a frame sync signal to lead or lag a clock signal.

The amount of phase shifting is specified as a 20-bit value in the $FSAPHASE_HI$ bit field (bits 29–20) of the appropriate PCG_CTLA0 register and in the $FSAPHASE_LO$ bit field (bits 9–0) of the PCG_CTLA1 register for unit A. A single 20-bit value spans these two bit fields. The upper half of the word (bits 19–10) resides in the PCG_CTLA0 register, and the lower half (bits 9–0) resides in the PCG_CTLA1 register.

Similarly, the phase shift for frame syncs B, C, and D is specified in the corresponding PCG_CTLx0 and PCG_CTLx1 registers.

Phase Shift



When using a clock and frame sync as a synchronous pair, the units must be enabled in a single atomic instruction before their parameters are modified. Both units must also be disabled in a single atomic instruction.

Phase Shift Settings

The phase shift between clock and frame sync outputs may be programmed under these conditions:

- The input clock source for the clock generator output and the frame sync generator output is the same.
- Clock and frame sync are enabled at the same time using a single atomic instruction.
- Frame sync divisor is an integral multiple of the clock divisor.

If the phase shift is 0, the clock and frame sync outputs rise at the same time. If the phase shift is 1, the frame sync output transitions one input clock period ahead of the clock transition. If the phase shift is divisor – 1, the frame sync transitions divisor – 1 input clock periods ahead of the clock transitions. This translates to the one input clock period after the clock transition ([Figure 13-3](#)).

Phase shifting is represented as a full 20-bit value so that even when frame sync is divided by the maximum amount, the phase can be shifted to the full range, from zero to one input clock short of the period.

Pulse Width

Pulse width is the number of input clock periods for which the frame sync output is high. Pulse width should be less than the divisor of the frame sync. The pulse width of frame sync A is specified in the `PWFSA` bits (15–0) of the `PCG_PW` register and the pulse width of frame sync B is specified in the `PWFSB` bits (31–16) of the `PCG_PW` register. Similarly, the pulse width of

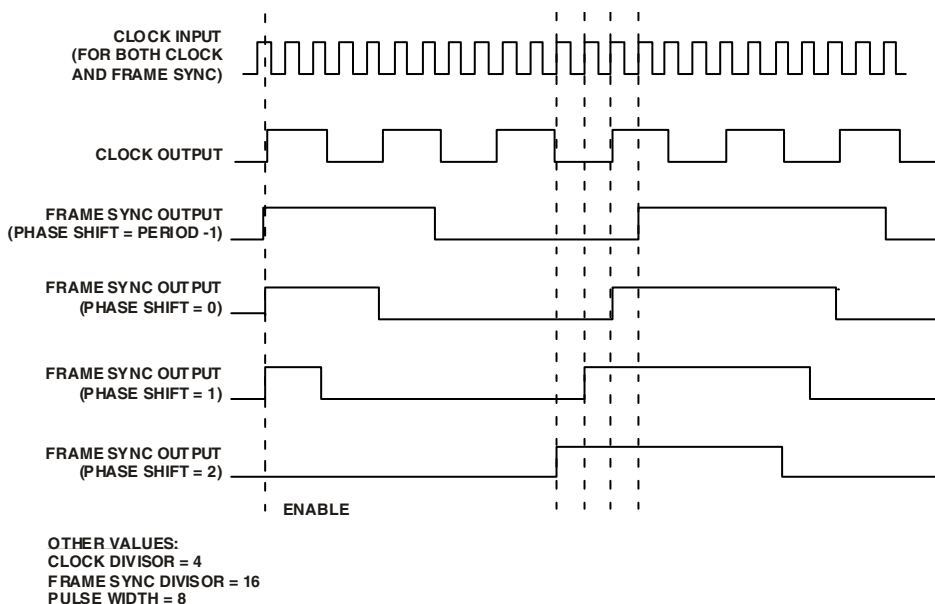


Figure 13-3. Phase Shift Settings

frame sync C is specified in the *WFSC* bits (15–0) of the *PCG_PW2* register and the pulse width of frame sync D is specified in the *WFSD* bits (31–16) of the *PCG_PW2* register.

If the pulse width is equal to 0 or if the divisor is even, then the actual pulse width of the frame sync output is equal to:

$$Pulse\ Width = \frac{FrameSyncDivisor}{2}$$

If the pulse width is equal to 0 or if the divisor is odd, then the actual pulse width of the frame sync output is equal to:

$$Pulse\ Width = \frac{FrameSyncDivisor - 1}{2}$$

Bypass Mode

When the divisor for the frame sync has a value of 0 or 1, the frame sync is in bypass mode, and the `PCG_PW` and `PCG_PW2` registers have different functionality than in normal mode. Two bit fields determine the operation in this mode. The one-shot (which is a strobe pulse) frame sync A, B, C, or D (`STROBEx`) bit (bits 0 and 16 in the `PCG_PW` and `PCG_PW2` registers, respectively) determines if the frame sync has the same width as the input, or of a single strobe. The active low frame sync select for the frame syncs (`INVFSx`) bit (bits 1 and 17 of the `PCG_PW` and `PCG_PW2` registers respectively) determines the nature of the output in the simple bypass and single strobe modes as described below. For additional information about the `PCG_PWx` registers, see [Figure A-76 on page A-159](#).



In bypass mode, bits 15–2 and bits 31–18 of the `PCG_PWx` registers are ignored.

Bypass as a Pass Through

When the `STROBEx` bit in the `PCG_PWx` register equals 0, the unit is bypassed and the output equals the input. If, for example, `INVFSA` (bit 1) for unit A or `INVFSB` (bit 17) for unit B is set, then the signal is inverted (see [Figure 13-4](#)).

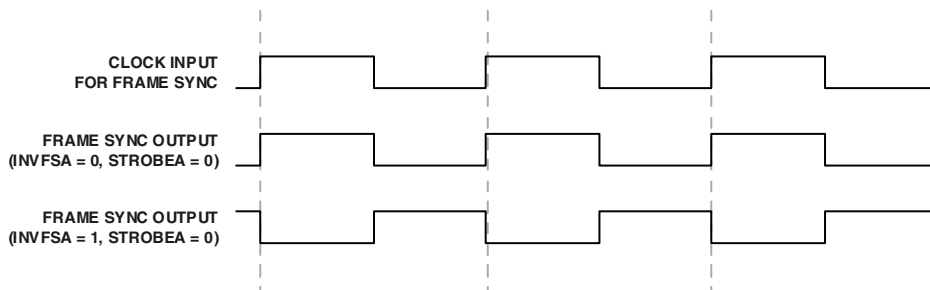


Figure 13-4. Frame Sync Bypass

Bypass mode also enables the generation of a strobe pulse (one-shot). Strobe usage ignores the counter and looks to SRU1 to provide the input signal.

Bypass as a One-Shot

When the STROBEA or STROBE_B bits (bit 0, bit 16 of the PCG_PW register) or STROBE_C or STROBE_D bits (bit 0, bit 16 of the PCG_PW2 register) are set (= 1), the one-shot option is used. When the STROBE_x bit is set (= 1), the frame sync is a pulse with a duration equal to one period, or one full cycle of MISCA2_I for unit A, MISCA3_I for unit B, MISCA4_I for unit C, and MISCA5_I for unit D that repeats at the beginning of every clock input period. This pulse is generated during the high period of the input clock when the INVFS_{A/B/C/D} bits (bits 1 or 17, respectively of the PCG_PW and PCG_PW2 registers) are cleared (INVFS_{A/B}=0) or during the low period of the input clock when invert bits INVFS_{A/B/C/D} are set (= 1).

A *strobe period* is equal to the period of the normal clock input signal specified by the FSASOURCE bit (bit 30 in the PCG_CTLA1 register for unit A) and the corresponding FS_xSOURCE bit (bit 30 in the PCG_CTL_x1 registers for units B, C, and D).

As shown in [Figure 13-5](#), the output pulse width is equal to the period of the SRU1 source signal (MISCA2_I for frame sync A, MISCA3_I for frame sync B, MISCA4_I for frame sync C, and MISCA5_I, for frame sync D). The pulse begins at the second rising edge of MISCA_x_I following a rising edge of the clock input. When the INVFS_{A/B/C/D} bit is set, the pulse begins at the second rising edge of MISCA_x_I, coincident or following a falling edge of the clock input.

For more information, see “Group E Connections—Interrupts and Miscellaneous Signals” on page 4-43.

Programming Examples

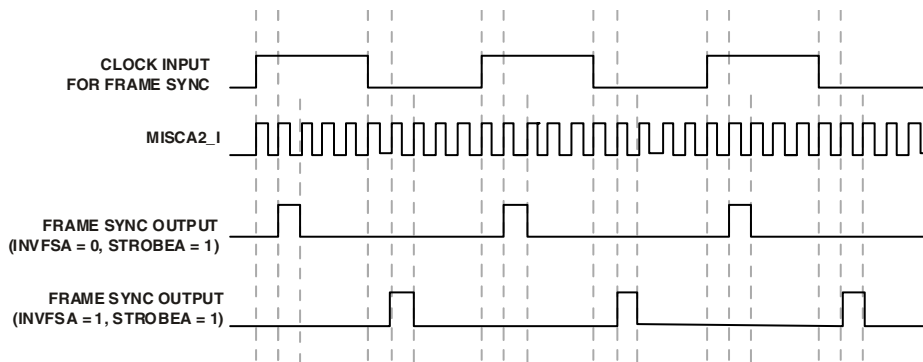


Figure 13-5. One-Shot (Synchronous Clock Input and MISCA2_I)

The second `INVFS` bit (bit 1) of the pulse width control register (`PCG_PW`) determines whether the falling or rising edge is used. When set (`= 1`), this bit selects an active low frame sync, and the pulse is generated during the low period of clock input. When cleared (`= 0`), this bit is set to active high frame sync and the pulse is generated during the high period of clock input. For more information on the `PCG_PWx` registers, refer to [Table A-66, “PCG_PWx Register Bit Descriptions \(in Bypass Mode\),”](#) on page A-159.

Programming Examples

This section contains three programming examples:

1. [“PCG Setup for I2S or Left-Justified DAI”](#) on page 13-15
2. [“Clock and Frame Sync Divisors PCG Channel B”](#) on page 13-20
3. [“PCG Channel A and B Output Example”](#) on page 13-23

PCG Setup for I²S or Left-Justified DAI

This example shows how to set up two precision clock generators using the S/PDIF receiver and an asynchronous sample rate converter (SRC) to interface to an external audio DAC. In this example an input clock (CLKIN) of 33.330 MHz is assumed and the PCG is configured to provide a fixed SRC/DAC output sample rate of 65.098 kHz. The input to the S/PDIF receiver is typically 44.1 kHz if supplied by a CD player but can also be from another source at any nominal sample rate from about 22 kHz to 192 kHz.

Three synchronous clocks are required: a framesync (FSYNC; FS), a master clock (PCGX_CLK; $256 \times FS$), and a serial bit clock (SCLK; $64 \times FS$). Since each PCG has only two outputs, this example requires two PCGs. Furthermore, because the digital audio interface requires a fixed-phase relation between SCLK and FSYNC, these two outputs should come from one PCG while the master clock comes from the other.

The CLKIN = 33.330 MHz is divided by the two PCGs to provide the three synchronous clocks — PCGX_CLK, SCLK, and FSYNC for the SRCs and external DAC. These divisors are stored in 20-bit fields in the PCG_CTL registers. [For more information, see “Precision Clock Generator Registers” on page A-155.](#)

The integer divisors for several possible sample rates based on 33.330 MHz CLKIN are shown in [Table 13-2](#).

Programming Examples

Table 13-2. Precision Clock Generator Division Ratios
(33.330 CLKIN)

Sample Rate kHz)	PCG Divisors		
	PCG CLOCK INPUT	SCLK	FSYNC ¹
130.195	1	4	256
65.098	2	8	512
43.398	3	12	768
32.549	4	16	1024
26.039	5	20	1280
21.699	6	24	1536
18.599	7	28	1792

1 The frame sync divisor should be an even integer in order to produce a 50% duty cycle waveform. See [“Frame Sync Outputs” on page 13-4](#).

For more information, see [“Power Management Control Register \(PMCTL\)” on page A-170](#). The equation and procedure for programming a master clock input is:

1. Set the core clock rate using the values below.

$$PLL_M/PLL_N = CCLK$$

where M = 29, N = 4 for a CCLK of 241.64 MHz

2. Divide CCLK rate by 2 (fixed) to provide a PCLK (peripheral clock) rate of 120.82 MHz.
3. PCLK is divided by 4 (fixed) to provide the PCG_CLKx_0 rate of 30.21 MHz for each of the SRCs.



The combined PCGs can provide a selection of synchronous clock frequencies to support alternate sample rates for the SRCs and external DACs. However, the range of choices is limited by `CLKIN` and the ratio of `PCG_CLKx_0:SCLK:FSYNC` which is normally fixed at 256:64:1 to support digital audio, left-justified, I²S, and right-justified interface modes. Many DACs also support 384, 512, and $786 \times \text{FSYNC}$ for `PCG_CLKx_0`, which allows some additional flexibility in choosing `CLKIN`.

Note also that in all three DAI modes, the falling edge of `SCLK` must always be synchronous with both edges of `FSYNC`. This requires that the phase of the `SCLK` and `FSYNC` for a common PCG be adjustable.

While the frequency of `PCG_CLKx_0` must be synchronous with the sample rate supplied to the external DAC, there is no fixed-phase requirement. For complete timing information, see the processor specific data sheet.

Figure 13-6 shows an example of the internal interconnections between the SPDIF receiver, SRC, and the PCGs. The interconnections are made by programming the signal routing unit. Note that in this example, `CCLK` is set at 242 MHz. This frequency can be adjusted up to the maximum `CCLK` for the chosen processor. Also note that master clock (`MCLK`) is the input source provided for the PCG. This input can come from `CLKIN`, any peripheral output, or from one of the DAI pins.

Programming Examples

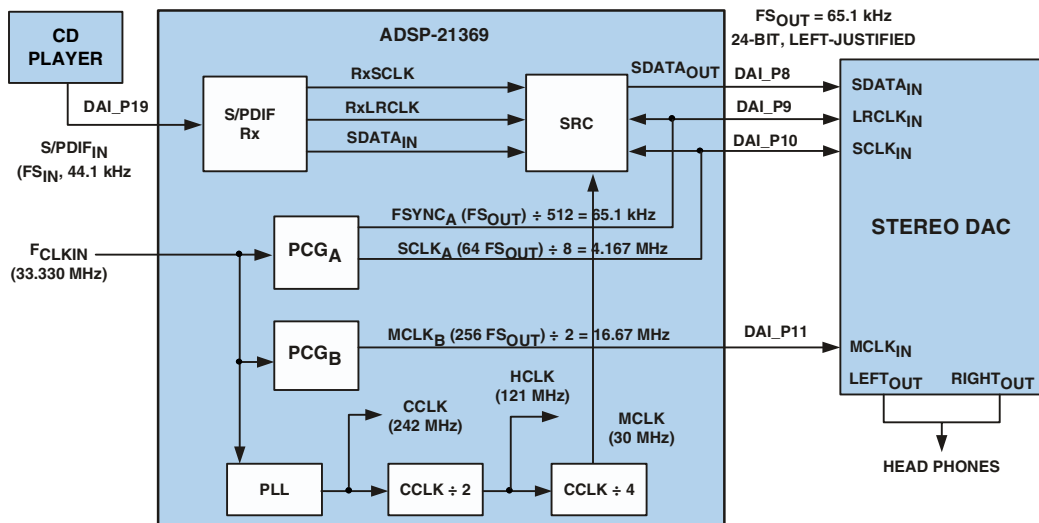


Figure 13-6. PCG Setup for I²S or Left-Justified DAI

In the following example code, the most significant two bits of the control registers (**PCG_CTLx**) specify the clock source and enable the clock generators. Set the clock divisor and source and low phase word first, followed by the control register enable bits, which must be set together. When the **PCG_PW** register is set to 0 (default) the **FS** pulse width is divisor/2 for even divisors and (divisor – 1)/2 for odd divisors. Alternatively, the **PCG_PW** register could be set high for exactly half the period of **CLKIN** cycles for a 50% duty cycle, provided the **FSYNC** divisor is an even number.

Listing 13-1. PCG Initialization

```

/*****
Required Output Sample Rate = 65.098 kHz
Function          Control   Reg      Phase/    Reg Hex
                  reg       Address  Divisor   Contents
FS_A_Ph_Hi/FS_A_Div PCG_CTLA0 0x24C0   0/512    0xC00/00200
FS_A_Ph_Lo/CLK_A_Div PCG_CTLA1 0x24C1   4/8      0x004/00008
-----
FS_B_Ph_Hi/FS_B_Div PCG_CTLB0 0x24C2   -/-      0x800/00000
FS_B_Ph_Lo/CLK_B_Div PCG_CTLB1 0x24C3   0/2      0x000/00002
PW_FS_B/PW_FS_A     PCG_PW    0x24C4   0/0      0x0000:0000
*****/
#include <def21369.h>
/* PCGA --> SCLK & FSYNC Divisors, Sample Rate = 65.098 kHz */
#define PCGA_CLK_DIVISOR 0x0008 /* SCLK output = Fs */
#define PCGA_FS_DIVISOR 0x0200 /* FSYNC output = 64xFs */
#define PCGA_FS_PHASE_L0 0x04 /* Set FSYNC/SCLK Phase for
                               digital audio IF mode */

PCGB --> PCG_CLKx_0 Divisor
#define PCGB_CLK_DIVISOR 0x0002 /* PCG_CLKx_0 output =
                               256xFs */

#define ENCLKA 0x80000000
#define ENFSA 0x40000000
#define PCGB_FS_DIVISOR 0x0000 /* Not used - disabled */
#define PCGB_FS_PHASE_L0 0x00 /* Don't care */
.section/pm seg_pmco; .global Init_PCG;
/*****/
Init_PCG:
/* Set PCGA SCLK & FSYNC Source first to Xtal Buffer and set
   SCLK_A divisor */
r0 = ((PCGA_FS_PHASE_L0 << 20) | PCGA_CLK_DIVISOR);
dm(PCG_CTLA1) = r0;

```

Programming Examples

```
/* Enable PCGA SCLK & FSYNC and set FSYNC_A divisor */
r0 = (ENCLKA | ENFSA | PCGA_FS_DIVISOR);
dm(PCG_CTLA0) = r0;

/* Set PCGB SCLK & FSYNC Source first to Xtal Buffer and set
   SCLK_B divisor */
r0 = ((PCGB_FS_PHASE_LO << 20) | PCGB_CLK_DIVISOR);
dm(PCG_CTLB1) = r0;

/* Enable PCGB SCLK and disable FSYNC_B */
r0 = (ENCLKB | ENFSB | PCGB_FS_DIVISOR);
dm(PCG_CTLB0) = r0;
ustat1 = dm(PCG_CTLB0);
bit clr ustat1 ENFSA;
dm(PCG_CTLB0) = ustat1;

/* Set FSYNC_A and FSYNC_B Pulse Width to 50% Duty Cycle
   (default) */
r0 = 0x00000000;
dm(PCG_PW) = r0;
dm(PCG_SYNC) = r0;
Init_PCG.end:
rts;
```

Clock and Frame Sync Divisors PCG Channel B

This section provides two programming examples written for the ADSP-21369 processor. The first listing, [Listing 13-2](#), uses PCG channel B to output a clock on DAI pin 1 and frame sync on DAI pin 2. The input used to generate the clock and frame sync is CLKIN. This example demonstrates the clock and frame sync divisors, as well as the pulse width and phase shift capabilities of the PCG.

Listing 13-2. PCG Channel B Output Example

```

/* Register Definitions */
#define SRU_CLK4      0x2434
#define SRU_PIN0      0x2460
#define SRU_PBEN0     0x2478
#define PCG_CTLB0     0x24C2
#define PCG_CTLB1     0x24C3
#define PCG_PW        0x24C4

/* SRU Definitions */
#define PCG_CLKB_P     0x39
#define PCG_FSB_P      0x3B
#define PBEN_HIGH_of   0x01

//Bit Positions
#define DAI_PB02       7
#define DAI_PBE02      6
#define PCG_PWB        16

/* Bit Definitions */
#define ENFSB          0x40000000
#define ENCLKB         0x80000000

/* Main code section */
.global _main;
.section/pm seg_pmco;
_main:
/* Route PCG Channel B clock to DAI Pin 1 via SRU */
/* Route PCG Channel B frame sync to DAI Pin 2 via SRU */
r0 = PCG_CLKB_P|(PCG_FSB_P<<DAI_PBE02);
dm(SRU_PIN0) = r0;

```

Programming Examples

```
/* Enable DAI Pins 1 & 2 as outputs */
r0 = PBEN_HIGH_Of|(PBEN_HIGH_Of<<DAI_PB02);
dm(SRU_PBEN0) = r0;

r0 = (100<<PCG_PWB);      /* PCG Channel B FS Pulse width = 100 */
dm(PCG_PW) = r0;

r2 = 1000;                /* Define 20-bit Phase Shift */
r0 = (ENFSB|ENCLKB|      /*Enable PCG Channel B Clock and FS*/
      1000000);          /* FS Divisor = 1000000 */
r1 = lshift r2 by -10;

/* Deposit the upper 10 bits of the Phase Shift in the */
/* correct position in PCG_CTLB0 (Bits 20-29) */
r1 = fdep r1 by 20:10;
r0 = r0 or r1;            /* Phase Shift 19-10 = 0 */
dm(PCG_CTLB0) = r0;

r0 = (100000);            /* Clk Divisor = 100000 */
                          /* Use CLKIN as clock source */
/* Deposit the lower 10 bits of the Phase Shift in the correct
   position in PCG_CTLB1 (Bits 20-29) */

r1 = fdep r2 by 20:10;
r0 = r0 or r1;           /* Phase Shift 9-0 = 0x3E8 */
dm(PCG_CTLB1) = r0;

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

PCG Channel A and B Output Example

[Listing 13-3](#) uses two PCG channels. Channel A is set up to only generate a clock signal. This clock signal is used as the input to channel B through SRU1. The clock and frame sync are routed to DAI pins 1 and 2, respectively, in the same manner as [Listing 13-1](#). The frame sync generated in this example is set for a 50% duty cycle, with no phase shift.

Listing 13-3. PCG Channel A and B Output Example

```
/* Register Definitions */
#define SRU_CLK4      0x2434
#define SRU_PIN0      0x2460
#define SRU_PBEN0     0x2478
#define PCG_CTLA0     0x24C0
#define PCG_CTLA1     0x24C1
#define PCG_CTLB0     0x24C2
#define PCG_CTLB1     0x24C3
#define PCG_PW        0x24C4

/* SRU Definitions */
#define PCG_CLKA_0     0x1c
#define PCG_CLKB_P     0x39
#define PCG_FSB_P      0x3B
#define PBEN_HIGH_0f   0x01

//Bit Positions
#define PCG_EXTB_I      5
#define DAI_PB02        7
#define DAI_PBE02       6
#define PCG_PWB        16
```

Programming Examples

```
/* Bit Definitions */
#define ENCLKA      0x80000000
#define ENFSB       0x40000000
#define ENCLKB      0x80000000
#define CLKBSOURCE  0x80000000
#define FSBSOURCE   0x40000000

/* Main code section */
.global _main;      /* Make main global to be accessed by ISR */
.section/.pm seg_pmco;
_main:
/*Route PCG Channel A clock to PCG Channel B Input via SRU*/
r0 = (PCG_CLKA_0<<PCG_EXTB_I);
dm(SRU_CLK4) = r0;

/* Route PCG Channel B clock to DAI Pin 1 via SRU */
/* Route PCG Channel B frame sync to DAI Pin 2 via SRU */
r0 = (PCG_CLKB_P|(PCG_FSB_P<<DAI_PEB02));
dm(SRU_PIN0) = r0;

/* Enable DAI Pins 1 & 2 as outputs */
r0 = (PBEN_HIGH_0f|(PBEN_HIGH_0f<<DAI_PB02));
dm(SRU_PBEN0) = r0;

r0 = ENCLKA; /* Enable PCG Channel A Clock, No Channel A FS */
          /* FS Divisor = 0, FS Phase 10-19 = 0 */
dm(PCG_CTLA0) = r0;

r1 = 0xfffff; /* Clk Divisor = 0xfffff, FS Phase 0-9 = 0 */
          /* Use CLKIN as clock source */
dm(PCG_CTLA1) = r1;

r0 = (5<<PCG_PWB); /* PCG Channel B FS Pulse width = 1 */
dm(PCG_PW) = r0;
```

```
r0 = (ENFSB|ENCLKB|10); /*Enable PCG Channel B Clock and FS*/  
                        /* FS Divisor = 10, FS Phase 10-19 = 0 */  
dm(PCG_CTLB0) = r0;  
  
r0 = (CLKBSOURCE|FSBSOURCE|10);    /* Clk Divisor = 10 */  
  
/* FS Phase 0-9 = 0, Use SRU_MISC4 as clock source */  
dm(PCG_CTLB1) = r0;  
  
_main.end: jump(pc,0);
```


14 SYSTEM DESIGN

The ADSP-21367/8/9 and ADSP-2137x processors support many system design options. The options implemented in a system are influenced by cost, performance, and system requirements. This chapter provides the following system design information:

- [“Processor Pin Descriptions” on page 14-2](#)
- [“Clock Derivation” on page 14-13](#). Includes [“Power Management Control Register”](#), [“Phase-Locked Loop Startup”](#), [“RESET and CLKIN”](#).
- [“Conditioning Input Signals” on page 14-32](#)
- [“Designing for High Frequency Operation” on page 14-33](#)
- [“Bootting” on page 14-37](#)
- [“Data Delays, Latencies, and Throughput” on page 14-52](#)

Other chapters also discuss system design issues. Some other locations for system design information include:

- [“SPORT Operation Modes” on page 5-10](#)
- [“SPI General Operations” on page 6-8](#)

By following the guidelines described in this chapter, you can ease the design process for your ADSP-21367/8/9 and ADSP-2137x processor product. Development and testing of your application code and hardware can begin without debugging the debug port.



Before proceeding with this chapter it is recommended that you become familiar with the ADSP-21367/8/9 and ADSP-2137x processor's core architecture. This information is presented in the *ADSP-2136x SHARC Processor Programming Reference*.

Processor Pin Descriptions

Refer to the processor-specific data sheet for pin information, including package pinouts for the currently available package options.

Pin Multiplexing

The ADSP-21367/8/9 and ADSP-2137x processors provide the same functionality as other SHARC processors but with a much lower pin count (reducing system cost). Pin multiplexing is used in the following ways.

- Reset output/local clock output/running reset input (ADSP-2137x processors only)

For more information, see “RESETOUT/CLKOUT/RUNRSTIN” on page 14-12.

- External memory interface data (input/output)

For more information, see “Choosing EP Data Mode” on page 14-6.

- PDAP (input only)
- FLAGS (input/output)

For more information, see “Core-Based Flag Pins” on page 14-8.

- PWM channels (output, not available on all models)

See Table 1-1 on page 1-5.

The processors also include the multiplexers for `FLAG0-3` pins. The `FLAG0-2` pins can act as core `FLAG0-2` or `TRQ0-2`, and the `FLAG3` pin can act as a core `FLAG3` or as the `TMREXPEN` signal of the system timer.

[Table 14-1 on page 14-7](#) shows the `SYSCTL` register bit settings for the different data pin functions, and [Figure 14-1](#) and [Figure 14-2](#) show the block diagrams of data pin multiplexing in the ADSP-21367/8/9 and ADSP-2137x processors respectively. Note that:

- In the PDAP control register (`IDP_PP_CTL`), the `IDP_EP_SELECT` bit (bit 26) is the logical AND of the `IDP_PDAP_EN` bit (bit 31). When `IDP_EP_SELECT` is set (= 1), the data bits are read from `DATA31-12` and the control signals come from `DATA11-8`. When `IDP_EP_SELECT` is cleared (= 0), the data bits are read from `DAI_P20-1`. When `IDP_EP_SELECT` is set to 1, the PDAP can be operated through data pins alone (data and controls are completely routed through data pins). For bit descriptions, see [“Parallel Data Acquisition Port Control Register \(`IDP_PP_CTL`\)” on page A-74](#).
- The PDAP, PWM, and memory-to-memory (`MTM_DATA`) signals can be mapped only to the upper bits of the data pins. Therefore, they can be used even when 8- or 16-bit external SRAM is used.
- The `FLAGS` and `PWM` can be mapped (in groups of four) to any of upper 16 data pins. The `FLAGS` alone can be mapped to any of the 32 data pins.
- For PDAP mode, the `SYSCTL` register must be explicitly programmed to put `DATA` pins in PDAP mode. For bit descriptions, see [“System Control Register \(`SYSCTL`\)” on page A-5](#).
- By default, after reset, all data pins are in external memory mode and the `FLAG0-3` pins are in `FLAGS` mode.

Processor Pin Descriptions

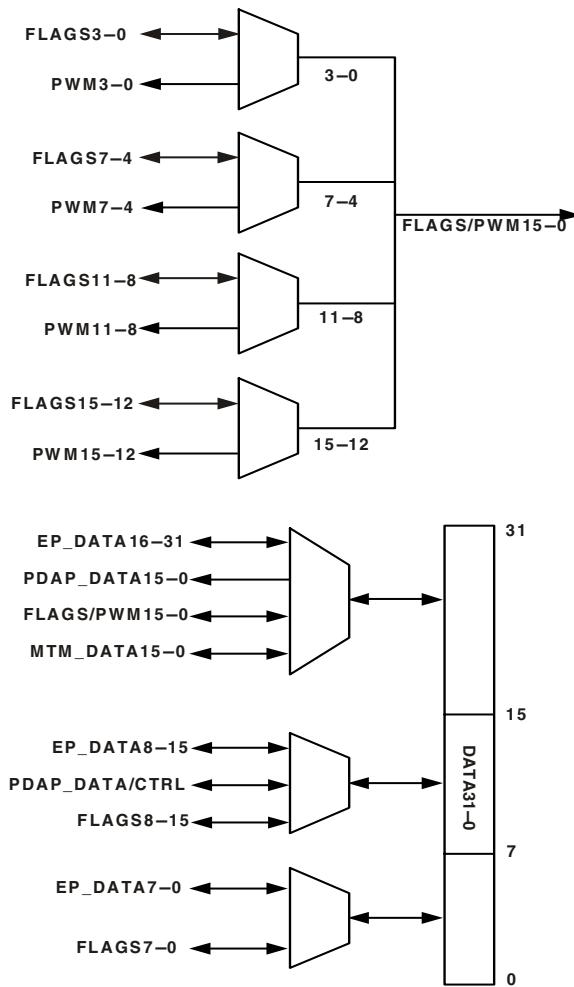


Figure 14-1. Block Diagram of Data Pin Multiplexing (ADSP-2136x)

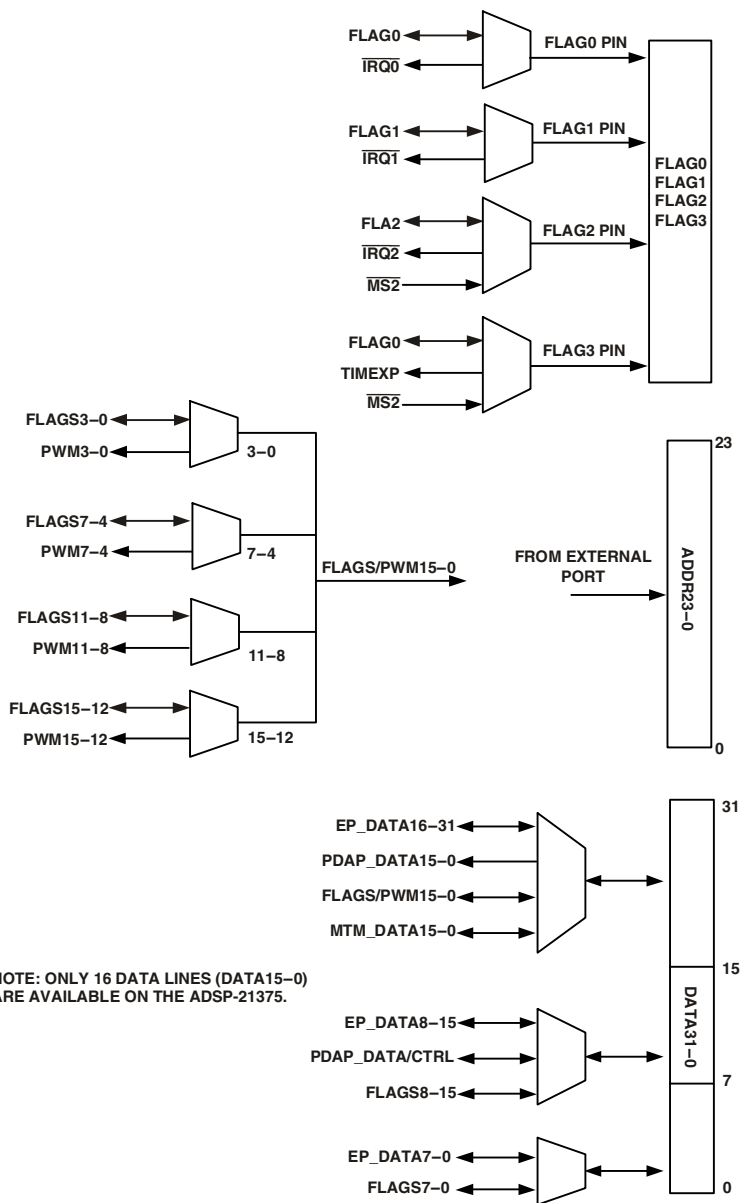


Figure 14-2. Block Diagram of Data Pin Multiplexing (ADSP-2137x)

Choosing EP Data Mode

The $\text{FLAG}/\overline{\text{TRQ}}$ (0, 1, 2, 3) and DATA_{31-0} pins are completely independent. Any mode of programming in one group does not affect the other.

Case 1

If 32-bit external SDRAM/FLASH/SRAM is used, then all data pins should be connected to the memory device so that no other functionality can be programmed in the data pins. In this case, use MODE 0. If flags are required, they can be moved to the DPI SRU2.

Case 2

If 16-bit external SDRAM/FLASH/SRAM is used, and if 16 flag inputs are required, then use MODE 1. Since 16 flags are required, set $\text{FLAG}/\text{PWM_SEL} = 0000$. The flag input mode should be specified in the `FLAGS` register.

Case 3

If 16-bit external SDRAM/FLASH/SRAM is used, and if 8 flag outputs and 8 PWM outputs are required, then use MODE 1. Since 8 flags and PWM outputs are required, set $\text{FLAG}/\text{PWM_SEL} = 1100$. (In this mode, DATA_{31-24} are PWM outputs and DATA_{23-16} are flag outputs.) The flag output mode should be specified in `FLAGS` register.

Case 4

If 8-bit external SDRAM/FLASH/SRAM is used, and if 8 flag inputs, 8 flag outputs, and 8 PWM outputs are required, then use MODE 2. Since 16 flags are required, then all flags on DATA_{31-16} are programmed. This means that there are not enough pins left for the PWM outputs (DATA_{15-8}). Therefore, program the PWM outputs on DATA_{23-16} and program the `FLAGS` on DATA_{31-24} and DATA_{15-8} . Set $\text{FLAGS}/\text{PWM_SEL} = 0011$. The flag I/O direction should be specified in the `FLAGS` register.

Case 5

If no external memory is used, and if the PDAP data lines are connected to DATA pins, and if 8 flags are required, then use MODE 6. Connect the PADP control lines to the DATA pins, and program the flag direction in the `FLAGS` register.

The upper 32 data pins of the external memory interface are muxed (using bits 23–21 in the `SYSTL` register) to support the external memory interface data (input/output), the PDAP (input only), the `FLAGS` (input/output), and the PWM channels (output). [Table 14-1](#) provides the pin settings for these functions.

Table 14-1. Function of Data Pins

SYSTL bits 23–21 Settings	DATA31–16 ¹	DATA15–8	DATA7–0
000	EPDATA32–0 (default at RESET)		
001	FLAGS/PWM15–0 ²	EPDATA15–0	
010	FLAGS/PWM15–0 ¹	FLAGS15–8	EPDATA7–0
011	FLAGS/PWM15–0 ¹	FLAGS15–0	
100	PDAP (DATA + CTRL)		EPDATA7–0
101	PDAP (DATA + CTRL)		FLAGS7–0
110	Reserved		
111	Three-state all pins		

1 Not available on the ADSP-21375 processor.

2 These signals can be `FLAGS` or `PWM` or a mix of both. However, they can be selected only in groups of four. Their function is determined by the control signals `FLAGS/PWM_SEL`.

In PDAP mode, the DATA pins function as follows.

- DATA12–31 pins act as PDAP0–19 (inputs)
- DATA11 pin acts as PDAP_HLD (input)
- DATA10 pin acts as PDAP_CLK (input)

Processor Pin Descriptions

- DATA9 pin is not used
- DATA8 pin acts as PDAP STROBE (output)

Interrupt and Timer Pins

The processor's external interrupt pins, flag pins, and timer pin can be used to send and receive control signals to and from other devices in the system. The $\overline{\text{IRQ2-0}}$ pins are mapped on the FLAG2-0 pins and the TIMEXP pin is mapped on the FLAG3 pin. Hardware interrupt signals ($\overline{\text{IRQ2-0}}$) are received on the FLAG2-0 pins. Interrupts can come from devices that require the processor to perform some task on demand. A memory-mapped peripheral, for example, can use an interrupt to alert the processor that it has data available. [For more information, see Appendix B, Interrupts.](#)

The TIMEXP output is generated by the on-chip timer. It indicates to other devices that the programmed time period has expired. For more information, see the *ADSP-2136x SHARC Processor Programming Reference*.

Core-Based Flag Pins

The FLAG3-0 pins allow single bit signalling between the processor and other devices. For example, the processor can raise an output flag to interrupt a host processor. Each flag pin can be programmed to be either an input or output. In addition, many processor instructions can be conditioned on a flag's input value, enabling efficient communication and synchronization between multiple processors or other interfaces.

The flags are bidirectional pins and all have the same functionality. The FLGX0 bits in the FLAGS register program the direction of each flag pin. For more information, see the *ADSP-2136x SHARC Processor Programming Reference*.



When the `SPIPDN` bit (bit 30 in the `PMCTL` register) is set (= 1 which shuts down the clock to the SPI), the `FLAGx` pins cannot be used (through the `FLAGS7-0` register bits) because the `FLAGx` pins are synchronized with the clock.

Programming Flags

There are 16 system flags and they can be programmed using the `FLAGS` register shown in [Figure 14-3](#). Only four flags are connected to `FLAG/IRQ0-3` pins after reset. If more flags are required, they can be programmed in the `DATA` pins (using the `SYSCTL` register, see [Table A-1 on page A-6](#)) or they can be programmed with the `SRU2` pins (using `SRU2` registers, see [“DPI/SRU2 Connection Groups” on page 4-51](#)).

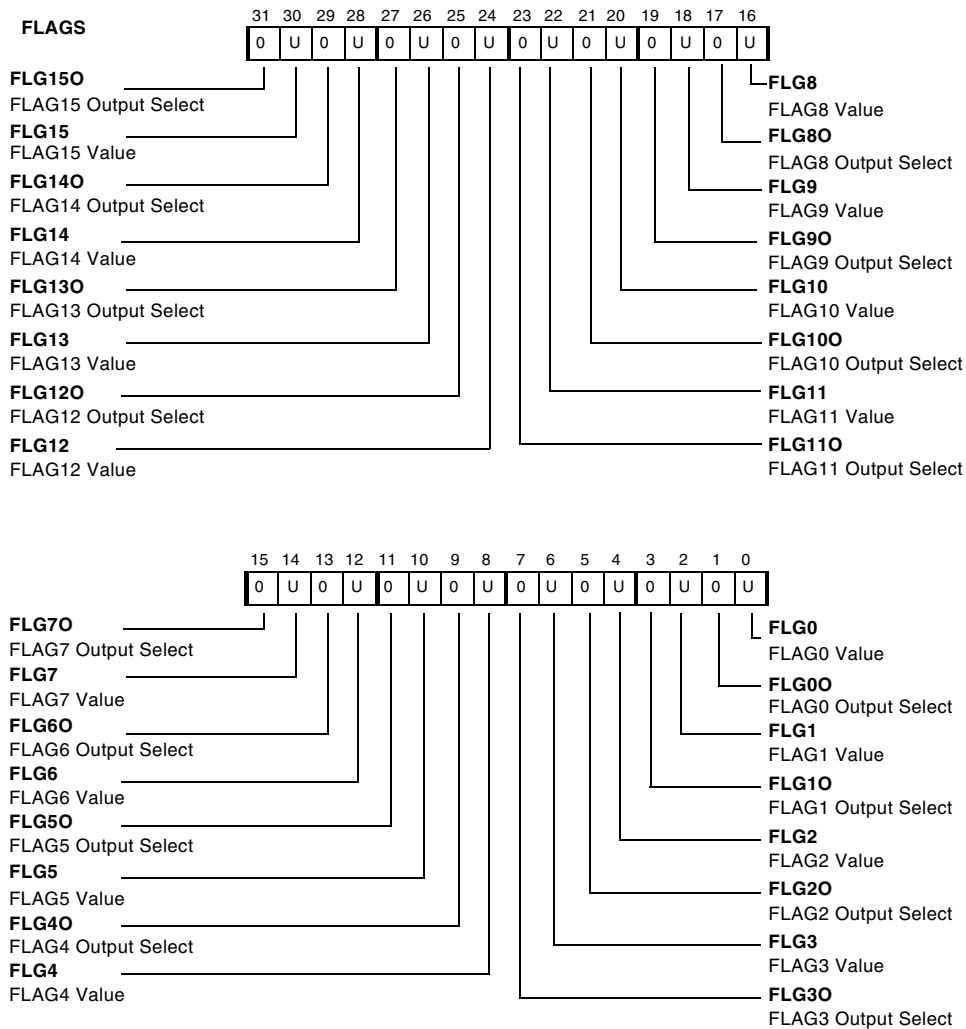
The `FLAGS` register consists of two bits for each flag, one bit to specify flag direction (or output select) where 0 = input, 1 = output and the other bit to specify the flag's value. If flags are in input mode, the flag value in `FLAGS` register is read-only and it shows the input flag status. If flags are in output mode, then the flag value can be written to change output flags and it can also be read to determine the last value written.

`FLAGS` can be mapped into any of the following three pin groups:

1. Four `FLAG/IRQ` pins

- Only `FLAG0-3` can be mapped.
- Should be programmed through `SYSCTL` register according to `EPDATA` mode, `IRQxEN`, `TMREXPEN`, and `MSEN`.
- `IRQEN` and `TMREXPEN` take priority over all other modes of these pins.

Processor Pin Descriptions



-For all FLGx bits, FLAGx values are as follows: 0=LOW, 1=HIGH.
 -For all FLGxO bits, FLAGx output selects are as follows: 0=FLAGx Input, 1=FLAGx Output.
 -U indicates the bit value is unknown at reset.

Figure 14-3. FLAGS Register

2. 32 data pins

- All 16 flags can be mapped.
- Can be mapped only in groups of four.
- Should be programmed through the SYSCCTL register. [For more information, see “System Control Register \(SYSCCTL\)” on page A-5.](#)

3. 14 DPI pins

- 12 flags (FLAG4–15) can be mapped.
- Any flag can go to any DPI pins.
- Should be programmed through the SRU2 registers. [For more information, see “DPI/SRU2 Connection Groups” on page 4-51.](#)

The flag data paths from the processor core to the data pin multiplexer block and SRU2 are parallel ([Figure 14-1 on page 14-4](#)). Therefore, for FLAGS0–3:

- In output mode, if the same flag is mapped to both data pins and flag pins, then the output comes from both pins.
- In input mode, if the same flag is mapped to both data pins and flag pins, then the input from data pins is given priority.

Processor Pin Descriptions

And for `FLAGS4-15`:

- In output mode, if the same flag is mapped to both data pins and DPI pins, then the output comes from both pins.
- In input mode, if the same flag is mapped to both data pins and DPI pins, the input from data pins is given priority.
- In input mode, if the same flag is mapped to both upper and lower data pins, the input from the lower data pins is given priority.

RESETOUT/CLKOUT/RUNRSTIN

The default behavior of the `RESETOUT` pin is to provide a 4096 cycle delay that allows the PLL to lock. In PLL bypass mode, where the `CLK_CFG` pins = 11, this pin functions as a `CLKOUT` signal to clock synchronous peripherals and memory. This can also be accomplished by setting (= 1) bit 12 (`CLKOUTEN`) in the power management control register and this functionality applies to all ADSP-21367/8/9 and ADSP-2137x processors. Finally, on the ADSP-2137x processors only, the `RESETOUT` pin can be configured to provide a running reset. [For more information, see “Running Reset \(ADSP-2137x\)” on page 14-22.](#)

JTAG Interface Pins

The JTAG test access port (TAP) consists of the `TCK`, `TMS`, `TDI`, `TD0`, and `TRST` pins. The JTAG port can be connected to a controller that performs a boundary scan for testing purposes. This port is also used by the Analog Devices DSP Tools product line of JTAG emulators and development software to access on-chip emulation features. To allow the use of the emulator, a connector for its in-circuit probe must be included in the target system.

If the $\overline{\text{TRST}}$ pin is not asserted (or held low) at power-up, the JTAG port is in an undefined state that may cause the processor to drive out on I/O pins that would normally be three-stated at reset. The $\overline{\text{TRST}}$ pin can be held low with a jumper to ground on the target board connector.

A detailed discussion of JTAG and its uses can be found in Engineer-to-Engineer Note EE-68, *Analog Devices JTAG Emulation Technical Reference*. This document is available on the Analog Devices Web site at www.analog.com/processors.

Clock Derivation

The processor uses a PLL (phased-locked loop) to provide clocks that switch at higher frequencies than the system clock (CLKIN). The PLL-based clocking methodology used on the ADSP-21367/8/9 and ADSP-2137x processors influences the clock frequencies and behavior for the serial, SPI, and external ports, in addition to the processor core and internal memory. In each case, the processor PLL provides a non-skewed clock to the core, internal memory, port logic and I/O pins.

The PLL provides a clock that switches at the processor core frequency to the serial ports. Each of the serial ports can be programmed to operate at clock frequencies derived from this clock. The six serial ports' transmit and receive clocks are divided down from the processor core clock frequency by setting the DIV_x registers appropriately.

On power-up, the CLKCFG1-0 pins are used to select ratios of 32:1, 16:1, and 6:1. After booting, numerous other ratios (slowing or speeding up the clock) can be selected through software control.

Power Management Control Register

The ADSP-21367/8/9 and ADSP-2137x processors have a power management control register (PMCTL) that allows programs to determine the amount of power dissipated. This includes the ability to program the PLL dynamically in software, achieving a slower core instruction rate that minimizes power use. For a complete register description, see [“Power Management Control Register \(PMCTL\)” on page A-170](#).

The PMCTL register also allows programs to disable the clock source to a particular processor peripheral, for example the serial ports or the timers, to further conserve power. By default, each peripheral block has its internal clock enabled only after it is initialized. Programs can use the PMCTL register to turn the specific peripheral off after the application no longer needs it. After reset, these peripheral clocks are not enabled until the peripheral itself is initialized by the program. [Listing 14-1](#) and [Listing 14-2](#) show some clock management options.

Listing 14-1. Using the System Clock for the SPI Module

```
ustat2 = dm(PMCTL);  
bit set ustat2 SPIPDN;    /* disable internal peripheral clock for  
                           SPI module. SPIPDN is defined as bit  
                           30 of PMCTL */  
  
dm(PMCTL) = ustat2;
```

Listing 14-2. PMCTL Example Code

```

ENABLING CLKOUT:

ustat2 = dm(PMCTL);
bit set ustat2 CLKOUTEN;    /* switch pin function from Reset Out
                             (RSTOUT) to CLKOUT */

dm(PMCTL) = ustat2;

PLL Divisor modification:
ustat2 = dm(PMCTL);
bit clr ustat2 PLLM63|PLLD8;    /* clear old multiplier and
                                divisor*/
bit set ustat2 DIVEN|PLLD8;    /* set and enable PLL Divisor for
                                CoreCLK = CLKIN/8 */

dm(PMCTL) = ustat2;

PLL Multiplier modification:
ustat2 = dm(PMCTL);
bit clr ustat2 PLLM63|PLLD8;    /* clear old multiplier and
                                divisor*/
bit set ustat2 PLLM8 | PLLBP; /* set a multiplier of 8
                                (default divisor is 2) and put
                                PLL in Bypass */

dm(PMCTL) = ustat2;
waiting loop:r0 = 4096;    /* wait for PLL to lock at new rate
                             (requirement for modifying
                             multiplier only) */

lcntr = r0, do pllwait until lce;
pllwait: nop;
ustat2 = dm(PMCTL);
bit clr ustat2 PLLBP;    /* take PLL out of Bypass, PLL is
                             now at CLKIN*4 (CoreCLK = CLKIN *
                             M/N = CLKIN* 16/4) */

dm(PMCTL) = ustat2;

```

Clock Derivation

When the PLL is programmed using a multiplier and a divisor, the `DIVEN` and `PLLBP` bits should NOT be programmed in the same core clock cycle. There should be a delay of at least one core clock cycle between programming these bits. The approaches described below and shown in [Listing 14-3](#), [Listing 14-4](#) and [Listing 14-5](#) can be used to accomplish this.

PLL Programming Examples

Use the following procedure to program the PLL. Please include the corresponding processor-specific header definition files `def21367.h`, `def21368.h`, `def21369.h`, `def21371.h`, `def21375.h` and `cdef21367.h`, `cdef21368.h`, `cdef21369.h`, `cdef21371.h`, and `cdef21375.h`, which contain the register and bit definitions.

1. Set the PLL multiplier and divisor value and enable the divisor by setting the `DIVEN` bit.
2. After one core clock cycle, place the PLL in bypass mode by setting (= 1) the `PLLBP` bit.
3. Wait in bypass mode until the PLL locks.
4. Take the PLL out of bypass mode by clearing (= 0) the bypass bit.

Listing 14-3. PLL Programming Example 1

```
ustat2 = dm(PMCTL);

bit clr ustat2 PLLM63|PLLD8;          /* clear old multiplier and
                                       divisor*/

bit set ustat2 DIVEN | PLLD4 | PLLM16; /* set a multiplier of
                                       16 and a divider of 4 */

dm(PMCTL) = ustat2;
bit set ustat2 PLLBP;                /* Put PLL in bypass mode. */
```



```

dm(PMCTL) = ustat2;
waiting_loop:
r0 = 4096;                                /* wait for PLL to lock at new rate
                                           (requirement for modifying
                                           multiplier only) */
lcntr = r0, do pllwait until lce;
pllwait: nop;
ustat2 = dm(PMCTL);
bit clr ustat2 PLLBP;    /* take PLL out of Bypass, PLL is now at
                           CLKIN*4 (CoreCLK = CLKIN * M/N =
                           CLKIN* 16/4) */
dm(PMCTL) = ustat2;

```

Use the following alternate procedure to program the PLL.

1. Set the PLL multiplier and divisor values and place the PLL in bypass mode by setting the PLLBP bit.
2. Wait in the bypass mode until the PLL locks.
3. Take the PLL out of bypass mode by clearing the bypass bit.
4. Wait for one core clock cycle.
5. Enable the divisor by setting the DIVEN bit.

Listing 14-4. PLL Programming Example 2

```
ustat2 = dm(PMCTL);

bit clr ustat2 PLLM63|PLLD8;          /* clear old multiplier and
                                       divisor*/

bit set ustat2 PLLBP | PLLD4 | PLLM16; /* set a multiplier of 16
                                       and a divider of 4 and
                                       enable Bypass mode*/

ustat2 = dm(PMCTL);
waiting_loop:
r0 = 4096;                          /* wait for PLL to lock at new rate
                                       (requirement for modifying multiplier only) */

lcntr = r0, do pllwait until lce;
pllwait: nop;ustat2 = dm(PMCTL);
bit clr ustat2 PLLBP;                /* take PLL out of Bypass*/
dm(PMCTL) = ustat2;

ustat2 = dm(PMCTL);
bit set ustat2 DIVEN;                /* Enable the DIVEN bit, PLL is now at
                                       CLKIN*4 (CoreCLK = CLKIN * M/N = CLKIN* 16/4) */
dm(PMCTL) = ustat2;
```

[Listing 14-5](#) is a PLL programming example in C.

Listing 14-5. PLL Programming Sequence (C)

```
pmctlsetting= *pPMCTL;
pmctlsetting &= ~(0xFF); /* Clear */

/* CLKIN= 24.576 MHz, Multiplier= 27, Divisor= 2,
   CCLK_SDCLK_RATIO 2. */
```

```

/* Core clock = (24.576 MHz * 27) / 2 = 331.776 MHz */
pmctlsetting= SDCKR2|PLLM27|PLLD2|DIVEN;
*pPMCTL= pmctlsetting;
pmctlsetting|= PLLBP;
*pPMCTL= pmctlsetting;
pmctlsetting ^= DIVEN;

/* Wait for around 4096 cycles for the pll to lock. */
for (i=0; i<4096; i++)
asm("nop;");
*pPMCTL ^= PLLBP;          /* Clear Bypass Mode */
*pPMCTL |= (CLKOUTEN);    /* and start clkout */

```

Phase-Locked Loop Startup

The $\overline{\text{RESET}}$ signal can be held low long enough to guarantee a stable CLKIN source and stable VDDINT/VDDEXT power supplies before the PLL is reset.

The PLL needs time to lock to the CLKIN frequency before the core can execute or begin the boot process. A delayed core reset (RESETOUT) has been added through the delay circuit to provide this time. There is a 12-bit counter that counts up to 4096 CLKIN cycles after $\overline{\text{RESET}}$ is transitioned from low to high. This is normally 1.3 μs for the minimum CLKIN frequency. The delay circuit is activated at the same time the PLL is taken out of reset.

The advantage of the delayed core reset is that the PLL can be reset any number of times without having to power down the system. If there is a brownout situation, the watchdog circuit only has to control the $\overline{\text{RESET}}$.



For more information on device power up, see the processor specific data sheet.

RESET and CLKIN

The processor receives its clock input on the `CLKIN` pin. The processor uses an on-chip, phase-locked loop (PLL) to generate its internal clock, which is a multiple of the `CLKIN` frequency. Because the PLL requires some time to achieve phase lock, `CLKIN` must be valid for a minimum time period during reset before the `RESET` signal can be deasserted. For information on minimum clock setup, see the appropriate ADSP-2136x or ADSP-2137x SHARC processor data sheet.

Table 14-2 describes the internal clock to `CLKIN` frequency ratios supported by the processor. Note that programs control the PLL through the `PMCTL` register. For more information, see “Power Management Control Register (PMCTL)” on page A-170.

Table 14-2. Pin Selectable Clock Rate Ratios

CLKCFG1-0	Core to CLKIN Ratio
00	6:1
01	32:1
10	16:1



When using an external crystal, the maximum crystal frequency cannot exceed 25 MHz. The internal clock generator, when used in conjunction with the `XTAL` pin and an external crystal, is designed to support up to a maximum of 25 MHz external crystal frequency. For all other external clock sources, the maximum `CLKIN` frequency is 50 MHz.

Table 14-3 demonstrates the internal core clock switching frequency across a range of `CLKIN` frequencies. The minimum operational range for any given frequency may be constrained by the operating range of the phase-lock loop. Note that the goal in selecting a particular clock ratio for

the application is to provide the highest permissible internal frequency for a given CLKIN frequency. For more information on available clock rates, see the processor-specific data sheet.

Table 14-3. Selecting Core to CLKIN Ratio

	Typical Crystal and Clock Oscillators Inputs					
	12.500	16.667	25.000	33.333	40.000	50.000
Clock Ratios	Core CLK (MHz)					
3:1	N/A	N/A	N/A	100.000	120.000	150.000
8:1	100	133.33	200.00	266.67	320.00	400.00
16:1	200	266.67	400	N/A	N/A	N/A



Shared memory processor designs that use several ADSP-21368 processors must have the same CLKIN source. These devices should also be brought out of reset together.

If an external master clock is used, it should not drive the CLKIN pin when the processor is not powered. The clock must be driven immediately after power-up; otherwise, internal gates stay in an undefined (hot) state and can draw excess current. After power-up, there should be sufficient time for the oscillator to start up, reach full amplitude, and deliver a stable CLKIN signal to the processor before the reset is released. This may take 100 μ s depending on the choice of crystal, operating frequency, loop gain and capacitor ratios. For details on timing, refer to the processor-specific data sheet.

After the external processor $\overline{\text{RESET}}$ signal is deasserted, the PLL starts operating. The rest of the chip is held in reset for 4096 CLKIN cycles after $\overline{\text{RESET}}$ is deasserted by an internal reset signal. This sequence allows the PLL to lock and stabilize. Add one CLKIN cycle if $\overline{\text{RESET}}$ does not meet setup requirements with respect to the CLKIN falling edge.

Running Reset (ADSP-2137x)

All members of the SHARC processor family, including the ADSP-21375 and ADSP-21371, continue to support the hardware reset controlled with the `RESET` pin. The de-assertion of this hardware reset enables the PLL and asserting it resets the PLL. In the time it takes the PLL to acquire lock (set to 4096 `CLKIN` cycles), the processor, internal memory, and the peripherals are held in reset. Upon completion of the 4096 `CLKIN` cycles, the chip is brought out of reset. This is indicated on the `RESETOUT/CLKOUT` pin for the three valid boot modes (00, 01, 10 settings of `BOOT_CFG1-0` pins). [For more information, see “Booting” on page 14-37.](#)

In addition to the hardware reset, there is also support for a software reset, which can be asserted by setting bit 0 of the `SYSCTL` register.

In the ADSP-21375 and ADSP-21371 processors, an additional feature, called *running reset*, has been added. Running reset resets everything on the chip, including the core and peripheral registers and the program counter (PC). Running reset also clears all stacks and counters. However, it does NOT reset the PLL (like the software reset) and the SDRAM controller (to maintain SDRAM auto-refresh). De-assertion of this reset does not result in a boot (unlike any of the other resets) even if a valid boot mode is configured on the `BOOT_CFG1-0` pins. Instead, the program counter value is reset to the first location of internal memory (0x90005) and the sequencer begins executing instructions starting from this address.

This feature has been added to allow the ADSP-2137x processors to execute self-modifying code that has previously overwritten existing code in internal memory or as an external watchdog that activates in case there is a malfunction or exception within a peripheral occurs, and a context reset of the processor is sufficient to restore the state, (whereas a complete boot is not required).

System Design Considerations

It is important that an external 10 k Ω pull-up resistor is placed on the RESETOUT/CLKOUT/RUNRSTOUT pin if it is intended to be used as an input for initiating a running reset on the ADSP-2137x processor as shown in Figure 14-4. Also, it is extremely important to ensure that an external device, such as a micro controller, does not drive this signal during or after coming out of a power-on or hard-reset.

Figure 14-4 shows the active state of the pin during and after $\overline{\text{RESET}}$. The ADSP-2137x processor is actively driving this pin as an output.

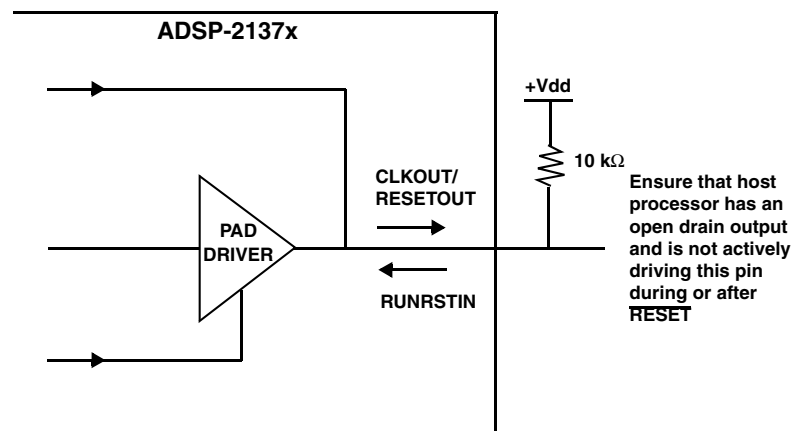


Figure 14-4. RESETOUT/CLKOUT Pin Muxed with RUNRSTIN

If the system uses an external host or micro controller to control running reset, ensure that the external device waits until the ADSP-2137x processor driver has been internally disabled (by writing to the RUNRSTCTL register) before actively driving this signal at $\overline{\text{RESET}}$. Connect the CLKOUT pin to an open-drain pin on the host side, or use an external three-state buffer.

Clock Derivation

There are several possible methods that can be used to implement running reset. The following illustrates one example of a running reset implementation involving an ADSP-2137x processor and a host processor.

External Host

In an AVR (audio-video receiver) system, a host micro controller may communicate with the ADSP-2137x processor using the serial peripheral interface (SPI). Or, if no SPI pins are available on the host device, it can use spare flag I/Os to connect with the SPI of SHARC as shown in [Figure 14-5](#)). In this case, the host implements the SPI protocol on the port pins.

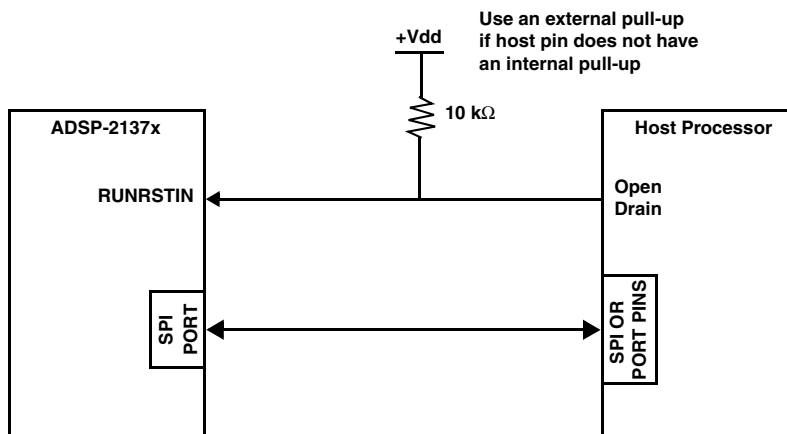


Figure 14-5. Example System Interface With an External Host

Using the SPI protocol with additional control words and commands, running reset can become an addition command from the host or from the ADSP-2137x processor as described in the following procedure.

1. The host initiates a running reset by informing the processor over the command interface.

2. The ADSP-2137x processor receives the command and completes any unfinished work which may also include writing to the `RUNRSTCTL` register.
3. When the ADSP-2137x processor is ready to accept the running reset, it signals the host over the command interface.
4. The host drives the running reset input into the ADSP-21375/71 processor.

Running Reset Control Register (RUNRSTCTL)

To program the running reset feature, (by toggling the `RESETOUT` pin), a new register, running reset control (`RUNRSTCTL`) has been added to the ADSP-2137x processor architecture.

This register is located at memory-mapped address 0x2100. On previous generations of SHARCs, the `RESETOUT/CLKOUT` pin was an output. On the ADSP-2137x processors, this pin is an input/output, and the sense and direction of the pin is controlled by bit 0 of the `RUNRSTCTL` register (see [Table 14-4](#)).

Table 14-4. Running Reset Control Register Bit Descriptions

Bit	Name	Description
0	PM_RUNRST_PINEN	Configures the <code>RESETOUT</code> pin for <code>RUNRST</code> input. Read Write 0 = <code>RESETOUT</code> pin is <code>CLKOUT/RESETOUT</code> 1 = <code>RESETOUT</code> pin is <code>RUNRST</code> input Reset value = 0
1	PM_RUNRST_EN	Enable the running reset functionality. 0 = Running reset disabled 1 = Running reset enabled Reset value = 0
31–2	Reserved	Reads return 0 Reset value = 0

Programming The RUNRSTCTL Register

To configure running reset:

1. Set bit 0 (=1) to change RESETOUT/CLKOUT pin direction to input.
2. Ensure that the RESETOUT/CLKOUT pin is driven to a proper state, and then assert RUNRSTEN to sensitize logic to the state of the RESETOUT/CLKOUT pin.

If bit 1 of the RUNRSTCTL register is not set, attempting to cause a running reset by toggling the RESETOUT/CLKOUT pin does not result in a reset.



The RUNRSTCTL register is reset only on assertion of a hardware reset, software reset, emulator reset, or by writing to the appropriate bits of the RUNRSTCTL register via software.

The system reacts to the assertion and recognition of a running reset in the following way.

- The core-PLL is NOT reset, and continues to run
- Internal memory SRAM contents remain unaltered
- The processor core and peripherals are reset exactly as if a Power-on (hardware) reset is asserted, except:
 - The SDRAM controller continues to run and refresh as programmed.
 - The contents of external SDRAM are unaffected, and retain their values prior to a running reset.
 - A system boot is NOT initiated. Instead, the program counter is cleared and program execution begins from the very first location of program memory (from the reset interrupt vector table).

Reset Generators

It is important that a processor (or programmable device) have a reliable active $\overline{\text{RESET}}$ that is released once the power supplies and internal clock circuits have stabilized. The $\overline{\text{RESET}}$ signal should not only offer a suitable delay, but it should also have a clean monotonic edge. Analog Devices has a range of microprocessor supervisory ICs with different features. Features include one or more of the following:

- Power-up reset
- Optional manual reset input
- Power low monitor
- Backup battery switching

The part number series for reset and supervisory circuits from Analog Devices are as follows:

- ADM69x
- ADM70x
- ADM80x
- ADM1232
- ADM181x
- ADM869x

A simple power-up reset circuit is shown in [Figure 14-6](#) using the ADM809-RART reset generator. The ADM809 provides an active low $\overline{\text{RESET}}$ signal whenever the supply voltage is below 2.63 V. At power-up, a 240 μs active reset delay is generated to give the power supplies and oscillators time to stabilize.

Clock Derivation

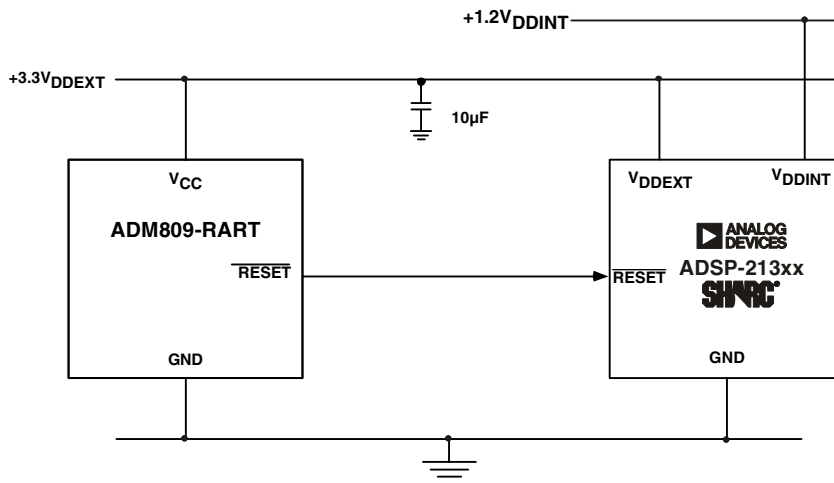


Figure 14-6. Simple Reset Generator

Another part, the ADM706TAR, provides power on $\overline{\text{RESET}}$ and optional manual $\overline{\text{RESET}}$. It allows designers to create a more complete supervisory circuit that monitors the supply voltage. Monitoring the supply voltage allows the system to initiate an orderly shutdown in the event of power failure. The ADM706TAR also allows designers to create a watchdog timer that monitors for software failure. This part is available in an 8-lead SOIC package. [Figure 14-7](#) shows a typical application circuit using the ADM706TAR.

Timing Specifications

The ADSP-21367/8/9 and ADSP-2137x processor's internal clock (a multiple of CLKIN) provides the clock signal for timing internal memory, processor core, serial ports, and SPI (as required for read/write strobes). During reset, program the ratio between the processor's internal clock frequency and external (CLKIN) clock frequency with the CLK_CFG1-0 pins.

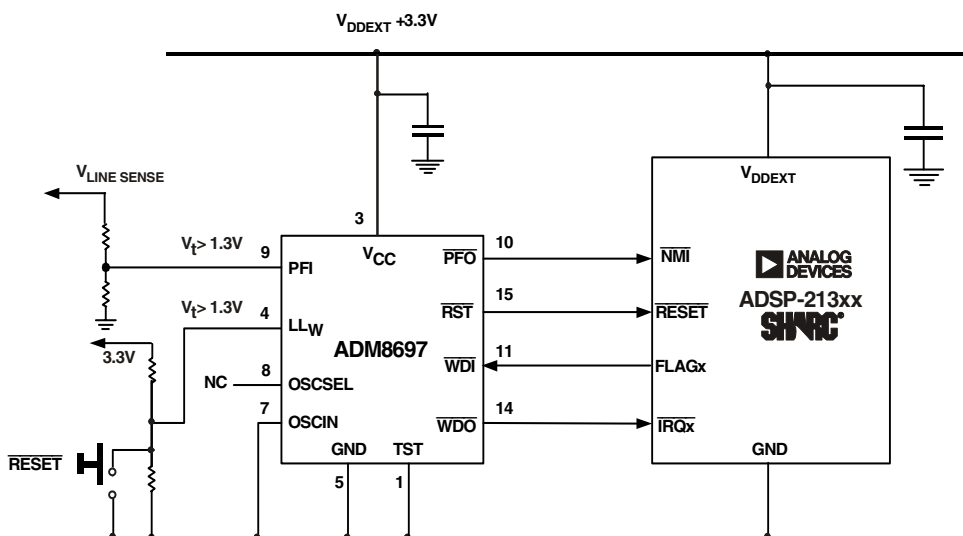


Figure 14-7. Reset Generator and Power Supply Monitor

To determine switching frequencies for the serial ports, divide down the internal clock, using the programmable divider control of each port (DIV_x for the serial ports). For the SPI port, the $BAUDR$ bit in the $SPICTL$ register controls the $SPICLK$ baud rate based on the core clock frequency.

Note the following definition and [Figure 14-8](#) for various clock periods that are functions of $CLKIN$ and the appropriate ratio control:

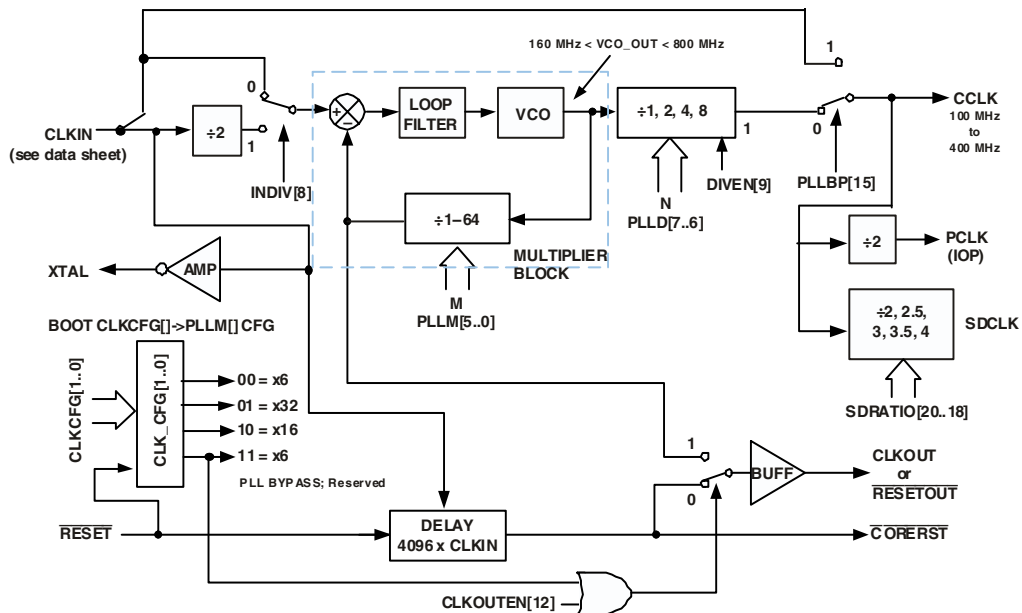
$CCLK = \text{core clock} = PLLICK \times \text{PLL multiply ratio}$ (determined by CLK_CFG pins)

Programs can modify this setting using bits in the $PMCTL$ register. For more information, see [“Power Management Control Register \(PMCTL\)”](#) on [page A-170](#).



Performing a hard reset on the processor also resets the PLL.

Clock Derivation



Notes

1. CLKOUT is muxed with RESETOUT. After reset, RESETOUT is selected. CLKOUT is selected by setting bit 12 in the PMCTL register.
2. The PLL ratio is controlled by the states of the CLKCFG[1:0] pins at reset and can be modified in software through the PLLM and PLLDx bits in the PMCTL register.
3. To place the PLL in bypass mode, set bit 15 in the PMCTL register. (CCLK = PLLCLK when set.)
4. Programs can interrupt the internal clock source to each of the following peripherals: timer, SPI, SPORTs, and parallel port. These internal clock sources are disabled at reset and are enabled and left enabled after each peripheral is enabled. Note that these peripherals DO NOT RUN at the core clock frequency. For more information please see the respective peripheral chapters in the *ADSP-2136x SHARC Processor Hardware Reference*.
5. Please refer to the processor specific data sheets for maximum CLKIN and crystal source specifications.

Figure 14-8. Core Clock and System Clock Relationship to CLKIN

Table 14-5. CLKOUT and CCLK Clock Generation Operation

Timing Requirements		Calculation		Description
CLKIN	=	$1/t_{CKIN}$	=	Input Clock
CLKOUT	=	$1/t_{TCK}$	=	Local Clock Out
PLLCLK	=	$1/t_{PLLIN}$	=	PLL Input Clock
CCLK	=	$1/t_{CCLK}$	=	Core Clock

Table 14-6. Clock Relationships

Timing Requirements		Description ¹
t_{CK}	=	CLKOUT Clock Period
t_{PLICK}	=	PLL Input Clock
t_{CCLK}	=	Core Clock Period (Processor)
t_{PCLK}	=	Peripheral Clock Period = $2 \times t_{CCLK}$
t_{SCLK}	=	Serial Port Clock Period = $(t_{PCLK}) \times SR$
t_{SPICLK}	=	SPI Clock Period = $(t_{CCLK}) \times SPIR$

¹ where:

SR = serial port-to-core clock ratio (wide range, determined by CLKDIV)

SPIR = SPI-to-core clock ratio (wide range, determined by SPICL register)

SCLK = serial port clock

SPICLK = SPI clock

Table 14-7 describes clock ratio requirements. Table 14-8 shows an example clock derivation.

Table 14-7. Clock Ratios

Timing Requirements		Description
c_{RTO}	=	Core to CLKOUT ratio (3:1, 8:1, or 16:1, determined by CLK_CFGx pins at reset). Programs can modify this ratio using the PMCTL register.
s_{RTO}	=	Sport:core clock ratio (wide range determined by xCLKDIV)

Table 14-8. Clock Derivation

Timing Requirements		Description
t_{CCLK}	=	$(t_{CK}) \times c_{RTO}$
t_{SCLK}	=	$(t_{CCLK}) \times s_{RTO}$

Input Synchronization Delay

The processor has several asynchronous inputs— $\overline{\text{RESET}}$, TRST , $\overline{\text{IRQ2-0}}$, MS3-0 , and FLAG16-0 (when configured as inputs). These inputs can be asserted in arbitrary phase to the processor clock, CLKIN . The processor synchronizes the inputs prior to recognizing them. The delay associated with recognition is called the synchronization delay.

Any asynchronous input must be valid prior to the recognition point in a particular cycle. If an input does not meet the setup time on a given cycle, it may be recognized in the current cycle or during the next cycle.

To ensure recognition of an asynchronous input, it must be asserted for at least one full processor cycle plus setup and hold time, except for $\overline{\text{RESET}}$, which must be asserted for at least four processor cycles. The minimum time prior to recognition (the setup and hold time) is specified in the processor's data sheet.

Conditioning Input Signals

The processor is a CMOS device. It has input conditioning circuits which simplify system design by filtering or latching input signals to reduce susceptibility to glitches or reflections.

The following sections describe why these circuits are needed and their effect on input signals.

A typical CMOS input consists of an inverter with specific N and P device sizes that cause a switching point of approximately 1.4 V. This level is selected to be the midpoint of the standard TTL interface specification of $V_{\text{IL}} = 0.8 \text{ V}$ and $V_{\text{IH}} = 2.0 \text{ V}$. This input inverter, unfortunately, has a fast response to input signals and external glitches wider than 1 ns. Filter circuits and hysteresis are added after the input inverter on some processor inputs, as described in the following sections.

RESET Input Hysteresis

Hysteresis is used only on the $\overline{\text{RESET}}$ input signal. Hysteresis causes the switching point of the input inverter to be slightly above 1.4 V for a rising edge and slightly below 1.4 V for a falling edge. The value of the hysteresis is approximately ± 0.1 V. The hysteresis is intended to prevent multiple triggering of signals which are allowed to rise slowly, as might be expected on a reset line with a delay implemented by an RC input circuit. Hysteresis is not used to reduce the effect of ringing on processor input signals with fast edges, because the amount of hysteresis that can be used on a CMOS chip is too small to make a difference. The small amount of hysteresis allowable is due to the restrictions on the tolerance of the V_{IL} and V_{IH} TTL input levels under worst-case conditions. Refer to the product-specific processor data sheet for exact specifications.

Designing for High Frequency Operation

Because the processor must be able to operate at very high clock frequencies, signal integrity and noise problems must be considered for circuit board design and layout. The following sections discuss these topics and suggest various techniques to use when designing and debugging target systems.

All synchronous processor behavior is specified to CLKIN . System designers are encouraged to clock synchronous peripherals with this same clock source (or a different low-skew output from the same clock driver).

Clock Specifications and Jitter

The clock signal must be free of ringing and jitter. Clock jitter can easily be introduced into a system where more than one clock frequency exists. Jitter should be kept to an absolute minimum. High frequency jitter on the clock to the processor may result in abbreviated internal cycles.

Designing for High Frequency Operation

Keep the portions of the system that operate at different frequencies as physically separate as possible. The clock supplied to the processor must have a rise time of 3 ns or less and must meet or exceed a high and low voltage of 2 V and 0.4 V, respectively.




Never share a clock buffer IC with a signal of a different clock frequency as this introduces excessive jitter.

Other Recommendations and Suggestions

- Use more than one ground plane on the PCB to reduce crosstalk. Be sure to use lots of vias between the ground planes. One V_{DD} plane for each supply is sufficient. These planes should be in the center of the PCB.
- To reduce crosstalk, keep critical signals such as clocks, strobes, and bus requests on a signal layer next to a ground plane and away from these signals, or layout critical signals perpendicular to other non-critical signals.
- If possible, position the processors on both sides of the board to reduce area and distances.
- To allow better control of impedance and delay, and to reduce crosstalk, design for lower transmission line impedances.
- Use 3.3 V peripheral components and power supplies to help reduce transmission line problems, ground bounce and noise coupling (the receiver switching voltage of 1.5 V is close to the middle of the voltage swing).
- Experiment with the board and isolate crosstalk and noise issues from reflection issues. This can be done by driving a signal wire from a pulse generator and studying the reflections while other components and signals are passive.

Decoupling Capacitors and Ground Planes

Extended copper planes must be used for the ground and power supplies. Designs should use an absolute minimum of 12 bypass capacitors (four 0.1 μF , four 10 nF and four 1 nF ceramic) for each VDD_{EXT} and VDD_{INT} supply. More extensive bypassing may be required for some applications. The capacitors should be placed close to the package as shown in [Figure 14-9](#). The decoupling capacitors should be tied directly to the power and ground planes with vias that touch their solder pads. Surface-mount capacitors are recommended because of their lower series inductances (ESL) and higher series resonant frequencies. Connect the power and ground planes to the ADSP-21367/8/9 and ADSP-2137x processor's power supply pins directly with vias—do not use traces. The ground planes should not be densely perforated with vias or traces as this reduces their effectiveness. In addition, there should be several large tantalum capacitors on the board.

 Designs can use either bypass placement case shown in [Figure 14-9](#), or combinations of the two. Designs should try to minimize signal feedthroughs that perforate the ground plane.

Oscilloscope Probes

When making high speed measurements, be sure to use a “bayonet”-type ground clip (or similarly short, < 0.5 inch), attached to the tip of the oscilloscope probe. The probe should be a low capacitance active probe with 3 pF or less of loading. The use of a standard ground clip with 4 inches of ground lead causes ringing to be seen on the displayed trace and makes the signal appear to have excessive overshoot and undershoot. A 1 GHz or better sampling oscilloscope is needed to see the signals accurately.

Designing for High Frequency Operation

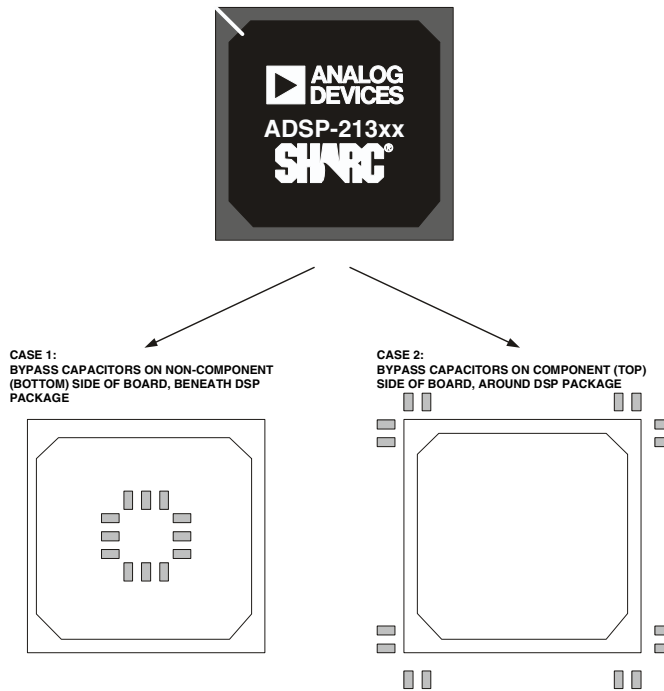


Figure 14-9. Bypass Capacitor Placement

Recommended Reading

The text *High-Speed Digital Design: A Handbook of Black Magic* is recommended for further reading. This book is a technical reference that covers the problems encountered in state-of-the-art, high frequency digital circuit design. It is also an excellent source of information and practical ideas. Topics covered in the book include:

- High-Speed Properties of Logic Gates
- Measurement Techniques
- Transmission Lines

- Ground Planes and Layer Stacking
- Terminations
- Vias
- Power Systems
- Connectors
- Ribbon Cables
- Clock Distribution
- Clock Oscillators

High-Speed Digital Design: A Handbook of Black Magic, Johnson & Graham, Prentice Hall, Inc., ISBN 0-13-395724-1.

Booting

When a processor is initially powered up, its internal SRAM is undefined. Before actual program execution can begin, the application must be loaded from an external non-volatile source such as flash memory or a host processor. This process is known as *bootstrap loading* or *booting* and is automatically performed by the processor after power-up or after a software reset.

The ADSP-21367/8/9 and ADSP-2137x processors support three booting modes—EPROM, SPI master and SPI slave. Each of these modes uses the following general procedure:

1. At reset, the processor is hardwired to load 256 48-bit instruction words through a DMA starting at location 0x90000. In this section, these instructions are referred to as the *boot kernel* or *loader kernel*.

Booting

2. The DMA completes and the interrupt associated with the peripheral that the processor is booting from is activated. The processor jumps to the applicable interrupt vector location and executes the code located there. (Typically, the first instruction at the interrupt vector is a return from interrupt (RTI) instruction.)
3. The loader kernel executes a series of direct memory accesses (DMAs) to import the rest of the application, overwriting itself with the applications' interrupt vector table (IVT).
4. After executing the kernel, the processor returns to location 0x90005 where normal program execution begins.

To support this process, a 256-word loader kernel and loader (which converts executables into boot-loader images) are supplied with the VisualDSP++ development tools for both SPI and external port booting. For more information on the loader, see the tools documentation in [Related Documents on page xxxviii](#).

The boot source is determined by strapping the two `BOOT_CFG1-0` pins to either logic low or logic high. These settings are shown in [Table 14-9](#).

Table 14-9. Booting Modes

BOOT_CFG1-0	Description
00	SPI Slave Boot
01	SPI Master Boot
10	EPROM/FLASH Boot Through The External Port
11	Bypass Mode, Reserved

External Port Booting

The ADSP-21367/8/9 processors allow booting through the external port. There are two options, which are described in the following sections.

Bootng Through the AMI

The asynchronous memory interface (AMI) supports an 8-bit user boot called AMI boot. Only the \overline{MSI} signal is used for AMI(FLASH/EEPROM) booting. [Table 14-10](#) shows the bit settings for AMI boot. These bits are described in detail in [“AMI Control Registers \(AMICTLx\)” on page A-17](#).

Table 14-10. AMI Boot Bit Settings (AMICTLx)

Bit	Setting
AMI Enable (AMIEN)	1
Bus Width (BW)	00
Packing Disabled (PKDIS)	0
Most Significant Word First (MSWF)	0
ACK Pin Enable (ACKEN)	0
Wait States (WS)	10111
Bus Hold Cycle at the End of Write Access (HC)	000
Idle Cycle (IC)	000
Buffer Flush (FLSH)	0
Read Hold Cycle at the End of Read Access (RHC)	000
Disable Predictive Reads (NO_OPT)	0

Shared Memory Booting

To boot multiple processors from a single EPROM/FLASH, the processor performs the following steps.

1. Arbitrate for the bus.
2. Receive through DMA the 256-word boot kernel, after becoming bus master.
3. Release the bus, allowing the next processor access to the EPROM/FLASH.
4. Execute the loaded instructions. These usually consist of the boot kernel, which brings in the remaining application code.

For more information on developing executables for shared-memory booting, see the *VisualDSP++ Loader Manual* or the *VisualDSP++ Loader and Linker Manual* (depending on the VisualDSP++ release you are using).

The $\overline{MS1}$ signals from each processor may be wire ORed together to drive the chip select pin of the EPROM. Each processor boots in turn, according to its priority. When the last processor has finished booting, it must inform the others (which may be in the idle state) that program execution can begin (if all the processors are to begin executing instructions simultaneously). An example system that uses an alternating technique appears in [Figure 14-10](#). When multiple processors boot from one EPROM, they can boot either identical code or different code from the EPROM.

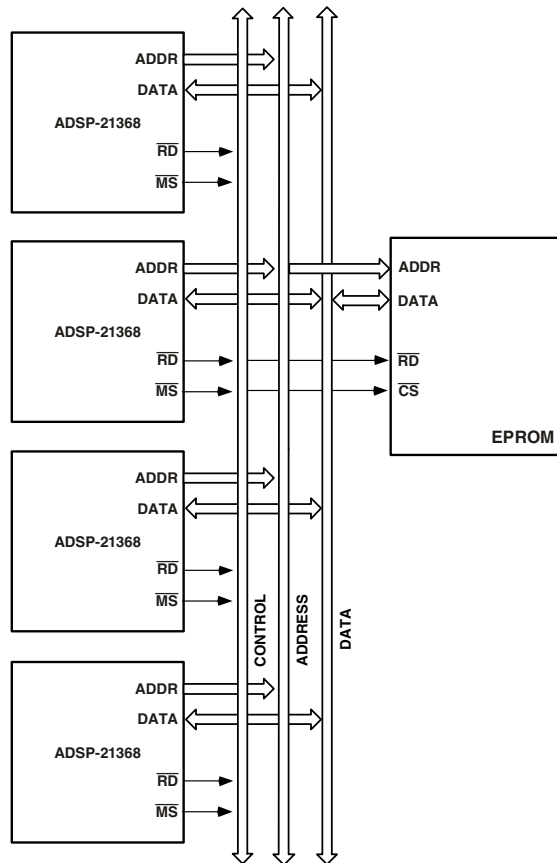


Figure 14-10. Alternating Booting From an EPROM

SPI Port Booting

The ADSP-21367/8/9 and ADSP-2137x processors support booting from a host processor through the SPI slave (`BOOT_CFG1-0 = 00`), and booting from an SPI flash, SPI PROM, or a host processor through SPI master mode (`BOOT_CFG1-0 = 01`).

i The SPI can also be configured to boot using the pins in the digital peripheral interface. [For more information, see “DPI/SRU2 Connection Groups” on page 4-51.](#)

Both SPI boot modes support booting from 8-, 16-, or 32-bit SPI devices. In all SPI boot modes, the data word size in the shift register is hardwired to 32 bits. Therefore, for 8- or 16-bit devices, data words are packed into the shift register to generate 32-bit words least significant bit (LSB) first, which are then shifted into internal memory. The relationship between the 32-bit words received into the `RXSPI` register and the instructions that need to be placed in internal memory is shown in [Figure 14-11](#).

For more information about 32- and 48-bit internal memory addressing, see the “Memory” chapter in the *ADSP-2136x SHARC Processor Programming Reference*.

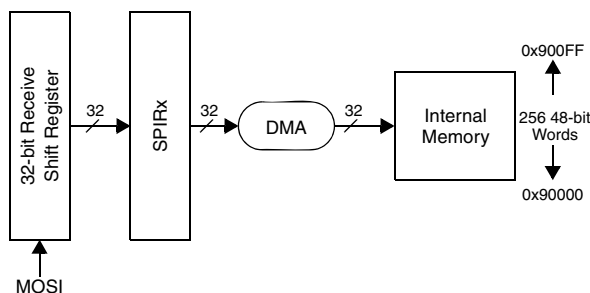


Figure 14-11. SPI Data Packing

For 16-bit SPI devices, two words shift into the 32-bit receive shift register (RXSR) before a DMA transfer to internal memory occurs. For 8-bit SPI devices, four words shift into the 32-bit receive shift register before a DMA transfer to internal memory occurs.

When booting, the ADSP-21367/8/9 and ADSP-2137x processors expect to receive words into the RXSPI register seamlessly. This means that bits are received continuously without breaks. [For more information, see “SPI Operation Using the Core” on page 6-13.](#) For different SPI host sizes, the processor expects to receive instructions and data packed in a least significant word (LSW) format.

[Figure 14-12](#) shows how a pair of instructions are packed for SPI booting using a 32-, 16-, and an 8-bit device. These two instructions are received as three 32-bit words as illustrated in [Figure 14-11](#).

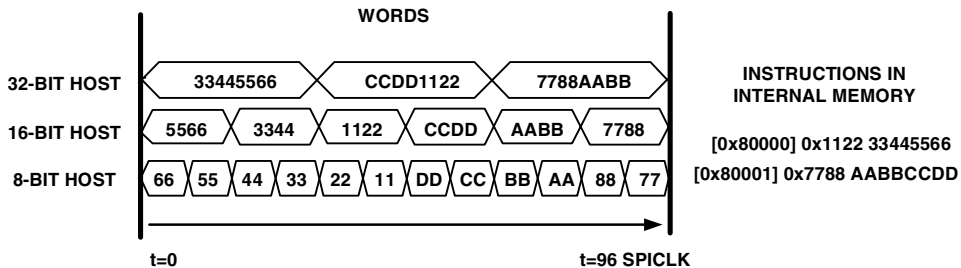


Figure 14-12. Instruction Packing for Different Hosts

The following sections examine how data is packed into internal memory during SPI booting for SPI devices with widths of 32, 16, or 8 bits.

32-Bit SPI Host Boot

[Figure 14-13](#) shows 32-bit SPI host packing of 48-bit instructions executed at PM addresses 0x90000 and 0x90001. The 32-bit word is shifted to internal program memory during the 256-word kernel load.

Booting

The following example shows a 48-bit instructions executed.

[0x90000] 0x112233445566

[0x90001] 0x7788AABBCCDD

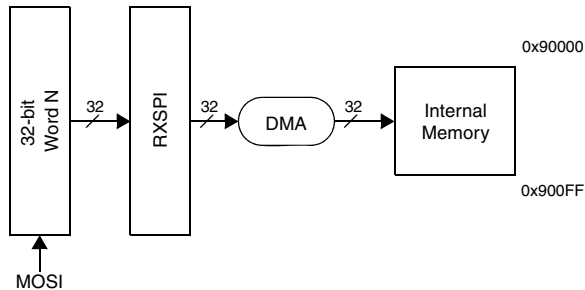


Figure 14-13. 32-Bit SPI Host Packing

The 32-bit SPI host packs or prearranges the data as:

SPI word 1 = 0x33445566

SPI word 2 = 0xCCDD1122

SPI word 3 = 0x7788AABB

16-Bit SPI Host Boot

Figure 14-14 shows how a 16-bit SPI host packs 48-bit instructions at PM addresses 0x90000 and 0x90001. For 16-bit hosts, two 16-bit words are packed into the shift register to generate a 32-bit word. The 32-bit word shifts to internal program memory during the kernel load.

The following example shows a 48-bit instructions executed.

[0x90000] 0x112233445566

[0x90001] 0x7788AABBCCDD

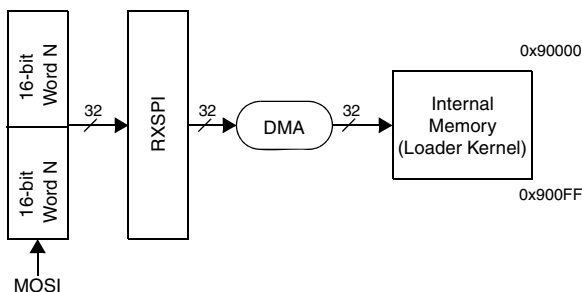


Figure 14-14. 16-Bit SPI Host Packing

The 16-bit SPI host packs or prearranges the data as:

SPI word 1 =	0x5566
SPI word 2 =	0x3344
SPI word 3 =	0x1122
SPI word 4 =	0xCCDD
SPI word 5 =	0xAABB
SPI word 6 =	0x7788

The initial boot of the 256-word loader kernel requires a 16-bit host to transmit 768 16-bit words. Two packed 16-bit words comprise the 32-bit word. The SPI DMA count value of 0x180 is equivalent to 384 words. Therefore, the total number of 16-bit words loaded is 768.

8-Bit SPI Host Boot

Figure 14-15 shows 8-bit SPI host packing of 48-bit instructions executed at PM addresses 0x90000 and 0x90001. For 8-bit hosts, four 8-bit words pack into the shift register to generate a 32-bit word. The 32-bit word shifts to internal program memory during the load of the 256-instruction word kernel.

The following example shows a 48-bit instructions executed.

[0x90000] 0x112233445566

[0x90001] 0x7788AABBCCDD

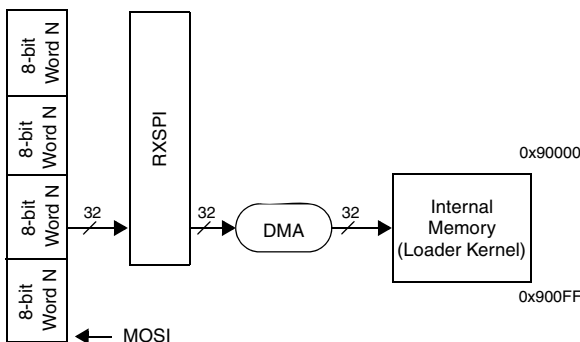


Figure 14-15. 8-Bit SPI Host Packing

The 8-bit SPI host packs or prearranges the data as:

SPI word 1 = 0x66
SPI word 2 = 0x55
SPI word 3 = 0x44
SPI word 4 = 0x33
SPI word 5 = 0x22
SPI word 6 = 0x11
SPI word 7 = 0xDD
SPI word 8 = 0xCC

SPI word 9 = 0xBB
 SPI word 10 = 0xAA
 SPI word 11 = 0x88
 SPI word 12 = 0x77

The initial boot of the 256-word loader kernel requires an 8-bit host to transmit 1536 8-bit words. The SPI DMA count value of 0x180 is equal to 384 words. Since one 32-bit word is created from four packed 8-bit words, the total number of 8-bit words transmitted is 1536.

For all boot modes, the VisualDSP++ loader automatically outputs the correct word width and count based on the project settings. For more information, see the VisualDSP++ tools documentation.

Slave Boot Mode

In slave boot mode, the host processor initiates the booting operation by activating the $\overline{\text{SPICLK}}$ signal and asserting the $\overline{\text{SPIDS}}$ signal to the active low state. The 256-word kernel is loaded 32 bits at a time, through the SPI receive shift register (RXSR). To receive 256 instructions (48-bit words) properly, the SPI DMA initially loads a DMA count of 0x180 (384) 32-bit words, which is equivalent to 0x100 (256) 48-bit words.



The processor's $\overline{\text{SPIDS}}$ pin should not be tied low. When in SPI slave mode, including booting, the $\overline{\text{SPIDS}}$ signal is required to transition from high to low. SPI slave booting uses the default bit settings shown in [Table 14-11](#).

Table 14-11. SPI Slave Boot Bit Settings

Bit	Setting	Comment
SPIEN	Set (= 1)	SPI enabled
SPIMS	Cleared (= 0)	Slave device
MSBF	Cleared (= 0)	LSB first
WL	10, 32-bit SPI	Receive shift register word length

Bootling

Table 14-11. SPI Slave Boot Bit Settings (Cont'd)

Bit	Setting	Comment
DMISO	Set (= 1) MISO	MISO disabled
SENDZ	Cleared (= 0)	Send last word
SPIRCV	Set (= 1)	Receive DMA enabled
CLKPL	Set (= 1)	Active low SPI clock
CPHASE	Set (= 1)	Toggle SPICLK at the beginning of the first bit

The SPI DMA channel is used when downloading the boot kernel information to the processor. At reset, the DMA parameter registers are initialized to the values listed in [Table 14-12](#).

Table 14-12. Parameter Initialization Value for Slave Boot

Parameter Register	Initialization Value	Comment
SPICTL	0x0000 4D22	
SPIDMAC	0x0000 0007	Enabled, receive, initialized on completion
IISPI	0x0008 0000	Start of Block 0 NW memory
IMSPI	0x0000 0001	32-bit data transfers
CSPI	0x0000 0180	

Master Boot

In master boot mode, the ADSP-21367/8/9 and ADSP-2137x processors initiate the booting operation by:

1. Activating the `SPICLK` signal and asserting the `FLAG0` signal to the active low state.
2. Writing the read command `0x03` and address `0x00` to the slave device as shown in [Figure 14-12 on page 14-43](#).

Master boot mode is used when the processor is booting from an SPI-compatible serial PROM, serial FLASH, or slave host processor. The specifics of booting from these devices are discussed individually. On reset, the interface starts up in master mode performing a 384 32-bit word DMA transfer.

SPI master booting uses the default bit settings shown in [Table 14-13](#).

Table 14-13. SPI Master Boot Mode Bit Settings

Bit	Setting	Comment
SPIEN	Set (= 1)	SPI Enabled
SPIMS	Set (= 1)	Master device
MSBF	Cleared (= 0)	LSB first
WL	10	32-bit SPI receive shift register word length
DMISO	Cleared (= 0)	MISO enabled
SENDZ	Set (= 1)	Send zeros
SPIRCV	Set (= 1)	Receive DMA enabled
CLKPL	Set (= 1)	Active low SPI clock
CPHASE	Set (= 1)	Toggle SPICLK at the beginning of the first bit

The SPI DMA channel is used when downloading the boot kernel information to the processor. At reset, the DMA parameter registers are initialized to the values listed in [Table 14-14](#).

Table 14-14. Parameter Initialization Value for Master Boot

Parameter Register	Initialization Value	Comment
SPICTL	0x0000 5D06	
SPIBAUD	0x0064	CCLK/400 = 500 KHz@ 200 MHz
SPIFLG	0xfe01	FLAG0 used as slave-select

Table 14-14. Parameter Initialization Value for Master Boot (Cont'd)

Parameter Register	Initialization Value	Comment
SPIDMAC	0x0000 0007	Enable receive, interrupt on completion
IISPI	0x0008 0000	Start of block 0 normal word memory
IMSPI	0x0000 0001	32-bit data transfers
CSPI	0x0000 0180	0x100 instructions = 0x180 32-bit words

From the perspective of the processor, there is no difference between booting from the three types of SPI slave devices. Since SPI is a full-duplex protocol, the processor is receiving the same amount of bits that it sends as a read command. The read command comprises a full 32-bit word (which is what the processor is initialized to send) comprised of a 24-bit address with an 8-bit opcode. The 32-bit word that is received while this read command is transmitted is thrown away in hardware, and can never be recovered by the user. Because of this, special measures must be taken to guarantee that the boot stream is identical in all three cases. The processor boots in least significant bit first (LSBF) format, while most serial memory devices operate in most significant bit first (MSBF) format. Therefore, it is necessary to program the device in a fashion that is compatible with the required LSBF format.

Also, because the processor always transmits 32 bits before it begins reading boot data from the slave device, it is necessary for the VisualDSP++ tools to insert extra data to the boot image (in the loader file) if using memory devices that do not use the LSBF format. VisualDSP++ has built-in support for creating a boot stream compatible with both endian formats and devices requiring 16-bit and 24-bit addresses, as well as those requiring no read command at all.

Figure 14-16 shows the initial 32-bit word sent out from the processor. As shown in the figure, the processor initiates the SPI master boot process by writing an 8-bit opcode (LSB first) to the slave device to specify a read operation. This read opcode is fixed to 0xC0 (0x03 in MSBF format).

Following that, a 24-bit address (all zeros) is always driven by the processor. On the following `SPICLK` cycle (cycle 32), the processor expects the first bit of the first word of the boot stream. This transfer continues until the kernel has finished loading the user program into the processor.

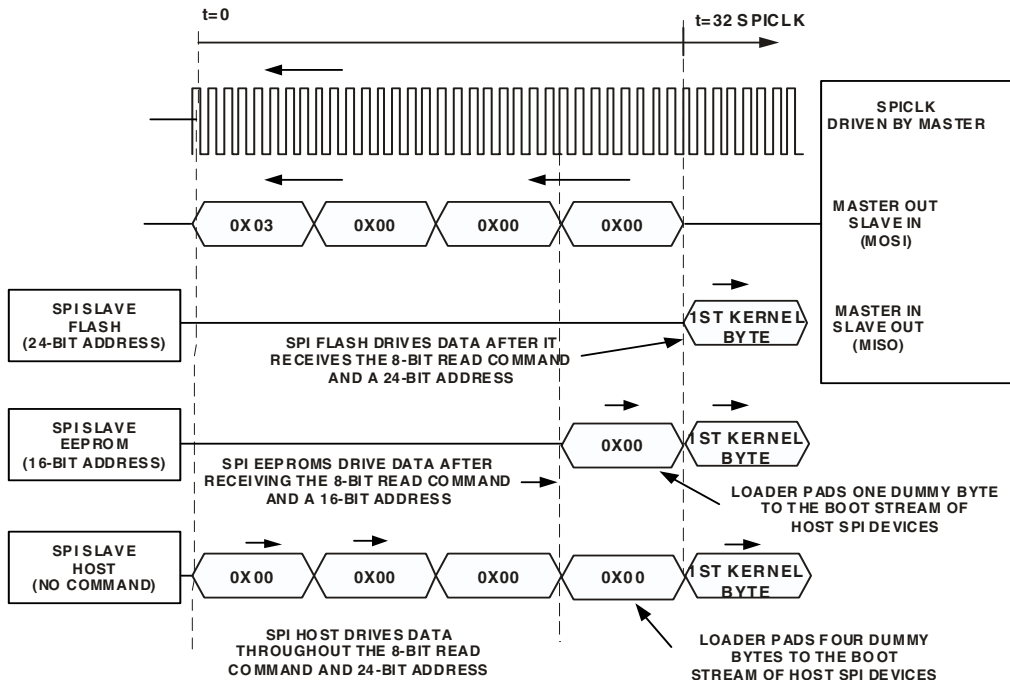


Figure 14-16. SPI Master Mode Booting Using Various Serial Devices

Bootng From an SPI Flash

For SPI flash devices, the format of the boot stream is identical to that used in SPI slave mode, with the first byte of the boot stream being the first byte of the kernel. This is because SPI flash devices do not drive out data until they receive an 8-bit command and a 24-bit address.

Booting From an SPI PROM (16-Bit address)

Figure 14-16 shows the initial 32-bit word sent out from the processor from the perspective of the serial PROM device.

As shown in Figure 14-16, SPI EEPROMS only require an 8-bit opcode and a 16-bit address. These devices begin transmitting on clock cycle 24. However, because the processor is not expecting data until clock cycle 32, it is necessary to pad an extra byte to the beginning of the boot stream when programming the PROM. In other words, the first byte of the kernel is the second byte of the boot stream. The VisualDSP++ tools automatically handles this in the loader file generation process for SPI PROM devices.

Booting From an SPI Host Processor

Typically, host processors in SPI slave mode transmit data on every `SPICLK` cycle. This means that the first four bytes that are sent by the host processor are part of the first 32-bit word that is thrown away by the processor (see Figure 14-16). Therefore, it is necessary to pad an extra four bytes to the beginning of the boot stream when programming the host. For example, the first byte of the kernel is the fifth byte of the boot stream. VisualDSP++ automatically handles this in the loader file generation process.

Data Delays, Latencies, and Throughput

Table 14-15 specifies latencies and throughput for the ADSP-21367/8/9 and ADSP-2137x processors. *Latency* is defined as the number of cycles, after the first cycle, required to complete the operation. A zero wait state memory has a latency of zero. A single wait state memory has a data delay of one. *Throughput* is the maximum rate at which the operation is performed. Data delay and throughput are the same whether the access is from a host processor or from another SHARC processor.

Table 14-15. Latencies and Throughput

Operation	Minimum Data Delay (cycles)	Maximum Throughput (cycles/ transfer)
Interrupts ($\overline{\text{IRQ}}2-0$)	3	-
DMA chain initialization	7–11	-
Serial ports ¹	35	32

1 Processor-to -processor transfers using 32-bit words.

Execution Stalls

The following events can cause an execution stall for the ADSP-21367/8/9 and ADSP-2137x SHARC processors:

- One cycle on a program memory data access with instruction cache miss
- Two cycles on non-delayed branches
- Two cycles on normal interrupts
- One to two cycles on short loops with small iterations
- On an IDLE instruction, execution is halted while waiting for an external event, such as an interrupt
- In a sequence of three instructions of the types shown below, the processor may stall for one cycle:

Instruction 1: Compute instruction affecting flags such as

$R2 = R3 - R4;$

Instruction 2: Conditional instruction involving post-modify addressing such as $\text{IF EQ DM}(I1, M1) = R15;$

Data Delays, Latencies, and Throughput

Instruction 3: Instruction involving post-modify addressing involving same I register such as `R0 = DM(I1,M2);`. This last instruction stalls the processor for one cycle.

- Any read reference to a memory-mapped register located physically within the core (registers like `SYSCTL`, which are not situated in the IOP) requires two cycles; therefore, the processor stalls for one cycle.
- Any read reference to a memory-mapped register located within a peripheral such as the SPI, SPORTS, or IDP requires a minimum of four cycles; so the minimum stall is three cycles.
- Any reference to a memory-mapped register in a conditional instruction stalls the processor for one extra cycle (with respect to an unconditional instruction).
- Writes to program memory breakpoint address registers (`PMDAS`, `PMDAE`) have an effect latency of one cycle. Therefore, the breakpoint address ranges are effective one cycle after the breakpoint address registers are initialized.

DAG Stalls

One cycle hold on register conflict. For more information on DAG operations, see the *ADSP-2136x SHARC Processor Programming Reference*.

Memory Stalls

One cycle on PM and DM bus access to the same block of internal memory.

When a new external memory instruction fetch occurs on the ADSP-2137x processor due to a jump from internal to external memory, or after a cache hit while executing instructions from external memory, there is one stall cycle present in the fetch1 stage. This stall avoids resource conflicts at the cache interface.

The `FLUSH CACHE` instruction has an effect latency of one instruction when executing program instructions from internal memory, and two instructions when executing from external memory. This applies to the ADSP-2137x processors only.

A one cycle stall is generated whenever an instruction that contains a conditional external memory access is in the decode stage, where the evaluation of the condition is dependent on the outcome of the previous instruction in address stage. It applies to all kinds of conditions, except for conditions based on FLAG status. The following is an example.

```
f12=f11+f10;  
if eq then dm(ext) = r0;
```

There is one cycle latency between a multiplier status change and an arithmetic loop abort. This extra cycle is machine cycle and not the instruction cycle. Therefore, if there is a pipeline stall (due to external memory access etc.) then the latency does not apply.

IOP Register Stalls

Read of the IOP registers takes a minimum of four cycles, therefore the processor stalls for at least three cycles.

DMA Stalls

- One cycle if an access to a DMA parameter register conflicts with the DMA address generation (for example, writing to the register while a register update is taking place) or reading while a DMA register conflicts with DMA chaining.
- Attempting to write to (or read from) a full (or empty) DMA buffer causes the core to hang indefinitely, unless the BHD (buffer hang disable) bit for that peripheral is set (for example in the corresponding SPCTLx register for a serial port).

IOP Buffer Stalls

Table 14-16 shows the number of stalls incurred with the I/O processor when writing to a full buffer or reading from an empty buffer.

Table 14-16. Latencies and Throughput

Operation	Minimum Data Delay (cycles)	Maximum Throughput (cycles/ transfer)
Interrupts ($\overline{\text{IRQ}}2-0$)	3	-
DMA chain initialization	7–11	-
Serial ports ¹	35	32

1 ADSP-2136x SHARC processor-to-ADSP-2136x SHARC processor transfers using 32-bit words.

A REGISTER REFERENCE


The ADSP-21367/8/9 and ADSP-2137x processors have general-purpose and dedicated registers in each of their functional blocks. The register reference information for each functional block includes bit definitions, initialization values, and memory-mapped addresses (for I/O processor registers). Information on each type of register is available at the following locations:

- [“I/O Processor Registers” on page A-2](#)
- [“External Port Registers” on page A-10](#)
- [“Serial Port Registers” on page A-29](#)
- [“Serial Peripheral Interface Registers” on page A-52](#)
- [“Input Data Port Registers” on page A-65](#)
- [“Pulse Width Modulation Registers” on page A-78](#)
- [“Sample Rate Converter Registers” on page A-97](#)
- [“Sony/Philips Digital Interface Registers” on page A-86](#)
- [“DAI/DPI Registers” on page A-109](#)
- [“UART Control and Status Registers” on page A-118](#)
- [“Two Wire Interface Registers” on page A-130](#)
- [“Precision Clock Generator Registers” on page A-155](#)
- [“Peripheral Interrupt Priority Control Registers” on page A-164](#)

I/O Processor Registers


- “Power Management Control Register (PMCTL)” on page A-170
- “Hardware Breakpoint Control Register” on page A-175
- “Enhanced Emulation Status Register” on page A-179

When writing programs, it is often necessary to set, clear, or test bits in the processor’s registers. While these bit operations can all be done by referring to the bit’s location within a register or (for some operations) the register’s address with a hexadecimal number, it is much easier to use symbols that correspond to the bit or register name. For convenience and consistency, Analog Devices provides a header file that provides these bit and registers definitions. An `#include` file is provided with VisualDSP++ tools and can be found in the `VisualDSP/2136x/include` directory.

 Many registers have reserved bits. When writing to a register, programs may only clear (write zero) the register’s reserved bits.

I/O Processor Registers

The I/O processor’s registers are accessible as part of the processor’s memory map. These registers occupy addresses 0x0000 0000 through 0x0003 FFFF of the memory map. The I/O registers control the following DMA operations: serial port, serial peripheral interface port (SPI), and input data port (IDP), and internal memory-to-memory transfers.

 I/O processor registers have a one cycle effect latency (changes take effect on the second cycle after the change).

Since the I/O processor’s registers are part of the processor’s memory map, buses access these registers as locations in memory. While these registers act as memory-mapped locations, they are separate from the processor’s internal memory and have different bus access. One bus can access one I/O processor register from one I/O processor register group at a time.

When there is contention among the buses for access to registers in the same I/O processor register group, the processor arbitrates register access as follows:

- Data memory (DM) bus accesses
- Program memory (PM) bus accesses
- I/O processor (IO) bus (lowest priority) accesses

The bus with highest priority gets access to the I/O processor register group, and the other buses are held off from accessing that I/O processor register group until that access has been completed.

Since the I/O processor registers are memory-mapped, the processor's architecture does not allow programs to directly transfer data between these registers and other memory locations, except as part of a DMA operation. To read or write I/O processor registers, programs must use the processor core registers.

The register names for I/O processor registers are not part of the processor's assembly syntax. To ease access to these registers, programs should use the header file containing the registers' symbolic names and addresses.

Notes on Reading Register Drawings

The register drawings in this appendix provide “at a glance” information about the register in question. They are designed to give experienced users basic information about a register and its bits. When using these registers, the following should be noted.

1. The register name and address are provided in the upper left-hand corner of the drawing.
2. The bit settings in the drawings represent their state at reset.

I/O Processor Registers

3. In cases where there are multiple registers that have the same bits (such as serial ports), one register drawing is shown and the names and addresses of the others are simply listed.
4. The bit descriptions are intentionally brief. More detailed information can be found in the tables that follow the register drawings and in the chapters that describe the particular module.
5. The VisualDSP++ tools suite contains the complete listing of registers in a header file, `def21369.h` and `def21371.h`.

System Control Register (SYSCTL)

The `SYSCTL` register configures memory use, interrupts, and many aspects of pin multiplexing. (For more information, see “Pin Multiplexing” on page 14-2.) This register’s address is 0x30024. The reset value for this register is 0. Bit descriptions for this register are shown in Figure A-1 and Figure A-2, and described in Table A-1.

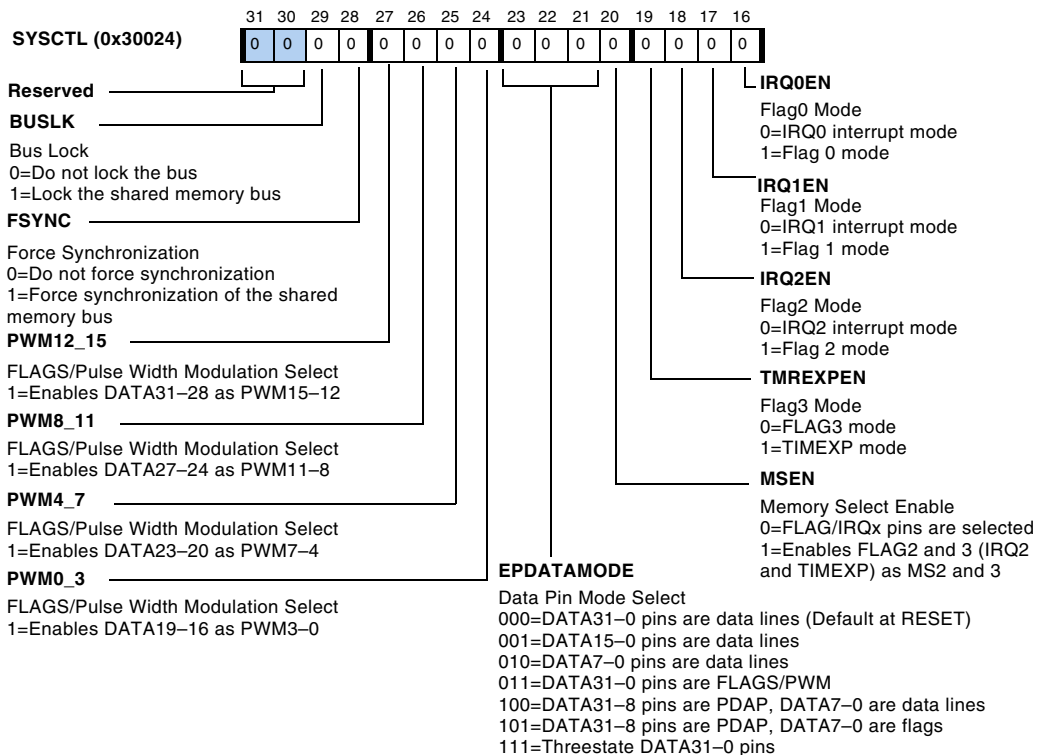


Figure A-1. SYSCTL Register (Bits 16–31)

I/O Processor Registers

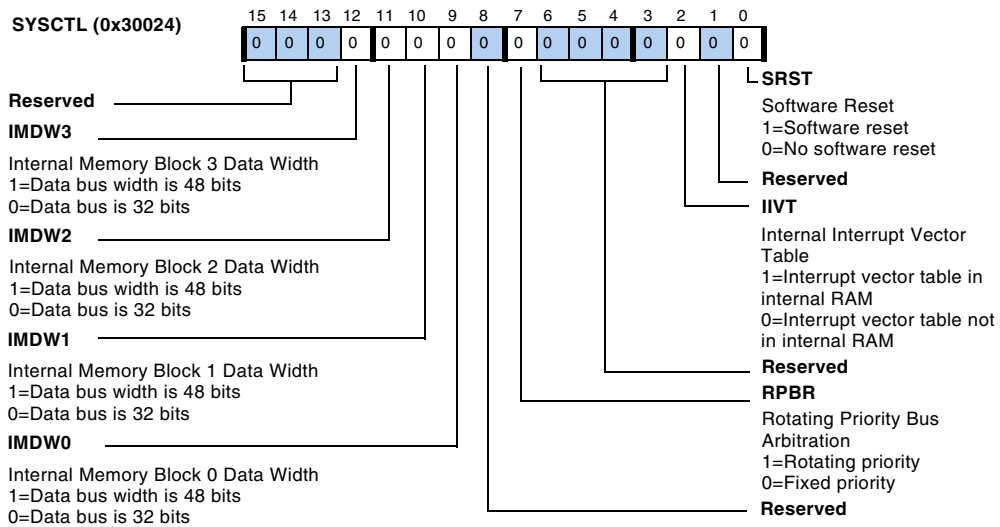


Figure A-2. SYSCTL Register (Bits 15–0)

Table A-1. SYSCTL Register Bit Descriptions

Bit	Name	Description
0	SRST	Software Reset. When set, this bit resets the processor and the processor responds to the non-maskable RSTI interrupt and clears (=0) SRST. Permits core writes. 0 = No software reset 1 = Software reset
1	Reserved	
2	IIVT	Internal Interrupt Vector Table. This bit forces placement of the interrupt vector table at address 0x0008 0000 regardless of booting mode or allows placement of the interrupt vector table as selected by the booting mode. Permits core writes. 0 = Interrupt vector table not in internal RAM 1 = Interrupt vector table in internal RAM
6–3	Reserved	

Table A-1. SYSCTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
7	RBPR	Rotating Priority Bus Arbitration. This bit enables or disables priority rotation among DMA channels. Permits core writes. 0 = Arbiter uses fixed priority 1 = Arbiter uses rotating priority
8	Reserved	
9	IMDW0	Internal Memory Data Width 0. Selects the data access size for internal memory block0 as 48- or 32-bit data. Permits core writes. 0 = Data bus width is 32 bits 1 = Data bus width is 48 bits
10	IMDW1	Internal Memory Data Width 1. Selects the data access size for internal memory block1 as 48- or 32-bit data. Permits core writes. 0 = Data bus width is 32 bits 1 = Data bus width is 48 bits
11	IMDW2	Internal Memory Data Width 2. Selects the data access size for internal memory block2 as 48- or 32-bit data. Permits core writes. 0 = Data bus width is 32 bits 1 = Data bus width is 48 bits
12	IMDW3	Internal Memory Data Width 3. Selects the data access size for internal memory block3 as 48- or 32-bit data. Permits core writes. 0 = Data bus width is 32 bits 1 = Data bus width is 48 bits
15-13	Reserved	
16	IRQ0EN	Flag0 Interrupt Mode. 0 = Flag0 pin is a general-purpose I/O pin. Permits core writes. 1 = Flag0 pin is allocated to interrupt request $\overline{\text{IRQ0}}$.
17	IRQ1EN	Flag1 Interrupt Mode. 0 = Flag1 pin is a general-purpose I/O pin. Permits core writes. 1 = Flag1 pin is allocated to interrupt request $\overline{\text{IRQ1}}$.
18	IRQ2EN	Flag2 Interrupt Mode. 0 = Flag2 pin is a general-purpose I/O pin. Permits core writes. 1 = Flag2 pin is allocated to interrupt request $\overline{\text{IRQ2}}$.

Table A-1. SYSCTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
19	TMREXPEN	Flag Timer Expired Mode. 0 = Flag3 pin is a general-purpose I/O pin. Permits core writes. 1 = Flag3 pin output is timer expired signal (TIMEXP).
20	MSEN	Memory Select. Selects FLGx or \overline{MSx} . Detailed modes of programming for these bits are given in “Programming Flags” on page 14-9. 0=FLAG/TRQ \overline{x} pins are selected 1=Enables FLAG2 and 3 (TRQ $\overline{2}$ and TIMEXP) as $\overline{MS2}$ and 3
23–21	EPDATA	Data Pin Mode Select. 000 = DATA31–0 pins are data lines (default at RESET) 001 = DATA15–0 pins are data lines 010 = DATA7–0 pins are data lines 011 = DATA31–0 pins are FLAGS/PWM 100 = DATA31–8 pins are PDAP, DATA7–0 are data lines 101 = DATA31–8 pins are PDAP, DATA7–0 are flags 111 = Threestate DATA31–0 pins.
24	PWM0_3	Pulse Width Modulation Select. When set (=1), enables DATA19–16 as PWM3–0.
25	PWM4_7	Pulse Width Modulation Select. When set (=1), enables DATA23–20 as PWM7–4.
26	PWM8_11	Pulse Width Modulation Select. When set (=1), enables DATA27–24 as PWM11–8.
27	PWM12_15	Pulse Width Modulation Select. When set (=1), enables DATA31–28 as PWM15–12.
28	FSYNC	Force Synchronization of the Shared Memory Bus. 0 = Do not force synchronization 1 = Force synchronization of the shared memory bus
29	BUSLK	Bus Lock Request. Requests bus lock where the processor maintains bus master control (if set, =1) or does not request bus lock (normal bus master control) id cleared (=0).
31–30	Reserved	

System Status Register (SYSTAT)

The SYSTAT register's address is 0x180F. The reset value has all bits initialized to zero, except for the IDC, CRBM, and CRAT fields, which are set from values on the ADSP-21367/8/9 and ADSP-2137x's pins. This register is shown in Figure A-3 and described in Table A-2.

SYSTAT(0x180F)

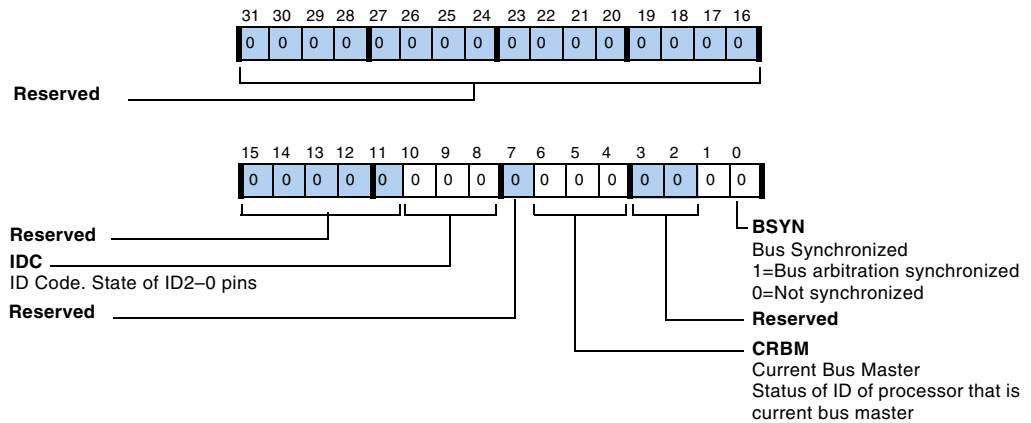


Figure A-3. SYSTAT Register

Table A-2. System Status Register (SYSTAT) Bit Descriptions

Bit	Name	Description
0	BSYN	Bus Synchronized. Indicates whether the processor's bus arbitration logic is synchronized (if set, =1) or is not synchronized (if cleared, =0, reset value).
3-1	Reserved (reset value =0)	
6-4	CRBM	Current Bus Master. These bits indicate the ID of the processor that currently is the bus master in a multiprocessor system. Because CRBM is only valid for DSPs with ID inputs other than zero (for example, a multiprocessor system), the processor keeps CRBM set to 001 when ID equals 000. The reset value of CRBM is undefined.

External Port Registers

Table A-2. System Status Register (SYSTAT) Bit Descriptions (Cont'd)

Bit	Name	Description
7	Reserved (reset value =0)	
10–8	IDC	ID Code. These bits indicate the state of the ID pins on the processor. The reset value of IDC is undefined.
31-11	Reserved (reset value =0)	

External Port Registers

The following registers are used to control asynchronous memory interface (AMI), the SDRAM controller (SDC), and the shared memory interface (ADSP-21368 only).

External Port Control Register (EPCTL)

The external port control register can be programmed to arbitrate the accesses between the processor core and DMA, and between different DMA channels. These registers are shown in [Figure A-4](#) and described in [Table A-3](#).

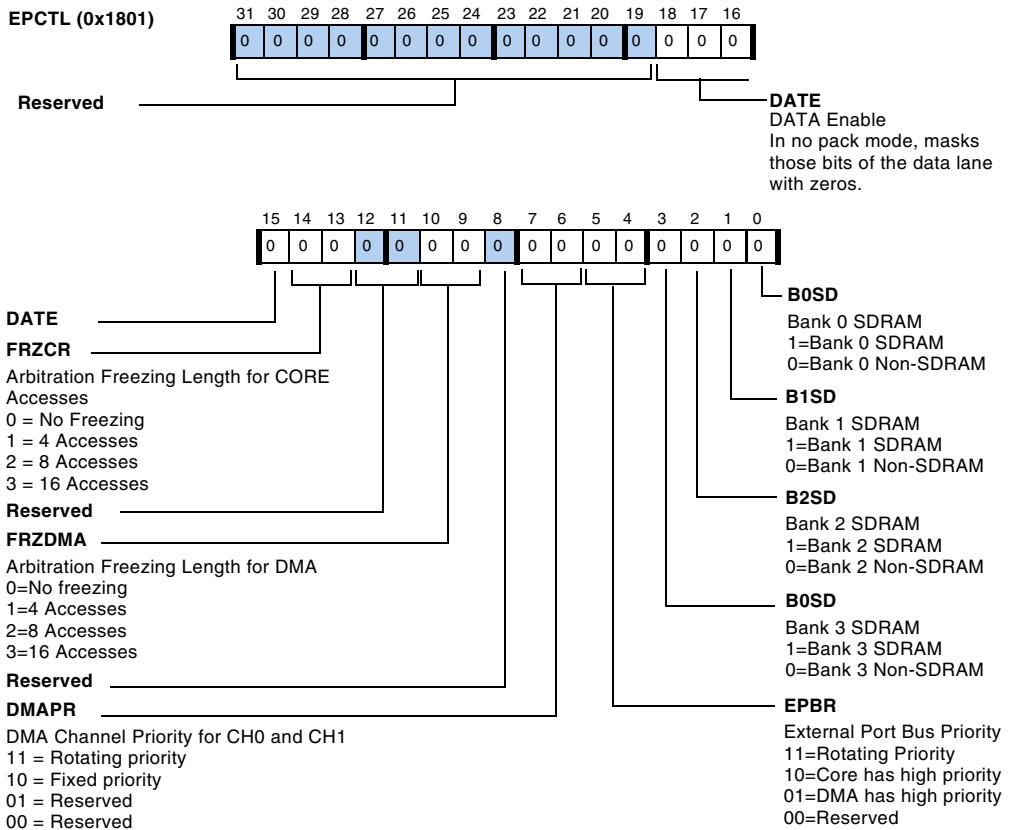


Figure A-4. External Port Control Register

Table A-3. EPCTL Register Bit Descriptions

Bit	Name	Description
0	B0SD	Select Bank 0 SDRAM. 1 = Bank 0 SDRAM 0 = Bank 0 Non-SDRAM
1	B1SD	Select Bank 1 SDRAM. 1 = Bank 1 SDRAM 0 = Bank 1 Non-SDRAM

External Port Registers

Table A-3. EPCTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
2	B2SD	Select Bank 2 SDRAM. 1 = Bank 2 SDRAM 0 = Bank 2 Non-SDRAM
3	B3SD	Select Bank 3 SDRAM. 1 = Bank 3 SDRAM 0 = Bank 3 Non-SDRAM
5–4	EPBR	External Port Bus Priority. 00 = Reserved 01 = DMA has high priority 10 = Core has high priority 11 = Rotating Priority
7–6	DMAPR	DMA Channel Priority for CH0 and CH1. 00 = Reserved 01 = Reserved 10 = Fixed priority 11 = Rotating priority
8	Reserved	
10–9	FRZDMA	Arbitration Freezing Length for DMA. 0 = No freezing 1 = 4 Accesses 2 = 8 Accesses 3 = 16 Accesses
12–11	Reserved	
14–13	FRZCR	Arbitration Freezing Length for CORE Accesses. 0 = No freezing 1 = 4 Accesses 2 = 8 Accesses 3 = 16 Accesses

Table A-3. EPCTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
18–15	DATE	Data Enable. In no pack mode of the sdram/ami memory controller, masks those bits of the data lane with zeros. The data lane is 8 bits. The 32-bit data bus has four data lanes. DATA31–0 is mapped to {dl3, dl2, dl1, dl0} For example, If DATE is 1010, then dl3 and dl1 are masked with zeros.
31–19	Reserved	

External Port Registers

External Port DMA Control Registers (DMACx)

The DMAC0-1 registers control the DMA function of their respective DMA channels. These registers are shown in [Figure A-5](#) and described in [Table A-4](#).

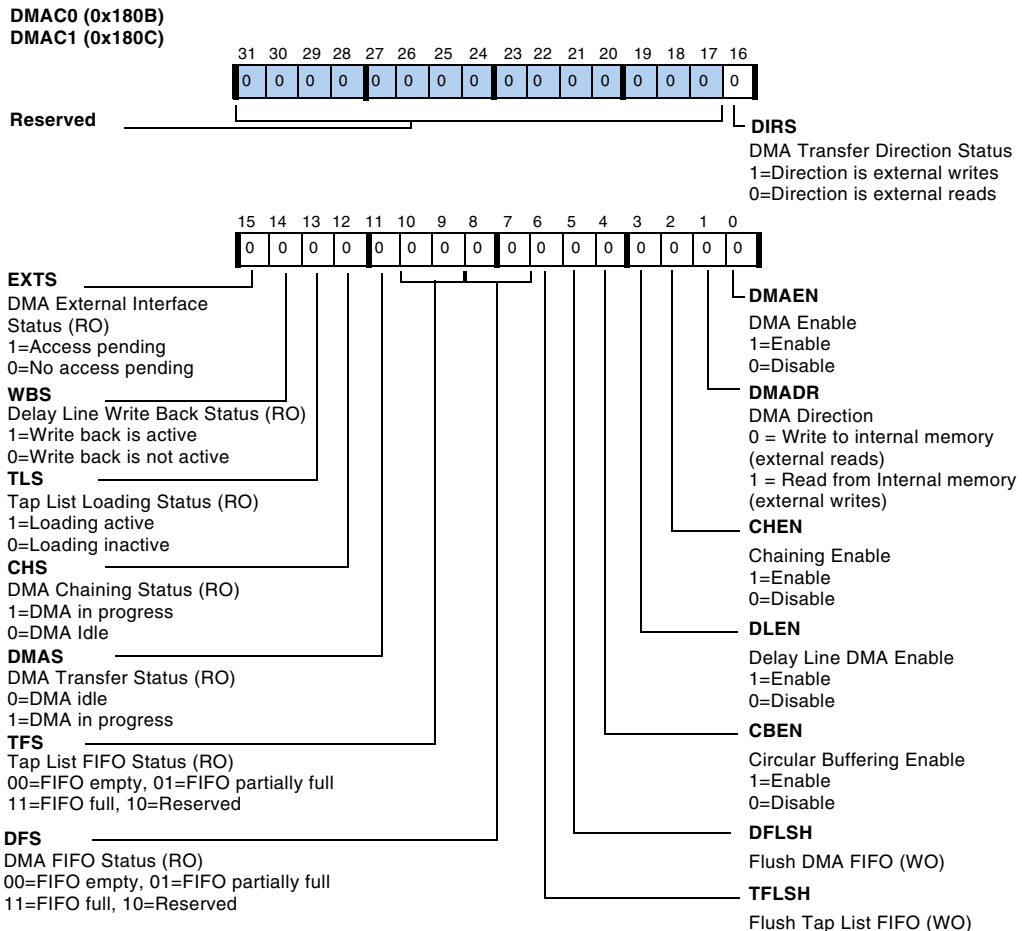


Figure A-5. External Port DMA Registers

Table A-4. External Port DMA Register Bit Descriptions

Bit	Name	Description
0	DMAEN	DMA Enable. 0 = External port channel x DMA is disabled 1 = Enable External port DMA for channel x
1	DMADR	DMA Direction 0 = Write to internal memory (external reads) 1 = Read from internal memory (external writes) Note: If delay line DMA is enabled then the DMADR bit doesn't have any effect. For delay line DMA, transfer direction depends on the state of delay line transfers
2	CHEN	Enable Chaining. 0 = Chaining disabled 1 = Chaining enabled
3	DLEN	Enable Delay Line DMA. DLEN is applicable only if CHEN=1. 0 = Delay-line DMA disabled 1 = Delay-line DMA enabled
4	CBEN	Circular Buffering Enable. 0 = Disables circular buffering with delay line DMA 1 = Enables circular buffering with delay line DMA Circular Buffering can be used with normal DMA as well
5	DFLSH	Flush DMA FIFO (write-only).
6	TFLSH	Flush Tap List FIFO (write-only).
8–7	DFS	DMA FIFO Status (read-only). 00 = FIFO empty 01 = FIFO partially full 11 = FIFO full 10 = Reserved
10–9	TFS	Tap List FIFO Status (read-only). 00 = FIFO empty 01 = FIFO partially full 11 = FIFO full 10 = Reserved
11	DMAS	DMA Transfer Status (read-only). 0 = DMA idle 1 = DMA in progress

External Port Registers

Table A-4. External Port DMA Register Bit Descriptions (Cont'd)

Bit	Name	Description
12	CHS	DMA Chaining Status (read-only). 0 = DMA chain loading is not active 1 = DMA chain loading is active
13	TLS	Tap List Loading Status (read-only). 0 = Tap list loading is not active 1 = Tap list loading is active
14	WBS	Delay Line Write Pointer Write Back Status (read-only). 0 = Write pointer write back is not active 1 = Write pointer write back is active
15	EXTS	DMA External Interface Status (read-only). 0 = DMA external interface does not have any access pending 1 = DMA external interface has access pending
16	DIRS	DMA Transfer Direction Status (read-only). 0 = DMA direction is external reads 1 = DMA direction is external writes Note: This is useful for delay line DMA where the transfer direction changes with the state of the DMA state machine. For normal DMA, DIRS will reflect the state of the DMADR bit.
31–17	Reserved	

AMI Control Registers (AMICTLx)

The AMICTL0-3 registers control the mode of operations for the four banks of external memory. These registers are shown in [Figure A-6](#) and described in [Table A-5](#).

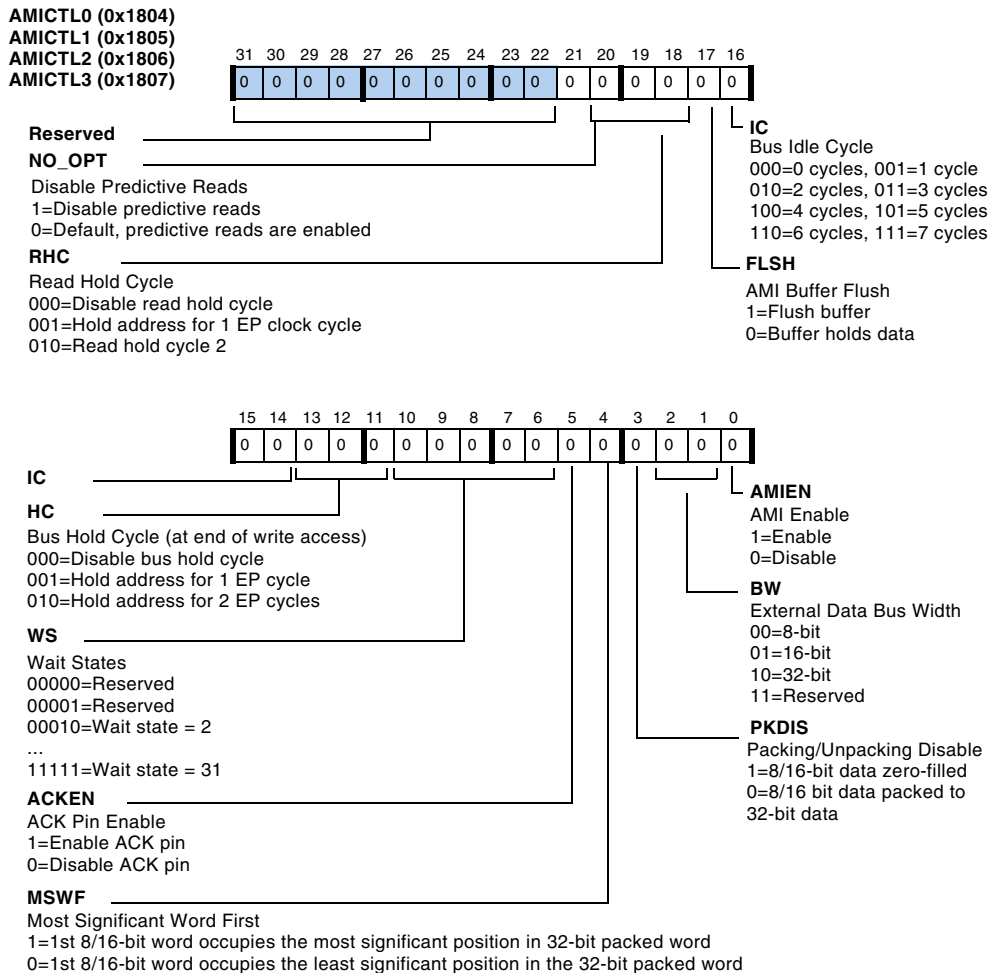


Figure A-6. AMICTLx Registers

External Port Registers

Table A-5. AMICTLx Register Bit Descriptions

Bit	Name	Description
0	AMIEN	AMI Enable. 0 = AMI is disabled 1 = AMI is enabled
2–1	BW	External Data Bus Width. 00 = 8 bits 01 = 16 bits 10 = 32 bits 11 = Reserved
3	PKDIS	Disable Packing/Unpacking. 0 = 8/16-bit data received packed to 32-bit data. Similarly, 32-bit data to be transmitted is unpacked to two 16-bit data or four 8-bit data. 1 = 8/16-bit data received zero-filled, for transmitted data only 16-bit or the 8-bit LSB part of the 32-bit data is written to external memory.
4	MSWF	Most Significant Word First. Applicable only with packing disabled (PKDIS=0). 0 = 1st 8/16-bit word read/write occupies the least significant position in the 32-bit packed word. 1 = 1st 8/16-bit word read/write occupies the most significant position in the 32-bit packed word.
5	ACKEN	Enable the ACK pin. If enabled, reads/writes to devices have to be extended by the corresponding devices by pulling ACK low. When ACKEN is set then the ACK pin is sampled after the waitstate value is programmed.
10–6	WS	Wait States. 00000 = Reserved (wait state value of 32 if used) 00001 = Reserved for internal use (can be used when the clock frequencies are low) 00010 = wait state = 2 external port clock cycles ... 11111 = Wait state = 31 external port clock cycles
13–11	HC	Bus Hold Cycle at the End of Write Access. 000 = Disable bus hold cycle 001 = Hold address for one external port clock cycle 010 = Hold address for two external port clock cycles

Table A-5. AMICTLx Register Bit Descriptions (Cont'd)

Bit	Name	Description
16–14	IC	Bus Idle Cycle. Idle cycle to be inserted whenever read from external memory is followed by a write to external memory – to avoid contention. 'IC' EP clock cycles are ensured between a read to write. 000 = 0 cycles, 001 = 1 cycle 010 = 2 cycles, 011 = 3 cycles 100 = 4 cycles, 101 = 5 cycles 110 = 6 cycles, 111 = 7 cycles
17	FLSH	AMI Buffer Flush (Write-only). 0 = Buffer holds the data 1 = Flush the buffer
20–18	RHC	Read Hold Cycle at the End of Read Access. Controls the delay between two reads. 000 = Disable read hold cycle 001 = Hold address for one external port clock cycle 010 = Hold address for two external port clock cycles
21	NO_OPT	Disable Predictive Reads. Default is predictive reads are enabled. For more information, see “SDRAM Read Optimization” on page 3-75.
31–22	Reserved	

AMI Status Register (AMISTAT)

This 32-bit, read-only register provides status information for the AMI interface and can be read at any time. This register is shown in [Figure A-7](#).

AMISTAT (0x180A)

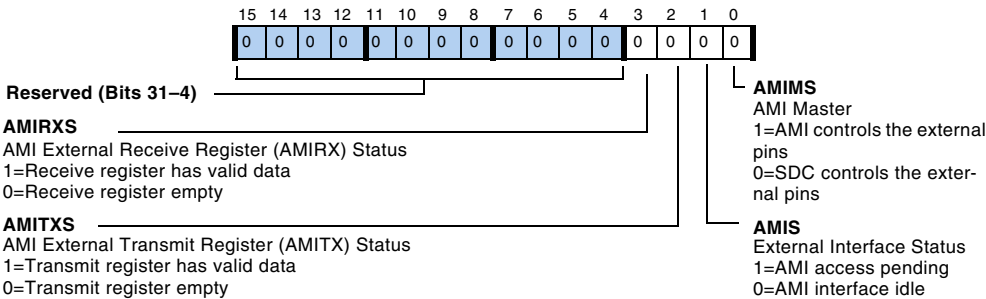


Figure A-7. AMISTAT Register

SDRAM Control Register (SDCTL)

The SDRAM memory control register includes all programmable parameters associated with the SDRAM access timing and configuration. This 32-bit register is located at address 0x1800 and is shown in [Figure A-8](#) and described in [Table A-6](#). For more information, see “SDRAM Control Register (SDCTL)” on page 3-39.

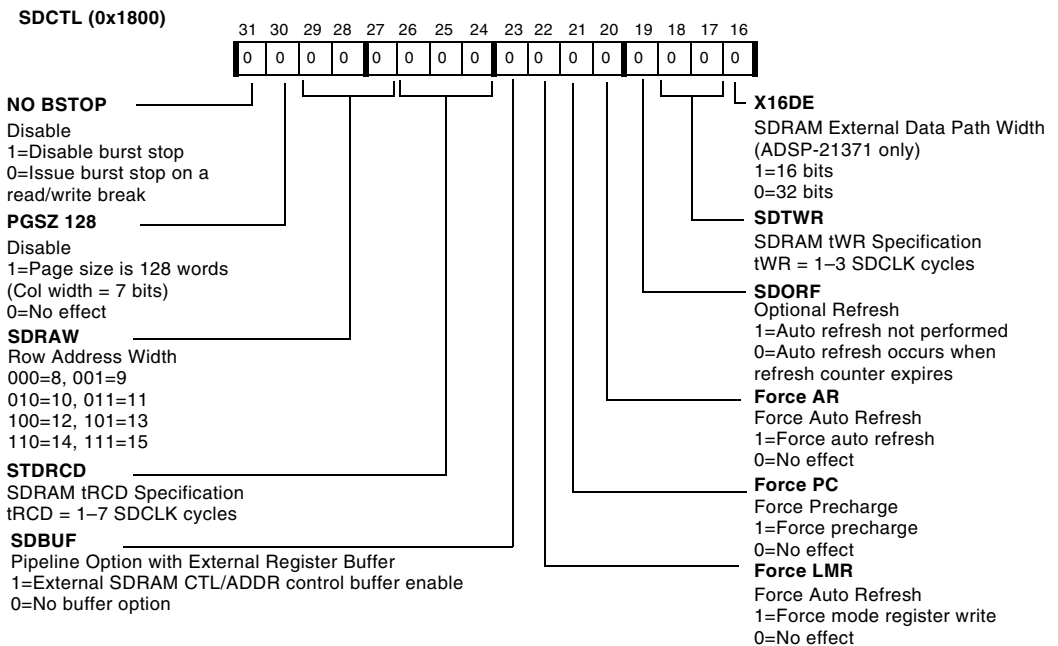


Figure A-8. SDRAM Control Register (Bits 16–31)

External Port Registers

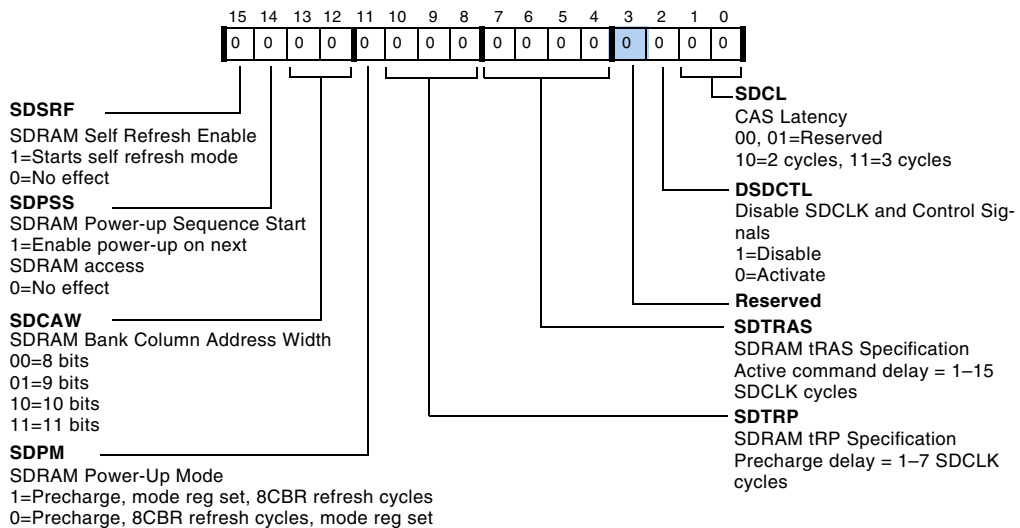


Figure A-9. SDRAM Control Register (Bits 0–15)

Table A-6. SDRAM Control Register Bit Descriptions

Bit	Name	Description
1–0	SDCL	SDRAM CAS Latency. The delay in clock cycles between when the SDRAM detects the read command and when it provides the data at its output pins. 00, 01 = Reserved 10 = 2 cycles 11 = 3 cycles
2	DSDCTL	Disable SDCLK and Control Signals. Used to enable or disable the SDC. If DSDCTL is disabled, any access to SDRAM address space does not occur externally. When DSDCTL is disabled, all SDC control pins are in their inactive states and the SDRAM clock SDCLK is not running. 1 = Disable 0 = Activate
3	Reserved.	

Table A-6. SDRAM Control Register Bit Descriptions (Cont'd)

Bit	Name	Description
7–4	SDTRAS	tRAS Specification. Based on the system clock frequency and the timing specifications of the SDRAM used. Programmed parameters apply to all four banks in the external memory. See t_{RAS} on page 3-35.
10–8	SDTRP	tRP Specification. Based on the system clock frequency and the timing specifications of the SDRAM used. Programmed parameters apply to all four banks in the external memory. See t_{RP} on page 3-35.
11	SDPM	SDRAM Power-Up Mode. The SDPM and SDPSS bits work together to specify and trigger an SDRAM power-up (initialization) sequence. If the SDPM bit is set (=1), the SDC does a precharge all command, followed by a load mode register command, followed by eight auto-refresh cycles. If the SDPM bit is cleared (=0), the SDC does a precharge all command, followed by eight auto-refresh cycles, followed by a load mode register command.
13–12	SDCAW	SDRAM Bank Column Address Width. 00 = 8 bits 01 = 9 bits 10 = 10 bits 11 = 11 bits
14	SDPSS	SDRAM Power-Up Sequence Start. The power-up sequence occurs and is followed immediately by the read or write transfer to SDRAM that is used to trigger the SDRAM power-up sequence. Note that there is a long latency for this first access to SDRAM because the SDRAM power-up sequence takes many cycles to complete. 1 = Enable power-up on next SDRAM access 0 = No effect
15	SDSRF	Self Refresh Enable. When the SDSRF bit is set to 1, self-refresh mode is triggered. Once the SDC completes any active transfers, the SDC executes the sequence of commands to put the SDRAM into self-refresh mode. The next access to the enabled SDRAM bank causes the SDC to execute the commands to exit from self-refresh and execute the access.

External Port Registers

Table A-6. SDRAM Control Register Bit Descriptions (Cont'd)

Bit	Name	Description
16	X16DE	SDRAM External Data Path Width. Selects whether the SDRAM interface is 32 or 16 bits wide. If X16DE = 0, DATA31–0 should be connected to the SDRAM. If X16DE = 1, DATA15–0 should be connected to the SDRAM and 16 to 32-bit packing is performed. (Valid for all processors except for the ADSP-21375 processor).
18–17	SDTWR	SDRAM tWR Specification. tWR = 1–3 SDCLK cycles. Based on the system clock frequency and the timing specifications of the SDRAM used. Programmed parameters apply to all four banks in the external memory. See tWR on page 3-35 .
19	SDORF	Optional Refresh. 1 = Auto refresh not performed 0 = Auto refresh occurs when refresh counter expires (see “Auto-Refresh” on page 3-70).
20	Force AR	Force Auto Refresh. 1 = Force auto refresh 0 = No effect
21	Force PC	Force Precharge. 1 = Force precharge 0 = No effect
22	Force LMR	Force Load Mode Register Write. This command performs a load mode register command immediately. This is in contrast to the normal load mode register set which requires some delay. This command performs a precharge all (if not precharged already) followed by a mode register write. (Valid for ADSP-2137x processors only).
23	SDBUF	Pipeline Option with External Register Buffer. 1 = External SDRAM CTL/ADDR control buffer enable 0 = No buffer option
26–24	SDTRCD	SDRAM tRCD Specification. tRCD = 1–7 SDCLK cycles. Based on the system clock frequency and the timing specifications of the SDRAM used. Programmed parameters apply to all four banks in the external memory. See tRCD on page 3-35 .

Table A-6. SDRAM Control Register Bit Descriptions (Cont'd)

Bit	Name	Description
29–27	SDRAW	Row Address Width. 000=8, 001=9 010=10, 011=11 100=12, 101=13 110=14, 111=15
30	PGSZ 128	Program the SDRAM Controller for Page Size of 128 Words. This bit allows programs to configure the SDC for a page size of 128 words (7 bits) which supports most available 32 Mb SDRAMs. 1 = Page size 128 words. Column width = 7 bits, override CAW settings. 0 = No effect, page size decided by SDCAW bits. (Valid for ADSP-2137x processors only).
31	NO BSTOP	No Burst Mode. This bit is used to select between full page burst or no burst mode (BL=1). If set (=1), no burst mode is active and the burst stop command is ignored. If cleared, full page burst is active using the burst stop command for access interruption. This bit must be cleared if the SDRAM does not support no burst mode but supports full page burst. (Valid for ADSP-2137x processors only).

SDRAM Control Status Register (SDSTAT)

The SDRAM control status register provides information on the state of the SDC. This information can be used to determine when it is safe to alter SDC control parameters or as a debug aid. This register is located at address 0x1803 and is shown in [Figure A-10](#).

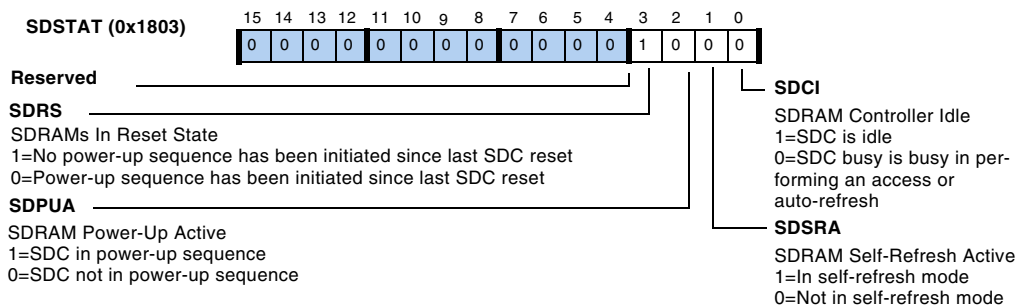


Figure A-10. SDRAM Control Status Register

SDRAM Refresh Rate Control Register (SDRRC)

The SDRAM refresh rate control register provides a flexible mechanism for specifying the auto-refresh timing. The SDC provides a programmable refresh counter which has a period based on the value programmed into the `RDIV` field of this register, that coordinates the supplied clock rate with the SDRAM device's required refresh rate. This register is located at address 0x1802 and is shown in [Figure A-11](#). For more information, see [“SDRAM Refresh Rate Control Register \(SDRRC\)”](#) on page 3-49 and for information on using the `S MODIFY` bit see [“SDRAM Read Optimization”](#) on page 3-75.

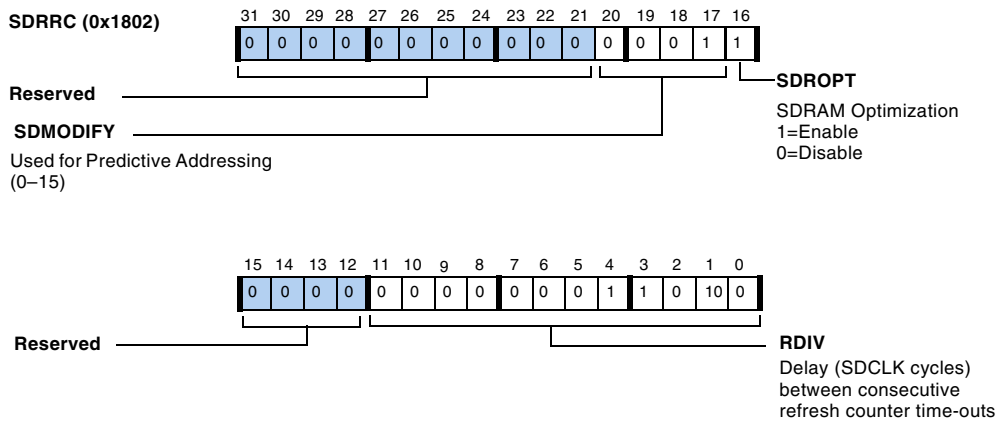


Figure A-11. SDRAM Control Refresh Rate Register

Memory-to-Memory DMA Register

The memory-to-memory (MTM) DMA register (MTMCTL) allows programs to transfer blocks of 64-bit data from one internal memory location to another. This transfer method uses two DMA channels, one for reading data and one for writing data. These transfers are controlled using the MTMCTL register shown in Figure A-12. For more information, see “Memory-to-Memory DMA” on page 2-48.

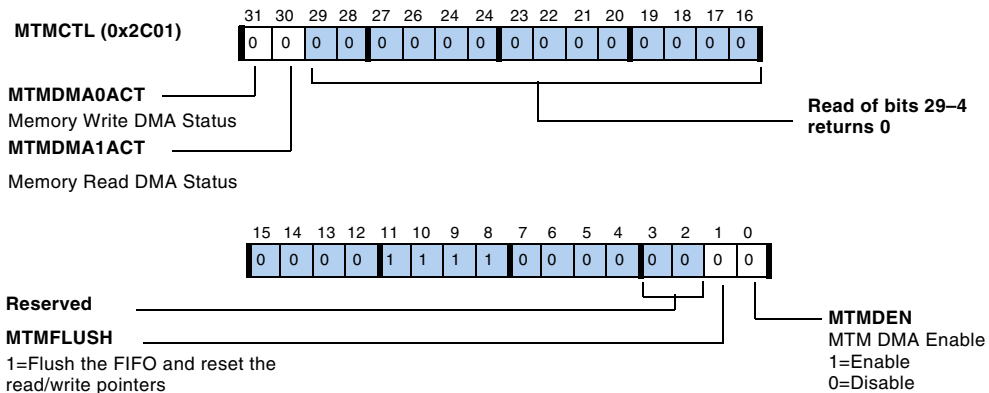


Figure A-12. MTM DMA Register (MTMCTL)

Serial Port Registers

The following section describes serial port (SPORT) registers.

SPORT Serial Control Registers (SPCTLx)

The SPORT serial control registers' addresses are:

SPCTL0 – 0xC00	SPCTL1 – 0xC01
SPCTL2 – 0x400	SPCTL3 – 0x401
SPCTL4 – 0x800	SPCTL5 – 0x801
SPCTL6 – 0x4800	SPCTL7 – 0x4801

The reset value for these registers is 0x0000 0000. The SPCTLx registers are transmit and receive control registers for the corresponding serial ports (SPORT 0 through 7). The following figures show the bit descriptions and settings for the SPORT operating modes.

- [Figure A-13](#) and [Figure A-14](#) provide bit definitions for standard DSP serial mode.
- [Figure A-15](#) provides bit definitions in left-justified sample-pair and I²S mode.
- [Figure A-16](#), [Figure A-17](#) and [Figure A-18](#) provide bit definitions for packed I²S and multichannel mode.



When changing SPORT operating modes, programs should clear the serial port's control register before writing the new settings to the control register. See [Table A-7](#) for a complete description of SPORT operation modes and [Table A-8](#) for complete bit descriptions.

Serial Port Registers

Table A-7. SPORT Operation Modes

Operating Modes	Bits					
	OPMODE	LAFS	FRFS	MCEA	MCEB	SLEN _x
Standard DSP Serial Mode	0	0, 1	X	0	0	3-32 ¹
I ² S (Tx/Rx on Left Channel First)	1	0	1	0	0	8-32
I ² S (Tx/Rx on Right Channel First)	1	0	0	0	0	8-32
Packed I ² S Mode A Channel	1	0	X	1	0	3-32
Packed I ² S Mode B Channel	1	0	X	0	1	3-32
Packed I ² S Mode A and B Channels	1	0	X	1	1	3-32
Left-Justified Sample Pair Mode (Tx/Rx on FS Rising Edge)	1	1	0	0	0	8-32
Left-Justified Sample Pair (Tx/Rx on FS Falling Edge)	1	1	1	0	0	8-32
Multichannel A Channels	0	0	X	1	0	3-32 ¹
Multichannel B Channels	0	0	X	0	1	3-32 ¹
Multichannel A and B Channels	0	0	X	1	1	3-32 ¹

- ¹ Although serial ports process word lengths of 3 to 32 bits, transmitting or receiving words smaller than 7 bits at core clock frequency/4 of the processor may cause incorrect operation when DMA chaining is enabled. Chaining disables the processor's internal I/O bus for several cycles while the new transfer control block (TCB) parameters are being loaded. Receive data may be lost (for example, overwritten) during this period.

SPCTL0 (0xC00) SPCTL1 (0xC01)
 SPCTL2 (0x400) SPCTL3 (0x401)
 SPCTL4 (0x800) SPCTL5 (0x800)
 SPCTL6 (0x4800) SPCTL7 (0x4801)

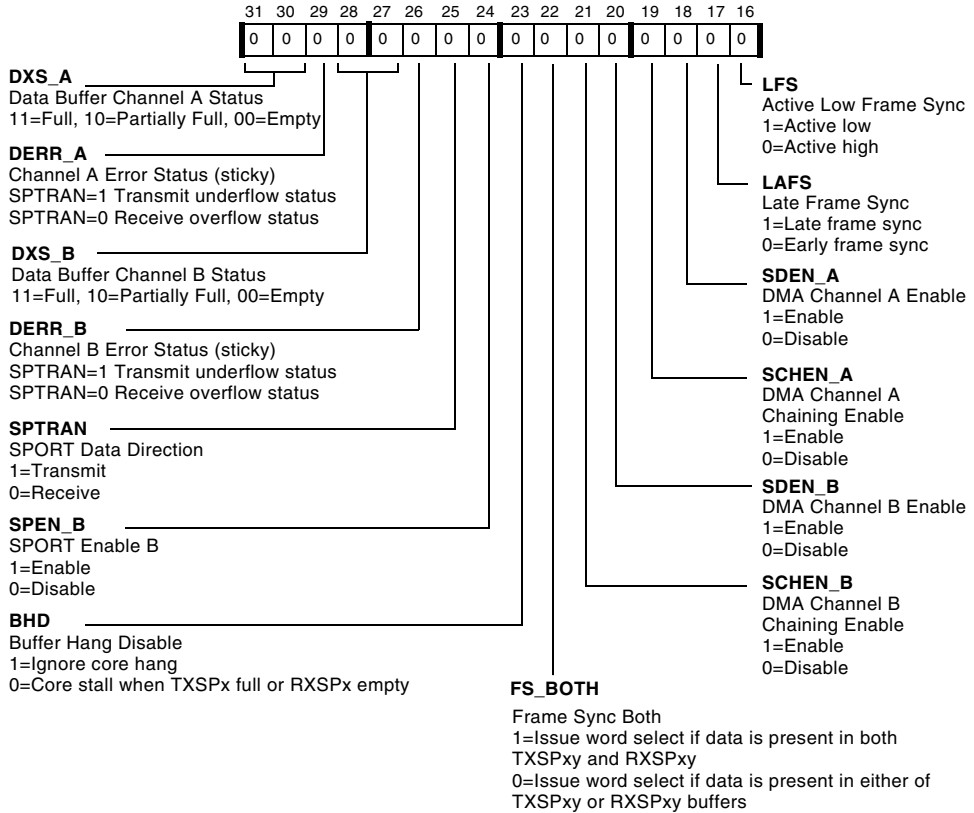


Figure A-13. SPCTLx Register (Bits 16–31) for Standard DSP Serial Mode

Serial Port Registers

SPCTL0 (0xC00) SPCTL1 (0xC01)
SPCTL2 (0x400) SPCTL3 (0x401)
SPCTL4 (0x800) SPCTL5 (0x801)
SPCTL6 (0x4800) SPCTL7 (0x4801)

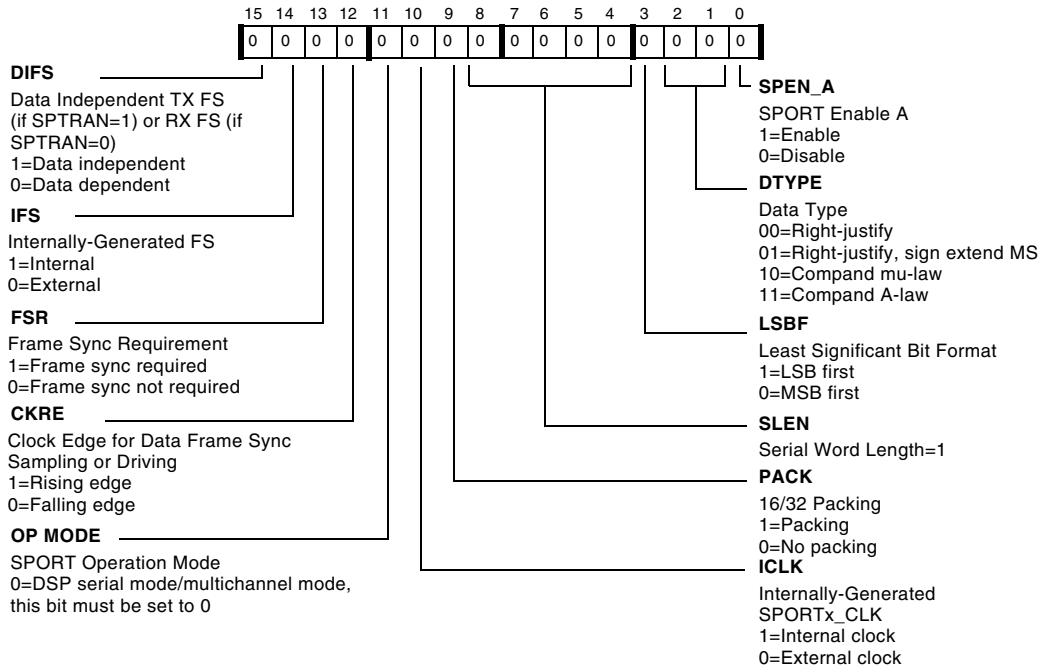


Figure A-14. SPCTLx Register (Bits 0–15) for Standard DSP Serial Mode

SPCTL0 (0xC00) SPCTL1 (0xC01)
 SPCTL2 (0x400) SPCTL3 (0x401)
 SPCTL4 (0x800) SPCTL5 (0x801)
 SPCTL6 (0x4800) SPCTL7 (0x4801)

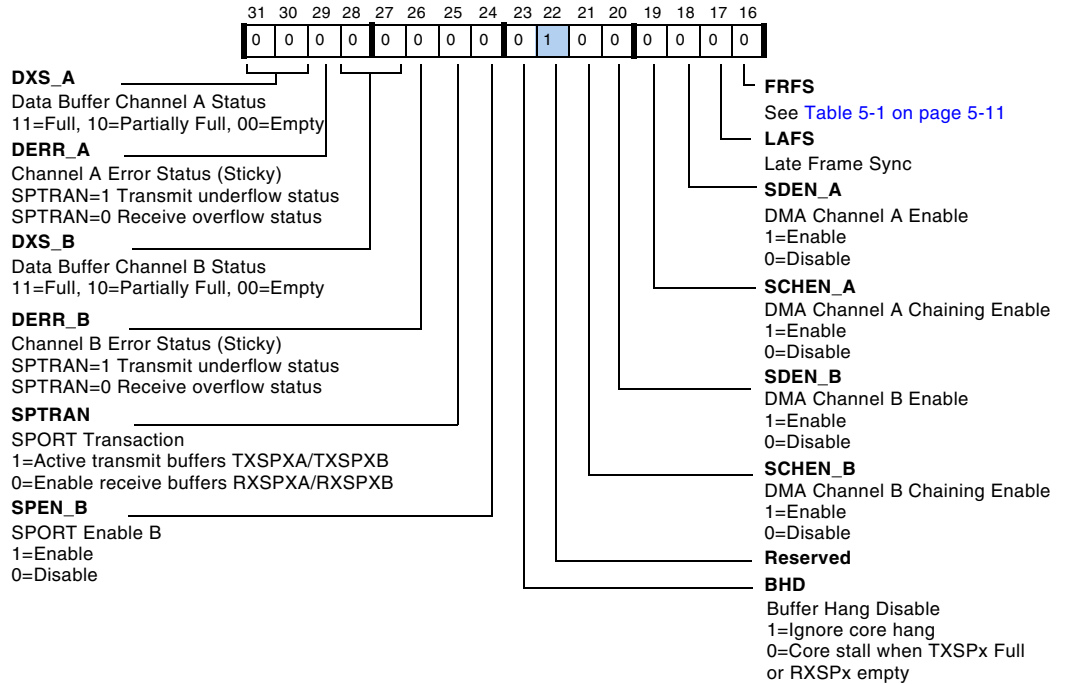


Figure A-15. SPCTLx Register (Bits 31–16) for I²S and Left-Justified Sample Pair Modes

Serial Port Registers

SPCTL0 (0xC00) SPCTL1 (0xC01)
 SPCTL2 (0x400) SPCTL3 (0x401)
 SPCTL4 (0x800) SPCTL5 (0x801)
 SPCTL6 (0x4800) SPCTL7 (0x4801)

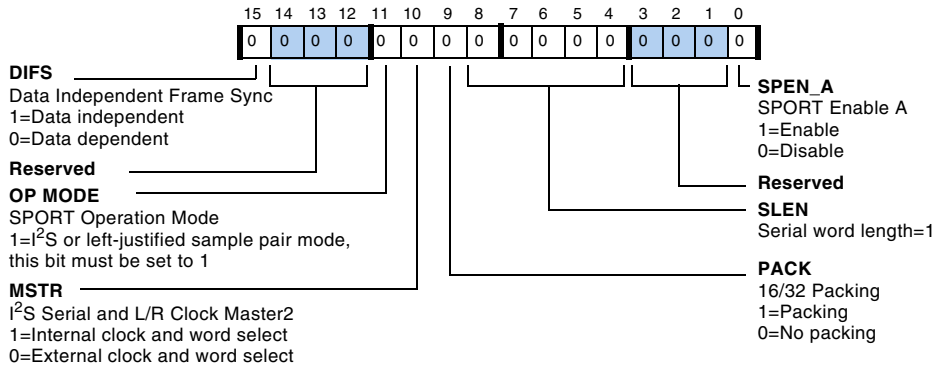


Figure A-16. SPCTLx Register (Bits 15 – 0) for I²S and Left-Justified Sample Pair Modes

SPCTL0 (0xC00) SPCTL1 (0xC01)
 SPCTL2 (0x400) SPCTL3 (0x401)
 SPCTL4 (0x800) SPCTL5 (0x801)
 SPCTL6 (0x4800) SPCTL7 (0x4801)

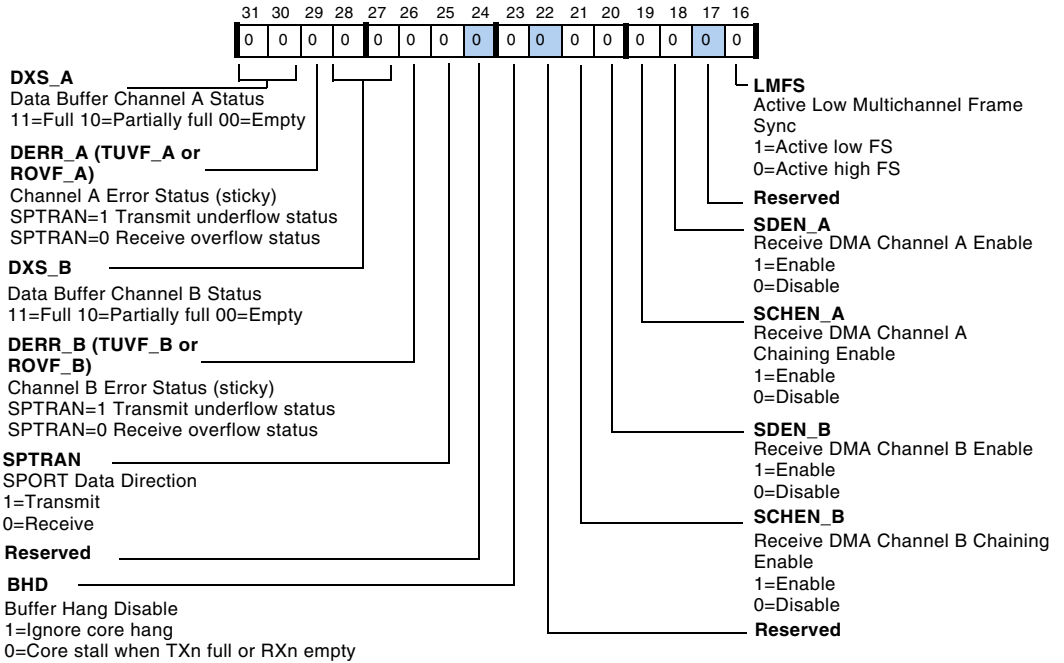


Figure A-17. SPCTLx Register (Bits 31–16) for Packed I²S and Multichannel Mode

Serial Port Registers

SPCTL0 (0xC00) SPCTL1 (0xC01)
 SPCTL2 (0x400) SPCTL3 (0x401)
 SPCTL4 (0x800) SPCTL5 (0x801)
 SPCTL6 (0x4800) SPCTL7 (0x4801)

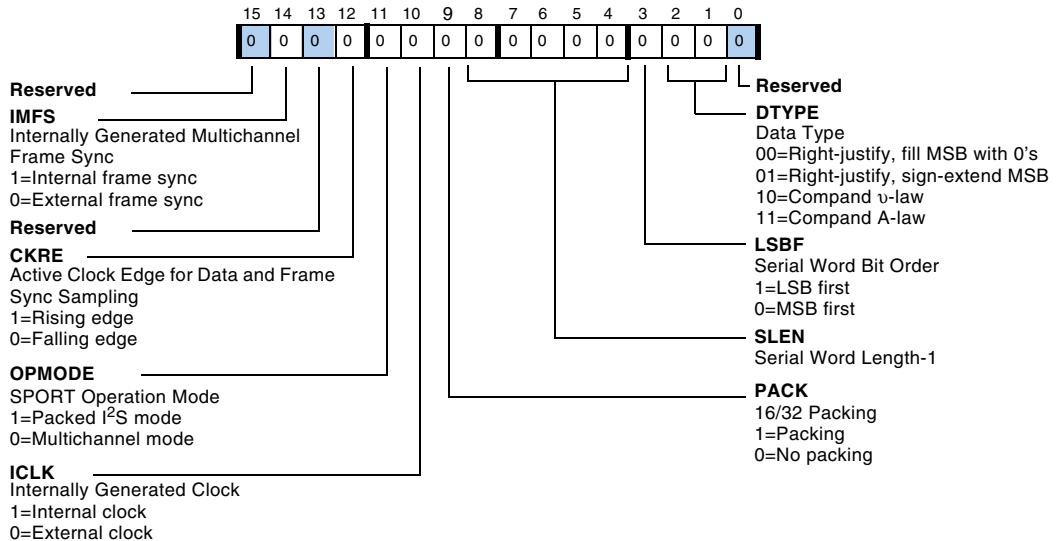


Figure A-18. SPCTLx Register (Bits 15–0) for Packed I²S and Multichannel Mode

Table A-8. SPCTLx Register Bit Descriptions

Bit	Name	Description															
0	SPEN_A	Enable Channel A Serial Port. 0 = Serial port A channel disabled 1 = Serial port A channel enabled This bit is reserved when the SPORT is in packed I ² S and multichannel modes.															
2–1	DTYPE	Data Type Select. Selects the data type formatting for normal and multichannel transmissions as follows: <table> <tr> <th>Normal</th><th>Multichannel</th><th>Data Type Formatting</th></tr> <tr> <td>00</td><td>x0</td><td>Right-justify, zero-fill unused MSBs</td></tr> <tr> <td>01</td><td>x1</td><td>Right-justify, sign-extend unused MSBs</td></tr> <tr> <td>10</td><td>0x</td><td>Compand using μ-law</td></tr> <tr> <td>11</td><td>1x</td><td>Compand using A-law</td></tr> </table>	Normal	Multichannel	Data Type Formatting	00	x0	Right-justify, zero-fill unused MSBs	01	x1	Right-justify, sign-extend unused MSBs	10	0x	Compand using μ -law	11	1x	Compand using A-law
Normal	Multichannel	Data Type Formatting															
00	x0	Right-justify, zero-fill unused MSBs															
01	x1	Right-justify, sign-extend unused MSBs															
10	0x	Compand using μ -law															
11	1x	Compand using A-law															
3	LSBF	Serial Word Endian Select. 0 = Big endian (MSB first) 1 = Little endian (LSB first) This bit is reserved when the SPORT is in I ² S or left-justified sample pair modes.															
8–4	SLEN	Serial Word Length Select. Selects the word length in bits. For DSP serial and multichannel modes, word sizes can be from 3 bits (SLEN = 2) to 32 bits (SLEN = 31). For I ² S and left-justified modes, word sizes can be from 8 bits (SLEN = 7) to 32 bits (SLEN = 31).															
9	PACK	16-Bit to 32-Bit Word Packing Enable. 0 = Disable 16- to 32-bit word packing 1 = Enable 16- to 32-bit word packing															
10	ICLK MSTR (I ² S mode only)	Internal Clock Select. 0 = Select external transmit clock 1 = Select the internal transmit clock This bit applies to DSP serial and multichannel modes, including packed I ² S modes. In I ² S and left-justified sample pair mode, this bit selects the word source and internal clock (if set, = 1) or external clock (if cleared, = 0)															
11	OPMODE	Sport Operation Mode. 0 = DSP serial/multichannel mode if cleared 1 = Selects the I ² S, packed I ² S, left-justified sample pair mode															

Serial Port Registers

Table A-8. SPCTLx Register Bit Descriptions (Cont'd)

Bit	Name	Description
12	CKRE	Clock Rising Edge Select. Determines clock signal to sample data and the frame sync selection. 0 = Falling edge 1 = Rising edge CKRE is reserved when the SPORT is in I ² S and left-justified sample pair modes.
13	FSR	Frame Sync Required Select. Selects whether the serial port requires (if set, = 1) or does not require a transfer frame sync (if cleared, = 0). FSR is reserved when the SPORT is in packed I ² S, I ² S, left-justified sample pair, and multichannel modes. See Table A-7 on page A-30 .
14	IFS (IMFS)	Internal Frame Sync Select. Selects whether the serial port uses an internally-generated frame sync (if set, = 1) or uses an external frame sync (if cleared, = 0). This bit is reserved when the SPORT is in I ² S mode, left-justified sample pair mode. See Table A-7 on page A-30 .
15	DIFS	Data Independent Frame Sync Select. 1 = Serial port uses a data-independent frame sync (sync at selected interval) 0 = Serial port uses a data-dependent frame sync (sync when TX FIFO is not empty or when RX FIFO is not full). This bit is reserved when the SPORT is in packed I ² S and multichannel modes. See Table A-7 on page A-30 .
16	LFS, FRFS, LMFS	Active Low Frame Sync Select. Depending on the OPMODE, selects an active high or low, left or right channel frame sync. See Table A-7 on page A-30 .
17	LAFS	Late Transmit Frame Sync Select. 0 = Early frame sync (FS before first bit). 1 = Late frame sync (FS during first bit) This bit is reserved when the SPORT is in packed I ² S and multichannel modes. See Table A-7 on page A-30 .
18	SDEN_A	Enable Channel A Serial Port DMA. 0 = Disable serial port channel A DMA 1 = Enable serial port channel A DMA
19	SCHEN_A	Enable Channel A Serial Port DMA Chaining. 0 = Disable serial port channel A DMA chaining 1 = Enable serial port channel A DMA chaining

Table A-8. SPCTLx Register Bit Descriptions (Cont'd)

Bit	Name	Description
20	SDEN_B	Enable Channel B Serial Port DMA. 0 = Disable serial port channel B DMA 1 = Enable serial port channel B DMA
21	SCHEN_B	Enable Channel B Serial Port DMA Chaining. 0 = Disable serial port channel B DMA chaining 1 = Enable serial port channel B DMA chaining
22	FS_BOTH	FS Both Enable. 0 = Issue WS if data is present in either transmit buffer 1 = Issue WS if data is present in <i>both</i> transmit buffers This bit is reserved when the SPORT is in packed I ² S, multichannel, I ² S and left-justified sample pair modes.
23	BHD	Buffer Hang Disable. 0 = Indicates a core stall. The core stalls when it tries to write to a full transmit buffer or read an empty receive buffer FIFO. 1 = Ignore a core hang This bit applies to all modes.
24	SPEN_B	Enable Channel B Serial Port. 0 = Serial port A channel disabled 1 = Serial port A channel enabled This bit is reserved when the SPORT is in packed I ² S and multichannel modes.
25	SPTRAN	Data Direction Control. 0 = Enable receive buffers 1 = Activate transmit buffers
26	DERR_B (TUVF_B or ROVF_B)	Channel B Error Status (sticky, read-only). Indicates if the serial transmit operation has underflowed or a receive operation has overflowed in the channel B data buffer.
28–27	DXS_B	Channel B Data Buffer Status (read-only). Indicates the status of the serial port's channel B data buffer as follows: 11 = Full 00 = Empty 10 = Partially full This bit is reserved when the SPORT is in multichannel mode.

Serial Port Registers

Table A-8. SPCTLx Register Bit Descriptions (Cont'd)

Bit	Name	Description
29	DERR_A (TUVF_A or ROVF_A)	Channel A Error Status (sticky, read-only). Indicates if the serial transmit operation has underflowed or a receive operation has overflowed in the channel A data buffer.
31–30	DXS_A	Channel A Data Buffer Status (read-only). Indicates the status of the 11 = Full 00 = Empty 10 = Partially full

SPORT Multichannel Control Registers (SPMCTLx)

Unlike previous SHARC designs, the serial ports in the ADSP-21367/8/9 and ADSP-2137x processors work individually, not in pairs. Therefore, each SPORT has its own multichannel control register. These registers are shown in [Figure A-19](#) and described in [Table A-9](#). The reset value for these registers is undefined and their addresses are:

SPMCTL0 – 0xC04

SPMCTL1 – 0xC17

SPMCTL2 – 0x404

SPMCTL3 – 0x417

SPMCTL4 – 0x804

SPMCTL5 – 0x817

SPMCTL6 – 0x4804

SPMCTL7 – 0x4817

SPMCTL0 (0xC04) SPMCTL1 (0xC17)
 SPMCTL2 (0x404) SPMCTL3 (0x417)
 SPMCTL4 (0x804) SPMCTL5 (0x817)
 SPMCTL6 (0x4804) SPMCTL7 (0x4817)

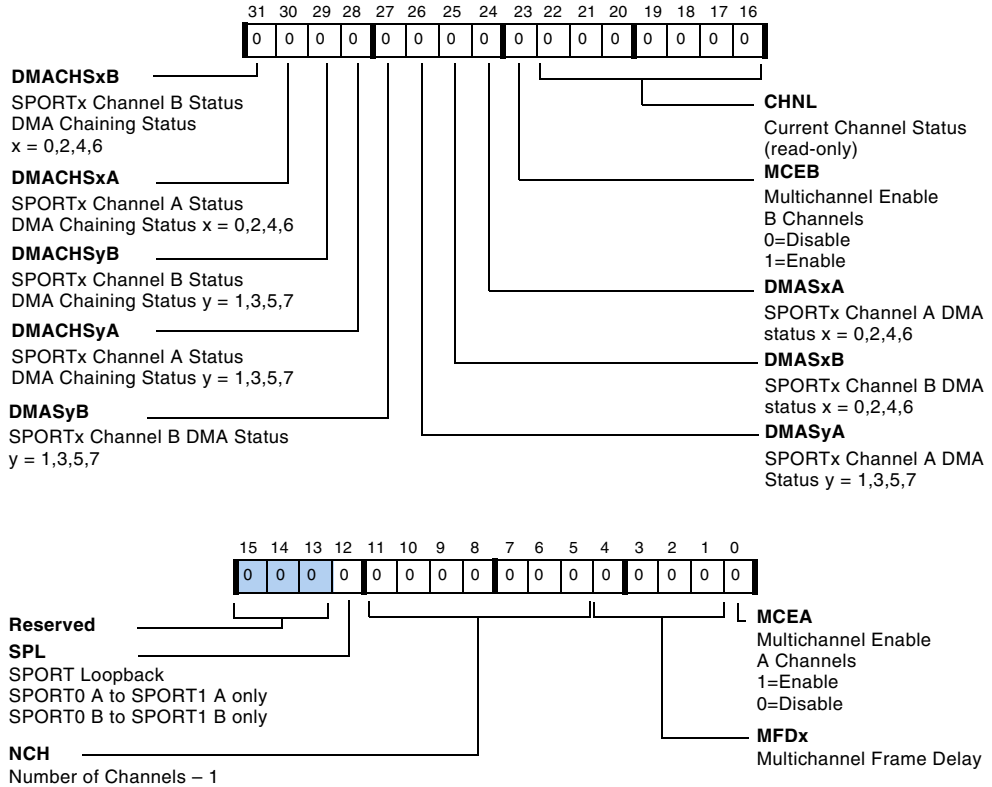


Figure A-19. SPMCTLx Registers – Multichannel Mode

Serial Port Registers

Table A-9. SPMCTLx Register Bit Descriptions

Bit	Name	Description
0	MCEA	Multichannel Mode Enable. Standard and multichannel modes only. One of two configuration bits that enable and disable multichannel mode on serial port channels. See OPMODE bit on page A-37 . 0 = Disable multichannel operation 1 = Enable multichannel operation if OPMODE = 0
4–1	MFD	Multichannel Frame Delay. Set the interval, in number of serial clock cycles, between the multichannel frame sync pulse and the first data bit. These bits provide support for different types of T1 interface devices. Valid values range from 0 to 15 with bits [4:1]. Values of 1 to 15 correspond to the number of intervening serial clock cycles. A value of 0 corresponds to no delay. The multichannel frame sync pulse is concurrent with first data bit.
11–5	NCH	Number of Multichannel Slots (minus one). Select the number of channel slots (maximum of 128) to use for multichannel operation. Valid values for actual number of channel slots range from 1 to 128. Use this formula to calculate the value for NCH: $NCH = \text{Actual number of channel slots} - 1$.
12	SPL	SPORT Loopback Mode. Enables if set (= 1) or disables if cleared (= 0) the channel loopback mode. Loopback mode enables developers to run internal tests and to debug applications. Loopback works under the following SPORT configurations where either of the two paired SPORTs can be set up to transmit or receive, depending on their SPTRAN bit setting. SPORT0 and SPORT1. SPORT0 can only be paired with SPORT1, controlled by the SPL bit in the SPMCTL0 register. SPORT2 and SPORT3. SPORT2 can only be paired with SPORT3, controlled by the SPL bit in the SPMCTL2 register. SPORT4 and SPORT5. SPORT4 can only be paired with SPORT5, controlled by the SPL bit in the SPMCTL4 register. SPORT6 and SPORT7. SPORT6 can only be paired with SPORT7, controlled via SPL bit in the SPMCTL6 register
15–13	Reserved	

Table A-9. SPMCTLx Register Bit Descriptions (Cont'd)

Bit	Name	Description
22–16	CHNL	Current Channel Selected (read-only, sticky). Identify the currently selected transmit channel slot (0 to 127).
23	MCEB	Multichannel Enable, B Channels. 0 = Disable 1 = Enable
27–24	DMASxy	DMA Status. Selects the transfer status. 0 = Inactive 1 = Active (read-only)
31–28	DMACHSxy	DMA Chaining Status. 0 = Inactive 1 = Active (read-only)

SPORT Transmit Buffer Registers (TXSPx)

The addresses of the TXSPx registers are:

TXSP0A – 0xC60	TXSP0B – 0xC62
TXSP1A – 0xC64	TXSP1B – 0xC66
TXSP2A – 0x460	TXSP2B – 0x462
TXSP3A – 0x464	TXSP3B – 0x466
TXSP4A – 0x860	TXSP4B – 0x862
TXSP5A – 0x864	TXSP5B – 0x866
TXSP6A – 0x4860	TXSP6B – 0x4862
TXSP7A – 0x4864	TXSP7B – 0x4866

The 32-bit TXSPx registers hold the output data for serial port transmit operations. The reset value for these registers is undefined. For more information on how transmit buffers work, see [“Transmit and Receive Data Buffers \(TXSPxA/B, RXSPxA/B\)”](#) on page 5-67.

SPORT Receive Buffer Registers (RXSPx)

The 32-bit `RXSPx` registers hold the input data from serial port receive operations. The reset value for these registers is undefined. For more information on how receive buffers work, see [“Transmit and Receive Data Buffers \(TXSPx A/B, RXSPx A/B\)” on page 5-67](#). The addresses of the `RXSPx` registers are:

<code>RXSP0A – 0xc61</code>	<code>RXSP0B – 0xc63</code>
<code>RXSP1A – 0xc65</code>	<code>RXSP1B – 0xc67</code>
<code>RXSP2A – 0x461</code>	<code>RXSP2B – 0x463</code>
<code>RXSP3A – 0x465</code>	<code>RXSP3B – 0x467</code>
<code>RXSP4A – 0x861</code>	<code>RXSP4B – 0x863</code>
<code>RXSP5A – 0x865</code>	<code>RXSP5B – 0x867</code>
<code>RXSP6A – 0x4861</code>	<code>RXSP6B – 0x4863</code>
<code>RXSP7A – 0x4865</code>	<code>RXSP7B – 0x4867</code>

SPORT Divisor Registers (DIVx)

The addresses of the `DIVx` registers are:

<code>DIV0 – 0xc02</code>	<code>DIV1 – 0xc03</code>
<code>DIV2 – 0x402</code>	<code>DIV3 – 0x403</code>
<code>DIV4 – 0x802</code>	<code>DIV5 – 0x803</code>
<code>DIV6 – 0x4802</code>	<code>DIV7 – 0x4803</code>

These registers, shown in [Figure A-20](#), have an undefined reset value. These registers contain two fields:

- Bits 15–1 are `CLKDIV`. These bits identify the serial clock divisor value for internally-generated `SCLK` as follows:

$$CLKDIV = \frac{f_{CLK}}{8(f_{SCLK})} - 1$$

- Bits 31–16 are $FSDIV$. These bits select the frame sync divisor for internally-generated TFS as follows:

$$FSDIV = \frac{f_{SCLK}}{f_{SFS}} - 1$$

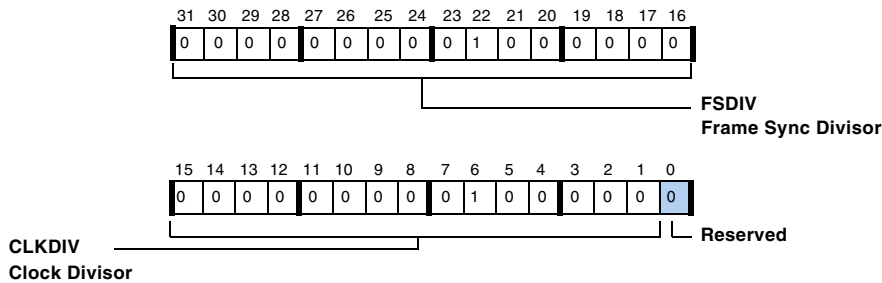


Figure A-20. $DIVx$ Register

SPORT Count Registers (SPCNTx)

The $SPCNTx$ registers provides status information for the internal clock and frame sync. The addresses of the $SPCNTx$ registers are:

$SPCNT0 - 0xC15$	$SPCNT1 - 0xC16$
$SPCNT2 - 0x415$	$SPCNT3 - 0x416$
$SPCNT4 - 0x815$	$SPCNT5 - 0x816$
$SPCNT6 - 0x4815$	$SPCNT7 - 0x4816$

The reset value for these registers is undefined.

SPORT Active Channel Select Registers (SPxCSy)

Each bit, 31–0, set (= 1) in one of the four SPxCSy registers corresponds to the active channel, 127–0, on a multichannel mode serial port. When the SPxCSy registers activate a channel, the serial port transmits or receives the word in that channel's position of the data stream. When a channel's bit in the SPxCSy register is cleared (= 0), the serial port's data transmit pin three-states during the channel's transmit time slot if the serial port is configured as transmitter. If the serial port is configured as the receiver it ignores the incoming data. The addresses of the SPxCSy registers are:

SP0CS0 – 0xC05	SP0CS1 – 0xC06
SP0CS2 – 0xC07	SP0CS3 – 0xC08
SP2CS0 – 0x405	SP2CS1 – 0x406
SP2CS2 – 0x407	SP2CS3 – 0x408
SP4CS0 – 0x805	SP4CS1 – 0x806
SP4CS2 – 0x807	SP4CS3 – 0x808
SP6CS0 – 0x4805	SP6CS1 – 0x4806
SP6CS2 – 0x4807	SP6CS3 – 0x4808

The reset value for these registers is undefined.

SPORT Compand Registers (SPxCCSy)

Each bit, 31–0, set (= 1) in one of the four SPxCCSy registers corresponds to a companded channel, 127–0, on a multichannel mode serial port. When the SPxCCSy register activates companding for a channel, the serial port applies the companding from the serial port's DTYPE selection to the word transmitted or received in that channel's position of the data stream. When a channel's bit in the SPxCCSy register is cleared (= 0), the serial port does not compand the outgoing or incoming data during the channel's time slot.

The addresses and channel assignments of the SPxCCSy registers are shown below.

SP0CCS0 – 0xC0D, SPORT0 channels 31 - 0	SP0CCS1 – 0xC0E, SPORT0 channels 63 - 32
SP0CCS2 – 0xC0F, SPORT0, channels 95 - 64	SP0CCS3 – 0xC10, SPORT0 channels 127 - 96
SP1CCS0 – 0xC11, SPORT1 channels 31 - 0	SP1CCS1 – 0xC12, SPORT1 channels 63 - 32
SP1CCS2 – 0xC13, SPORT1, channels 95 - 64	SP1CCS3 – 0xC14, SPORT1 channels 127 - 96
SP2CCS0 – 0x40D, SPORT2 channels 31 - 0	SP2CCS1 – 0x40E, SPORT2 channels 63 - 32
SP2CCS2 – 0x40F, SPORT2, channels 95 - 64	SP2CCS3 – 0x410, SPORT2 channels 127 - 96
SP3CCS0 – 0x411, SPORT3 channels 31 - 0	SP3CCS1 – 0x412, SPORT3 channels 63 - 32
SP3CCS2 – 0x413, SPORT3, channels 95 - 64	SP3CCS3 – 0x414, SPORT3 channels 127 - 96
SP4CCS0 – 0x80D, SPORT4 channels 31 - 0	SP4CCS1 – 0x80E, SPORT4 channels 63 - 32
SP4CCS2 – 0x80F, SPORT4, channels 95 - 64	SP4CCS3 – 0x810, SPORT4 channels 127 - 96
SP5CCS0 – 0x811, SPORT5 channels 31 - 0	SP5CCS1 – 0x812, SPORT5 channels 63 - 32
SP5CCS2 – 0x813, SPORT5, channels 95 - 64	SP5CCS3 – 0x814, SPORT5 channels 127 - 96
SP6CCS0 – 0x480D, SPORT6 channels 31 - 0	SP6CCS1 – 0x480E, SPORT6 channels 63 - 32
SP6CCS2 – 0x480F, SPORT6, channels 95 - 64	SP6CCS3 – 0x4810, SPORT6 channels 127 - 96
SP7CCS0 – 0x4811, SPORT7 channels 31 - 0	SP7CCS1 – 0x4812, SPORT7 channels 63 - 32
SP7CCS2 – 0x4813, SPORT7, channels 95 - 64	SP7CCS3 – 0x4814, SPORT7 channels 127 - 96

The reset value for these registers is undefined.

SPORT Error Control Register (SPERRCTLx)

The SPERRCTLx registers control and report the status of the interrupts generated by each SPORT (see [Figure A-21](#)).

SPERRCTL0 (0xC18), SPERRCTL1 (0xC19)
 SPERRCTL2 (0x418), SPERRCTL3 (0x419)
 SPERRCTL4 (0x818), SPERRCTL5 (0x819)
 SPERRCTL6 (0x4818), SPERRCTL7 (0x4819)

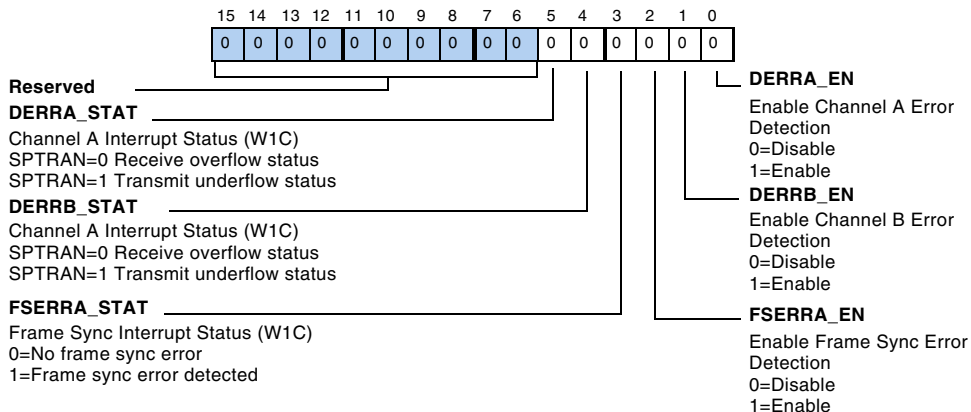


Figure A-21. SPERRCTLx Register

SPORT Error Status Register (SPERRSTAT)

In the ADSP-21367/8/9 and ADSP-2137x processors, there is one global interrupt status register, SPERRSTAT, that checks the status of SPORT interrupts. This read-only register is located at address 0x2300 and is 24 bits wide (see [Figure A-22](#)).

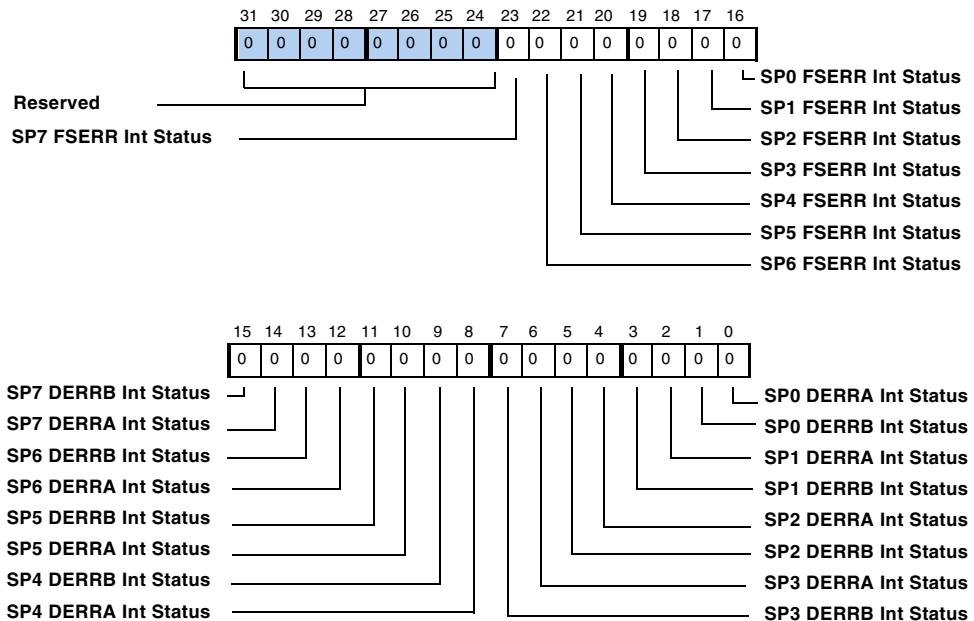


Figure A-22. SPERRSTATx Register

SPORT DMA Index Registers (IISP_x)

The IISP_x register is 19 bits wide. It holds an address and acts as a pointer to memory for a DMA transfer. [For more information, see “I/O Processor” on page 2-1.](#) The addresses of the IISP_x registers are:

IISP0A – 0xC40	IISP0B – 0xC44
IISP1A – 0xC48	IISP1B – 0xC4C
IISP2A – 0x440	IISP2B – 0x444
IISP3A – 0x448	IISP3B – 0x44C
IISP4A – 0x840	IISP4B – 0x844
IISP5A – 0x848	IISP5B – 0x84C
IISP6A – 0x4840	IISP6B – 0x4844
IISP7A – 0x4848	IISP7B – 0x484C

The reset value for these registers is undefined.

SPORT DMA Modifier Registers (IMSP_x)

The IMSP_x register is 16 bits wide and it provides the increment or step size by which an IISP_x register is post-modified during a DMA operation. [For more information, see “I/O Processor” on page 2-1.](#) The reset value for these registers is undefined. The addresses of the IMSP_x registers are:

IMSP0A – 0xC41	IMSP0B – 0xC45
IMSP1A – 0xC49	IMSP1B – 0xC4D
IMSP2A – 0x441	IMSP2B – 0x445
IMSP3A – 0x449	IMSP3B – 0x44D
IMSP4A – 0x841	IMSP4B – 0x845
IMSP5A – 0x849	IMSP5B – 0x84D
IMSP6A – 0x4841	IMSP6B – 0x4845
IMSP7A – 0x4849	IMSP7B – 0x484D

SPORT DMA Count Registers (CSPx)

The CSPx registers are 16 bits wide and they hold the word count for a DMA transfer. [For more information, see “I/O Processor” on page 2-1](#). The reset value for these registers is undefined. The addresses of the CSPx registers are:

CSP0A – 0xC42	CSP0B – 0xC46
CSP1A – 0xC4A	CSP1B – 0xC4E
CSP2A – 0x442	CSP2B – 0x446
CSP3A – 0x44A	CSP3B – 0x44E
CSP4A – 0x842	CSP4B – 0x846
CSP5A – 0x84A	CSP5B – 0x84E
CSP6A – 0x4842	CSP6B – 0x4846
CSP7A – 0x484A	CSP7B – 0x484E

SPORT Chain Pointer Registers (CPSPx)

The CPSPx registers are 20 bits wide. They hold the address for the next transfer control block in a chained DMA operation. [For more information, see “I/O Processor” on page 2-1](#). The reset value for these registers is undefined. The addresses of the CPSPx registers are:

CPSP0A – 0xC43	CPSP0B – 0xC47
CPSP1A – 0xC4B	CPSP1B – 0xC4F
CPSP2A – 0x443	CPSP2B – 0x447
CPSP3A – 0x44B	CPSP3B – 0x44F
CPSP4A – 0x843	CPSP4B – 0x847
CPSP5A – 0x84B	CPSP5B – 0x84F
CPSP6A – 0x4843	CPSP6B – 0x4847
CPSP7A – 0x484B	CPSP7B – 0x484F

Serial Peripheral Interface Registers

The following sections describe the registers associated with the two serial peripheral interfaces (SPIs). The SPI B port is routed through the DAI.

SPI Control Registers (SPICTL, SPICTLB)

The addresses of these registers are 0x1000 and 0x2800 (SPICTLB). The reset value for these registers is 0x0400. The SPI control (SPICTL) registers, shown in [Figure A-23](#) and described in [Table A-10](#), are used to configure and enable the SPI system.

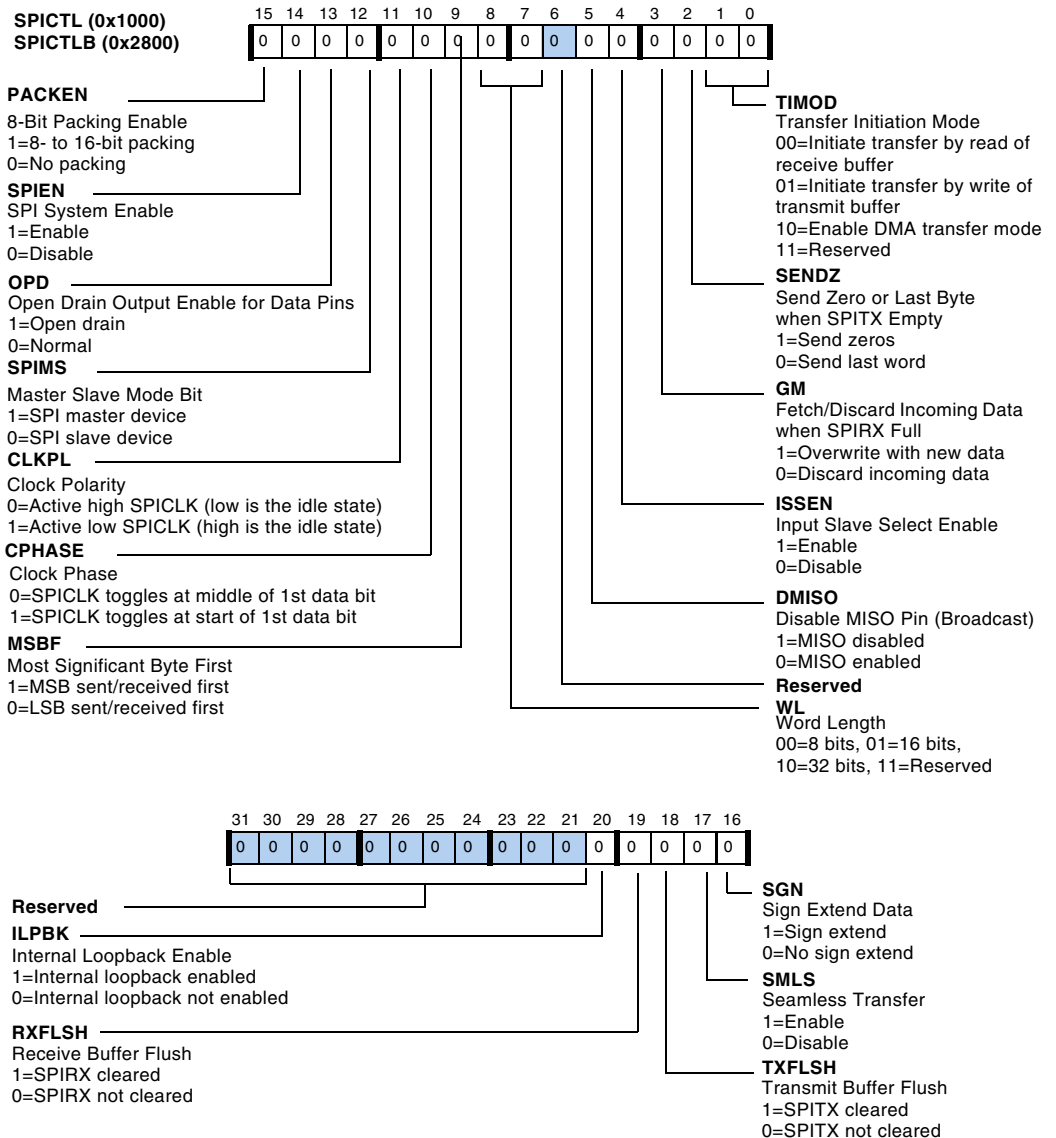


Figure A-23. SPICTL, SPICTLB Registers

Serial Peripheral Interface Registers

Table A-10. SPICTL Register Bit Descriptions

Bit	Name	Description
1–0	TIMOD	Transfer Initiation Mode. Defines transfer initiation mode and interrupt generation. 00 = Initiate transfer by read of receive buffer. Interrupt active when receive buffer is full. 01 = Initiate transfer by write to transmit buffer. Interrupt active when transmit buffer is empty. 10 = Enable DMA transfer mode. Interrupt configured by DMA. 11 = Reserved
2	SENDZ	Send Zero. Send zero or the last word when TXSPI is empty. 0 = Send last word 1 = Send zeros
3	GM	Get Data. When RXSPI is full, get data or discard incoming data 0 = Discard incoming data 1 = Get more data, overwrites the previous data
4	ISSEN	Input Slave-Select Enable. Enables slave-select ($\overline{\text{SPIDS}}$) input for the master. When not used, SPIDS can be disabled, freeing up a chip pin as a general-purpose I/O pin. 0 = Disable 1 = Enable
5	DMISO	Disable MISO Pin. Disables MISO as an output. This is needed in an environment where a master wishes to transmit to various slaves at one time (broadcast). However, only one slave is allowed to transmit data back to the master. This bit should be set for all slaves, except the one from whom the master wishes to receive data. 0 = MISO enabled 1 = MISO disabled
6	Reserved	
8–7	WL	Word Length. 00 = 8 bits 01 = 16 bits 10 = 32 bits
9	MSBF	Most Significant Byte First. 0 = LSB sent/received first 1 = MSB sent/received first

Table A-10. SPICTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
10	CPHASE	Clock Phase. Selects the transfer format. 0 = SPICLK starts toggling at the middle of 1st data bit. 1 = SPICLK starts toggling at the start of 1st data bit.
11	CLKPL	Clock Polarity. 0 = Active high SPICLK (SPICLK low is the idle state) 1 = Active low SPICLK (SPICLK high is the idle state)
12	SPIMS	SPI Master Select. Configures SPI module as master or slave. 0 = Device is a slave device 1 = Device is a master device
13	OPD	Open Drain Output Enable. Enables open drain data output enable (for MOSI and MISO). 0 = Normal 1 = Open drain
14	SPIEN	SPI Port Enable. 0 = SPI module is disabled 1 = SPI module is enabled
15	PACKEN	Packing Enable. 0 = No packing 1 = 8- to16-bit packing Note: This bit may be 1 only when WL = 00 (8-bit transfer). When in transmit mode, the PACKEN bit unpacks data.
16	SGN	Sign Extend. 0 = No sign extension 1 = Sign extension
17	SMLS	Seamless Transfer. 0 = Seamless transfer disabled 1 = Seamless transfer enabled, not supported in mode TIMOD1=0 = 00 and CPHASE=0 for all modes
18	TXFLSH	Flush Transmit Buffer. Write a 1 to this bit to clear TXSPI. 0 = TXSPI not cleared 1 = TXSPI cleared
19	RXFLSH	Clear RXSPI. Write a 1 to this bit to clear RXSPI. 0 = RXSPI not cleared 1 = RXSPI cleared

Serial Peripheral Interface Registers

Table A-10. SPICTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
20	ILPBK	Internal Loop Back. 0 = No internal loopback 1 = Internal loopback enabled
31–21	Reserved	

SPI Port Status (SPISTAT, SPISTATB) Registers

These registers' addresses are 0x1002 and 0x2802 (SPISTATB). The reset value for these registers is 0x01. The SPISTAT and SPISTATB registers, shown in Figure A-24 and described in Table A-11, are read-only registers used to detect when an SPI transfer is complete, if transmission/reception errors occur, and the status of the TXSPI and RXSPI FIFOs. The bits in these registers are W1C-type (write 1-to-clear).

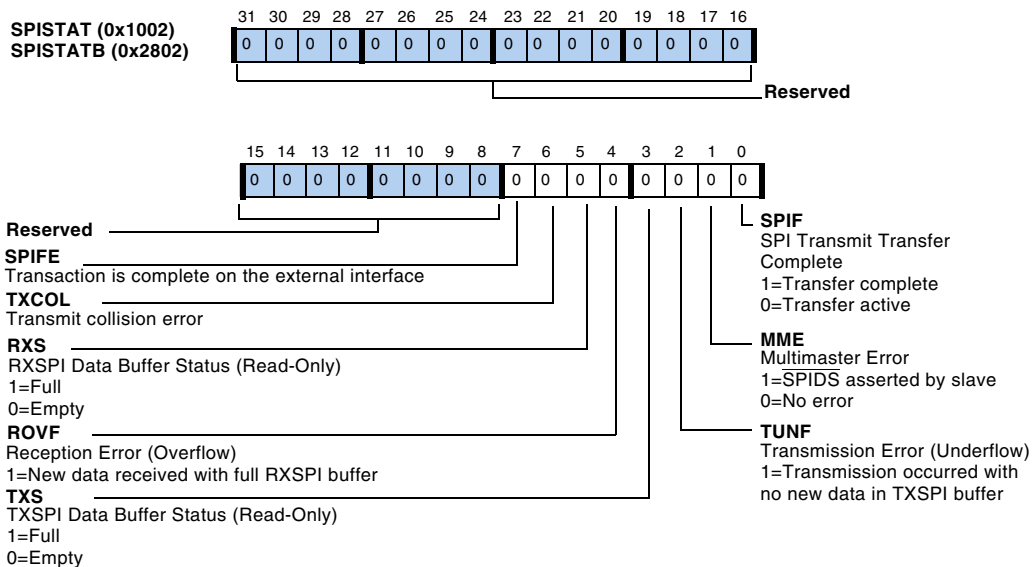


Figure A-24. SPISTAT, SPISTATB Registers

Table A-11. SPISTAT Register Bit Descriptions

Bit	Name	Description
0	SPIF	SPI Transmit or Receive Transfer Complete. SPIF is set when an SPI single-word transfer is complete.
1	MME	Multimaster Error or Mode-Fault Error. MME is set in a master device when some other device tries to become the master.
2	TUNF	Transmission Error. TUNF is set when transmission occurred with no new data in TXSPI register.
3	TXS	Transmit Data Buffer Status. TXSPI data buffer status. 0 = Empty 1 = Full
4	ROVF	Reception Error. ROVF is set when data is received with receive buffer full
5	RXS	Receive Data Buffer Status. 0 = Empty 1 = Full
6	TXCOL	Transmit Collision Error. When TXCOL is set, it is possible that corrupt data was transmitted.
7	SPIFE	External Transaction Complete. Set (= 1) when the SPI transaction is complete on the external interface.
31-8	Reserved	

SPI Port Flags Registers (SPIFLG, SPIFLGB)

These registers' addresses are 0x1001 and 0x2801 (SPIFLGB). The reset value is 0x0F80. The SPIFLG and SPIFLGB registers, shown in [Figure A-25](#) and described in [Table A-12](#), are used to enable individual SPI slave-select lines when the SPI is enabled as a master.

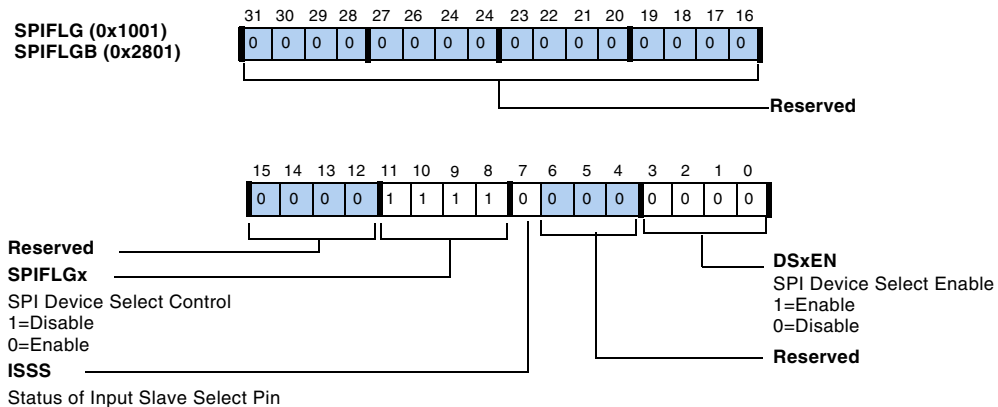


Figure A-25. SPIFLG, SPIFLGB Registers

Table A-12. SPIFLG, SPIFLGB Register Bit Descriptions

Bit	Name	Description
3–0	DSxEN (3-0)	SPI Device Select Enable. Enables or disables the corresponding flag as a flag output to be used for SPI slave-select. 0 = Disable 1 = Enable
6–4	Reserved	
7	ISSS	Input Service Select. This read-only bit reflects the status of the slave-select input pin.
11–8	SPIFLGx (3-0)	SPI Device Select Control. Selects (if cleared, = 0) a corresponding flag output to be used for an SPI slave-select.
12–31	Reserved	

SPI Receive Buffer Registers (RXSPI, RXSPIB)

These registers' addresses are 0x1004 (for RXSPI) and 0x2804 (for RXSPIB). The reset values are undefined. These are 32-bit, read-only registers accessible by the core or DMA controller. At the end of a data transfer, RXSPIx is loaded with the data in the shift register. During a DMA receive operation, the data in RXSPIx is automatically loaded into the internal memory. For core- or interrupt-driven transfers, you can also use the RXS status bits in the SPISTAT register to determine if the receive buffer is full.

RXSPI Shadow Registers (RXSPI_SHADOW, RXSPIB_SHADOW)

These registers' addresses are 0x1006 (for RXSPI_SHADOW) and 0x2806 (for RXSPIB_SHADOW). The reset values are undefined. These registers act as shadow registers for the receive data buffer, RXSPI and RXSPIB registers, and are used to aid software debugging. These registers, are at a different address from RXSPI and RXSPIB but their contents are identical to that of RXSPI and RXSPIB. When a software read of RXSPIx occurs, the RXS bit is cleared and an SPI transfer may be initiated (if TIMOD=00). No such hardware action occurs when the shadow register is read.

SPI Transmit Buffer Registers (TXSPI, TXSPIB)

These registers' addresses are 0x1003 (for TXSPI) and 0x2803 (for TXSPIB). The reset values are undefined. These SPI transmit data registers are 32-bit registers that are part of the IOP register set and can be accessed by the core or the DMA controller. Data is loaded into these registers before being transmitted. Prior to the beginning of a data transfer, data in TXSPIx is automatically loaded into the transmit shift register. During a DMA transmit operation, the data in TXSPIx is automatically loaded from internal memory.

SPI Baud Rate Registers (SPIBAUD, SPIBAUDB)

These registers' addresses are 0x1005 (for SPIBAUD) and 0x2805 (for SPIBAUDB) and their reset values are undefined ([Table A-13](#)). These SPI registers are 16-bit, read-write registers that are used to set the bit transfer rate for a master device. When configured as slaves, the value written to these registers is ignored. The (SPIBAUDx) registers can be read or written at any time. The serial clock rate is determined by the following formula:

$$\text{SPI Baud Rate} = (\text{Peripheral clock rate (PCLK)}) \div (4 \times \text{SPIBAUD}_{15-1})$$

Writing a value of 0 or 1 to these registers disables the serial clock. Therefore, the maximum serial clock rate is one-fourth the peripheral clock rate (PCLK).

Table A-13. SPIBAUD, SPIBAUDB Register Bit Descriptions

Bit	Name	Description
0	Reserved	
15–1	BAUDR	Baud Rate Enable. Enables the SPICLK per the following equation: SPICLK baud rate = core clock (CCLK)/8 x BAUDR Default=0
31–16	Reserved	

Various possible baud rate configurations are shown in [Table A-14](#).

Table A-14. SPI Master Baud Rate Example

BAUDR (Decimal Value)	SPI CLock Divide Factor	Baud Rate for CCLK
0	N/A	N/A
1	8	41.7 MHz
2	16	20.8 MHz

Table A-14. SPI Master Baud Rate Example (Cont'd)

BAUDR (Decimal Value)	SPI CLock Divide Factor	Baud Rate for CCLK
3	24	13.9 MHz
4	32	10.4 MHz
32,767, (0x7FFF)	262136	1.3 KHz

SPI DMA Registers

There are ten SPI DMA-specific registers:

- “SPI DMA Configuration Registers (SPIDMAC, SPIDMACB)” on page A-62
- “SPI DMA Start Address Registers (IISPI, ISPIB)” on page A-64
- “SPI DMA Address Modify Registers (IMSPI, IMSPIB)” on page A-64
- “SPI DMA Word Count Registers (CSPI, CSPIB)” on page A-64
- “SPI DMA Chain Pointer Registers (CPSPI, CPSPIB)” on page A-65

For information on configuring DMA using the SPI, see “Setting Up and Starting Chained DMA over the SPI” on page 2-42.

SPI DMA Configuration Registers (SPIDMAC, SPIDMACB)

These registers addresses are 0x1084 (for SPIDMAC) and 0x2884 (for SPIDMACB) and their reset value is undefined. These 17-bit SPI registers, shown in [Figure A-26](#) and described in [Table A-15](#), are used to control DMA transfers.

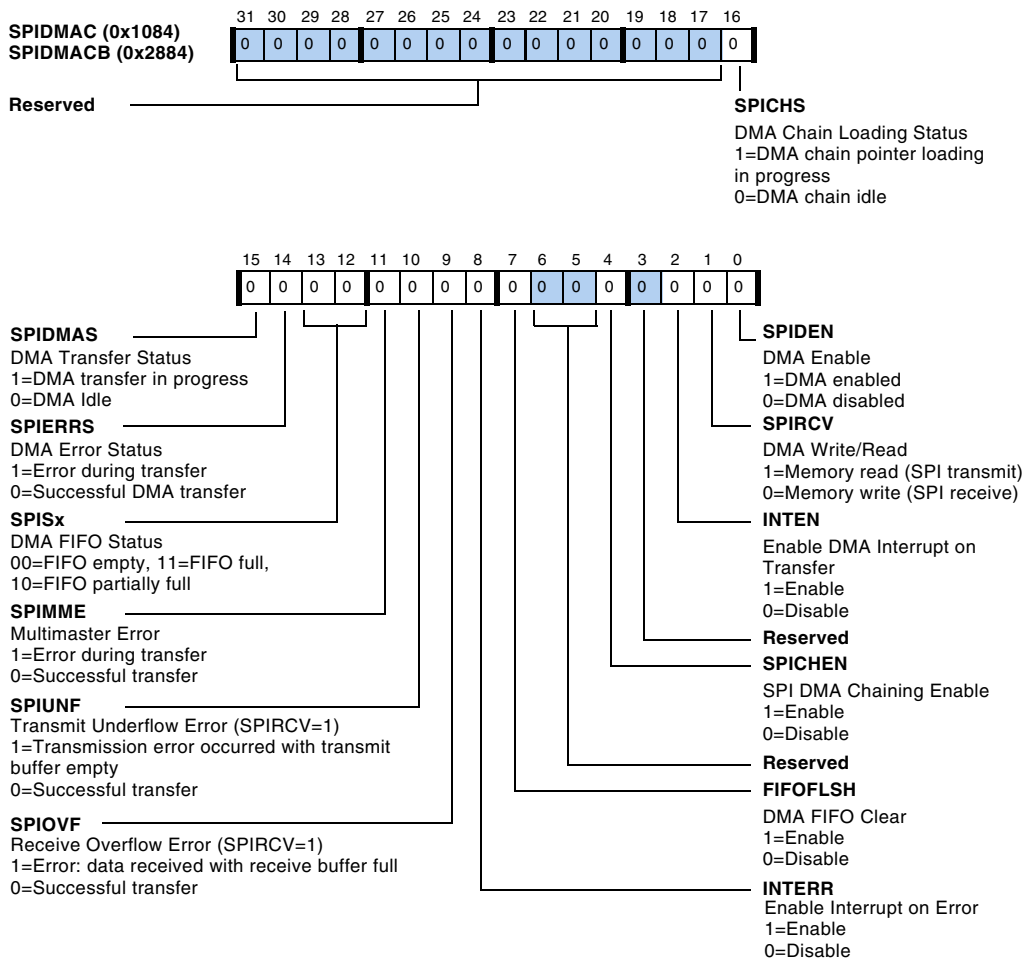


Figure A-26. SPIDMAC, SPIDMACB Registers

Table A-15. SPIDMAC, SPIDMACB Register Bit Descriptions

Bit	Name	Description
0	SPIDEN	DMA Enable. 0 = Disable 1 = Enable
1	SPIRCV	DMA Write/Read. 0 = Memory write (SPI transmit) 1 = Memory read (SPI receive)
2	INTEN	Enable DMA Interrupt on Transfer. 0 = Disable 1 = Enable
3	Reserved	
4	SPICHEN	SPI DMA Chaining Enable. 0 = Disable 1 = Enable
6–5	Reserved	
7	FIFOFLSH	DMA FIFO Clear. 0 = Disable 1 = Enable
8	INTERR	Enable Interrupt on Error. 0 = Disable 1 = Enable
9	SPIOVF	Receive OverFlow Error (SPIRCV = 1). 0 = Successful transfer 1 = Error – data received with RXSPI full
10	SPIUNF	Transmit Underflow Error (SPIRCV = 0). 0 = Successful transfer 1 = Error occurred in transmission with no new data in TXSPI.
11	SPIMME	Multimaster Error. 0 = Successful transfer 1 = Error during transfer

Serial Peripheral Interface Registers

Table A-15. SPIDMAC, SPIDMACB Register Bit Descriptions (Cont'd)

Bit	Name	Description
13–12	SPISx	DMA FIFO Status. 00 = FIFO empty 11 = FIFO full 10 = FIFO partially full 01 = Reserved
14	SPIERRS	DMA Error Status. 0 = Successful DMA transfer 1 = Errors during DMA transfer
15	SPIDMAS	DMA Transfer Status. 0 = DMA idle 1 = DMA in progress
16	SPICHS	DMA Chain Loading Status. 0 = Chain idle 1 = Chain loading in progress
31–17	Reserved	

SPI DMA Start Address Registers (IISPI, IISPIB)

The reset values for these registers are undefined. These 19-bit read-write SPI registers contain the start address of the buffer in memory. Their addresses are 0x1080 (for IISPI) and 0x2880 (for IISPIB).

SPI DMA Address Modify Registers (IMSPI, IMSPIB)

The reset values for these registers are undefined. These 16-bit, read-write SPI registers contain the address modifier. Their addresses are 0x1081 (for IMSPI) and 0x2881 (for IMSPIB).

SPI DMA Word Count Registers (CSPI, CSPIB)

The reset values for these registers are undefined. These 16-bit, read-write SPI registers contain the number of words to be transferred. Their addresses are 0x1082 (for CSPI) and 0x2882 (for CSPIB).

SPI DMA Chain Pointer Registers (CPSPI, CPSPIB)

The reset values for these registers are undefined. These 20-bit, read-write SPI registers contain the address of the next TCB when DMA chaining is enabled. Their addresses are 0x1083 (for CPSPI) and 0x2883 (for CPSPIB).

Input Data Port Registers

The input data port (IDP) provides an additional input path to the processor core. The IDP is configurable as eight channels of serial data or seven channels of serial data and a single channel of up to a 20-bit wide parallel data. Six registers are used to specify modes, track status of inputs and outputs, and permit the IDP FIFO buffer to be read.

- “Input Data Port Control Register 0 (IDP_CTL0)” on page A-66
- “Input Data Port Control Register 1 (IDP_CTL1)” on page A-68
- “Input Data Port FIFO Register (IDP_FIFO)” on page A-69
- “Input Data Port DMA Control Registers” on page A-70
- “Input Data Port Ping-Pong DMA Registers” on page A-72
- “Parallel Data Acquisition Port Control Register (IDP_PP_CTL)” on page A-74

Input Data Port Registers

Input Data Port Control Register 0 (IDP_CTL0)

Use the IDP_CTL0 registers to configure and enable the input data port and each of its channels. This register is shown in [Figure A-27](#) and described in [Table A-16](#).



The IDP may also be routed through the DAI using its bits. [For more information, see “DAI/DPI Registers” on page A-109.](#)

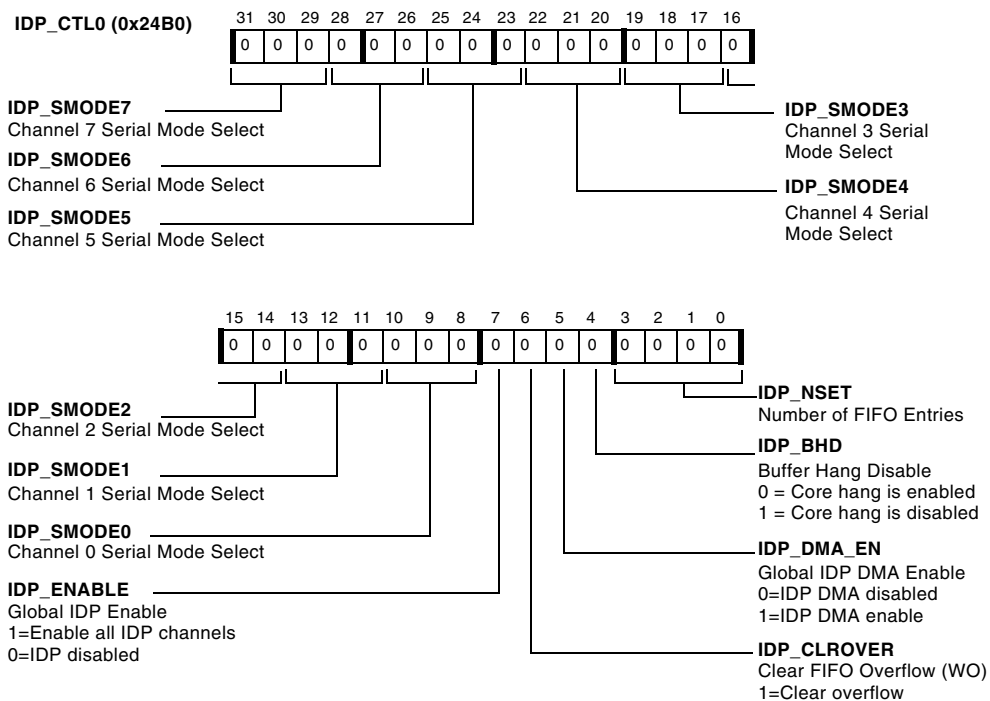


Figure A-27. IDP_CTL0 Register

Table A-16. IDP_CTL0 Register Bit Descriptions

Bit	Name	Description
3–0	IDP_NSET	Monitors number of FIFO entries where $N > \text{samples}$ raises interrupt controller bit 8.
4	IDP_BHD	IDP Buffer Hang Disable. Reads of an empty FIFO or writes to a full FIFO make the core hang. This condition continues until the FIFO has valid data (in the case of reads) or the FIFO has at least one empty location (in the case of writes). 0 = Core hang is enabled 1 = Core hang is disabled
5	IDP_DMA_EN	DMA Enable. Enables DMA on all IDP channels. 0 = DMA not enabled 1 = DMA enabled
6	IDP_CLROVR	FIFO Overflow Clear Bit. Writes of 1 to this bit clear the overflow condition in the DAI_STAT register. Because this is a write-only bit, it always returns LOW when read.
7	IDP_ENABLE	Enable IDP. 1 to 0 transition on this bit clears IDP_FIFO. 0 = IDP is disabled. Data does not come to IDP_FIFO from IDP channels. 1 = IDP is enabled.
10–8	IDP_SMODE0	Serial Input Mode Select. These eight inputs (0-7), each of which contains 3 bits, indicate the mode of the serial input for each of the eight IDP channels. Input format: 000 = Left-justified sample pair mode 001 = I ² S mode 010 = Left-justified 32 bits 011 = I ² S 32 bits 100 = Right-justified 24 bits 101 = Right-justified 20 bits 110 = Right-justified 18 bits 111 = Right-justified 16 bits
13–11	IDP_SMODE1	
16–14	IDP_SMODE2	
19–17	IDP_SMODE3	
22–20	IDP_SMODE4	
25–23	IDP_SMODE5	
28–26	IDP_SMODE6	
31–29	IDP_SMODE7	

Input Data Port Registers

Input Data Port Control Register 1 (IDP_CTL1)

Use the IDP_CTL1 register to configure and enable individual IDP channels. This register is shown in [Figure A-28](#) and described in [Table A-17](#).

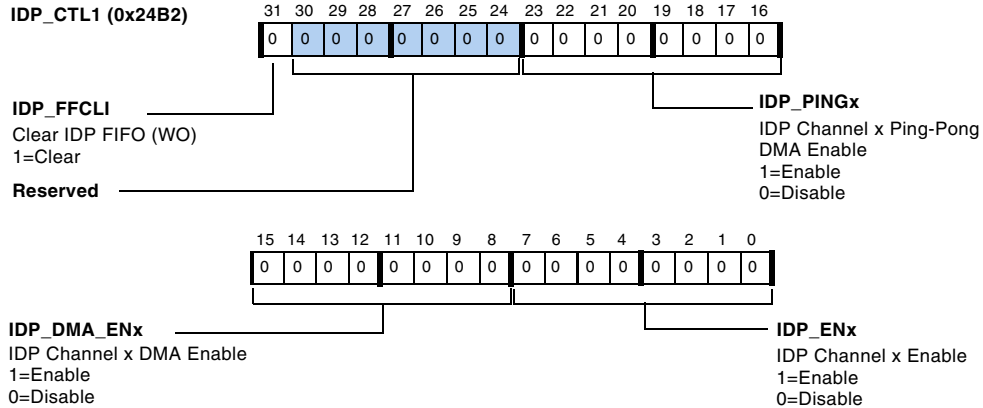


Figure A-28. IDP_CTL1 Register

Table A-17. IDP_CTL1 Register Bit Descriptions

Bit	Name	Description
7–0	IDP_ENx	IDP Channel Enable. Enables individual IDP channels. Bit 0 enables channel 0, bit 1 enables channel 1, and so on.
15–8	IDP_DMA_ENx	IDP DMA Enable. Enables standard DMA on all IDP channels. Bit 8 enables channel 0, bit 9 enables channel 1, and so on. 0 = DMA Disabled 1 = DMA Enabled
23–16	IDP_PINGx	DMA Ping-Pong Enable. Enables ping-pong DMA on all IDP channels. Bit 16 enables channel 0, bit 17 enables channel 1 and so on.
30–24	Reserved	
31	IDP_FFCLR	Clear IDP FIFO. Setting this bit to 1 clears IDP FIFO. This is a write-only bit and always returns 0 on reads.

Input Data Port FIFO Register (IDP_FIFO)

This register (shown in [Figure A-29](#)) provides information about the output of the IDP FIFO. Normally, IDP_FIFO is used only to read and remove the top sample from the FIFO. However, the core may also write to this register. When it does so, the audio data word is pushed into the input side of the FIFO as if it had come from the SRU on the channel encoded in the three LSBs. This can be useful for verifying the operation of the FIFO, the DMA channels, and the status portions of the IDP. The IDP FIFO is an 8-deep FIFO.

Channel-encoding provides for eight combinations, corresponding to the eight inputs. This register format applies when using channels 1–7 and also when using Channel 0 in serial mode. When using channel 0 in parallel mode, refer to the description of the four possible packing modes. [For more information, see “Packing Unit” on page 7-9.](#)



The information in [Table A-18](#) is not valid when data comes from the PDAP channel.

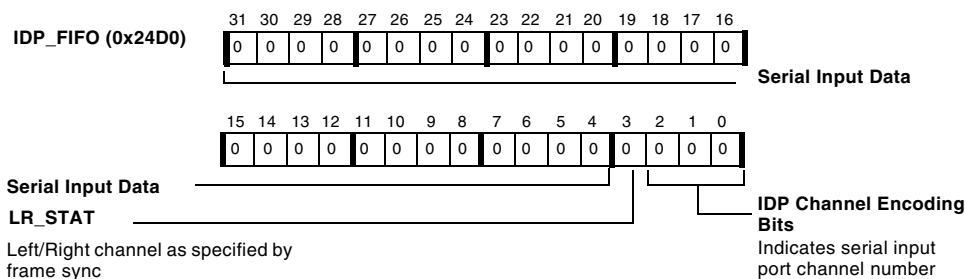


Figure A-29. IDP_FIFO Register

Input Data Port Registers

Table A-18. IDP_FIFO Register Bit Descriptions

Bit	Name	Description
2–0		IDP Channel Encoding. These bits indicate the serial input port channel number that provided this serial input data. Note: This information is not valid when data comes from the PDAP.
3	LR_STAT	Left/Right Channel Status. Indicates whether the data in bits 31-4 is the left or the right audio channel as dictated by the frame sync signal. The polarity of the encoding depends on the serial mode selected in IDP_SMODE for that channel. See Table A-16 on page A-67 .
31–4		Input Data (Serial). Some LSBs can be zero, depending on the mode.

Input Data Port DMA Control Registers

Each of the eight DMA channels have an I-register with an index pointer (19 bits), an M-register with a modifier/stride (6 bits), and a C-register with a count (16 bits). For example, IDP_DMA_I0, IDP_DMA_M0 and IDP_DMA_C0 control the DMA for IDP channel 0. The following sections describe these registers.

IDP_DMA_Ix

[Table A-19](#) provides information about the IDP DMA index registers.

Table A-19. IDP_DMA_Ix Registers

Register	Address	Reset State	Description
IDP_DMA_I0	0x2400	0x00000	IDP channel 0 DMA index register
IDP_DMA_I1	0x2401	0x00000	IDP channel 1 DMA index register
IDP_DMA_I2	0x2402	0x00000	IDP channel 2 DMA index register
IDP_DMA_I3	0x2403	0x00000	IDP channel 3 DMA index register
IDP_DMA_I4	0x2404	0x00000	IDP channel 4 DMA index register
IDP_DMA_I5	0x2405	0x00000	IDP channel 5 DMA index register

Table A-19. IDP_DMA_Ix Registers (Cont'd)

Register	Address	Reset State	Description
IDP_DMA_I6	0x2406	0x00000	IDP channel 6 DMA index register
IDP_DMA_I7	0x2407	0x00000	IDP channel 7 DMA index register

IDP_DMA_Mx

[Table A-20](#) provides information about the IDP DMA modifier registers.

Table A-20. IDP_DMA_Mx Registers

Register	Address	Reset State	Description
IDP_DMA_M0	0x2410	0x00	IDP channel 0 DMA modifier register
IDP_DMA_M1	0x2411	0x00	IDP channel 1 DMA modifier register
IDP_DMA_M2	0x2412	0x00	IDP channel 2 DMA modifier register
IDP_DMA_M3	0x2413	0x00	IDP channel 3 DMA modifier register
IDP_DMA_M4	0x2414	0x00	IDP channel 4 DMA modifier register
IDP_DMA_M5	0x2415	0x00	IDP channel 5 DMA modifier register
IDP_DMA_M6	0x2416	0x00	IDP channel 6 DMA modifier register
IDP_DMA_M7	0x2417	0x00	IDP channel 7 DMA modifier register

IDP_DMA_Cx

[Table A-21](#) provides information about the IDP DMA counter registers.

Table A-21. IDP_DMA_Cx Registers

Register	Address	Reset State	Description
IDP_DMA_C0	0x2420	0x00000	IDP channel 0 DMA count register
IDP_DMA_C1	0x2421	0x00000	IDP channel 1 DMA count register
IDP_DMA_C2	0x2422	0x00000	IDP channel 2 DMA count register

Input Data Port Registers

Table A-21. IDP_DMA_Cx Registers (Cont'd)

Register	Address	Reset State	Description
IDP_DMA_C3	0x2423	0x00000	IDP channel 3 DMA count register
IDP_DMA_C4	0x2424	0x00000	IDP channel 4 DMA count register
IDP_DMA_C5	0x2425	0x00000	IDP channel 5 DMA count register
IDP_DMA_C6	0x2426	0x00000	IDP channel 6 DMA count register
IDP_DMA_C7	0x2427	0x00000	IDP channel 7 DMA count register

Input Data Port Ping-Pong DMA Registers

Each of the eight DMA channels have an index register with an index pointer (19 bits) and a counter register with a count (16 bits) that are used when performing ping-pong DMA. For example, IDP_DMA_I1A and IDP_DMA_PC1 control ping-pong DMA for IDP channel 1. The following sections describe these registers.

IDP Ping-Pong Index Registers (IDP_DMA_IxA)

[Table A-22](#) provides information about the IDP ping-pong DMA index registers.

Table A-22. IDP_DMA_IxA Registers

Register	Address	Reset State	Description
IDP_DMA_I0A	0x2408	0x00000	IDP channel 0 index A ping-pong DMA register
IDP_DMA_I1A	0x2409	0x00000	IDP channel 1 index A ping-pong DMA register
IDP_DMA_I2A	0x240A	0x00000	IDP channel 3 index A ping-pong DMA register
IDP_DMA_I3A	0x240B	0x00000	IDP channel 4 index A ping-pong DMA register
IDP_DMA_I4A	0x240C	0x00000	IDP channel 4 index A ping-pong DMA register
IDP_DMA_I5A	0x240D	0x00000	IDP channel 5 index A ping-pong DMA register
IDP_DMA_I6A	0x240E	0x00000	IDP channel 6 index A ping-pong DMA register

Table A-22. IDP_DMA_IxA Registers (Cont'd)

Register	Address	Reset State	Description
IDP_DMA_I7A	0x240F	0x00000	IDP channel 7 index A ping-pong DMA register
IDP_DMA_I0B	0x2418	0x00000	IDP channel 0 index B ping-pong DMA register
IDP_DMA_I1B	0x2419	0x00000	IDP channel 1 index B ping-pong DMA register
IDP_DMA_I2B	0x241A	0x00000	IDP channel 2 index B ping-pong DMA register
IDP_DMA_I3B	0x241B	0x00000	IDP channel 3 index B ping-pong DMA register
IDP_DMA_I4B	0x241C	0x00000	IDP channel 4 index B ping-pong DMA register
IDP_DMA_I5B	0x241D	0x00000	IDP channel 5 index B ping-pong DMA register
IDP_DMA_I6B	0x241E	0x00000	IDP channel 6 index B ping-pong DMA register
IDP_DMA_I7B	0x241F	0x00000	IDP channel 7 index B ping-pong DMA register

IDP Ping-Pong Count Registers (IDP_DMA_PCx)

[Table A-23](#) provides information about the IDP ping-pong DMA count registers.

Table A-23. IDP_DMA_PCx Registers

Register	Address	Reset State	Description
IDP_DMA_PC0	0x2428	0x00000	IDP DMA channel 0 ping-pong count
IDP_DMA_PC1	0x2429	0x00000	IDP DMA channel 1 ping-pong count
IDP_DMA_PC2	0x242A	0x00000	IDP DMA channel 2 ping-pong count
IDP_DMA_PC3	0x242B	0x00000	IDP DMA channel 3 ping-pong count
IDP_DMA_PC4	0x242C	0x00000	IDP DMA channel 4 ping-pong count
IDP_DMA_PC5	0x242D	0x00000	IDP DMA channel 5 ping-pong count
IDP_DMA_PC6	0x242E	0x00000	IDP DMA channel 6 ping-pong count
IDP_DMA_PC7	0x242F	0x00000	IDP DMA channel 7 ping-pong count

Parallel Data Acquisition Port Control Register (IDP_PP_CTL)

Setting `IDP_PP_CTL[31]` enables either the 20 DAI pins or the `DATA31-8` pins to be used as a parallel input channel. These parallel words may be packed into 32-bit words for efficiency. The data then flows through the FIFO and is transferred by a dedicated DMA channel into the core's memory, as with any IDP channel.

The `IDP_PP_CTL` register, shown in [Figure A-30](#) and described in [Table A-24](#), provides 20 mask bits that allow the input from any of the 20 pins to be ignored. The mask is specified by setting the `IDP_Pxx_PDAPMASK` bits (bits 19–0 of the `IDP_PP_CTL` register) for the 20 parallel input signals. For each of the parallel inputs, a bit is set (= 1) to indicate the bit is unmasked and therefore its data can be passed on to be read, or masked (= 0), so its data is not read. After this masking process, data gets passed along to the packing unit.

For more information on the operation of the parallel data acquisition port, see [“Parallel Data Acquisition Port \(PDAP\)” on page 7-8](#). For information on the pin muxing that is used in conjunction with this module, see [“Pin Multiplexing” on page 14-2](#).

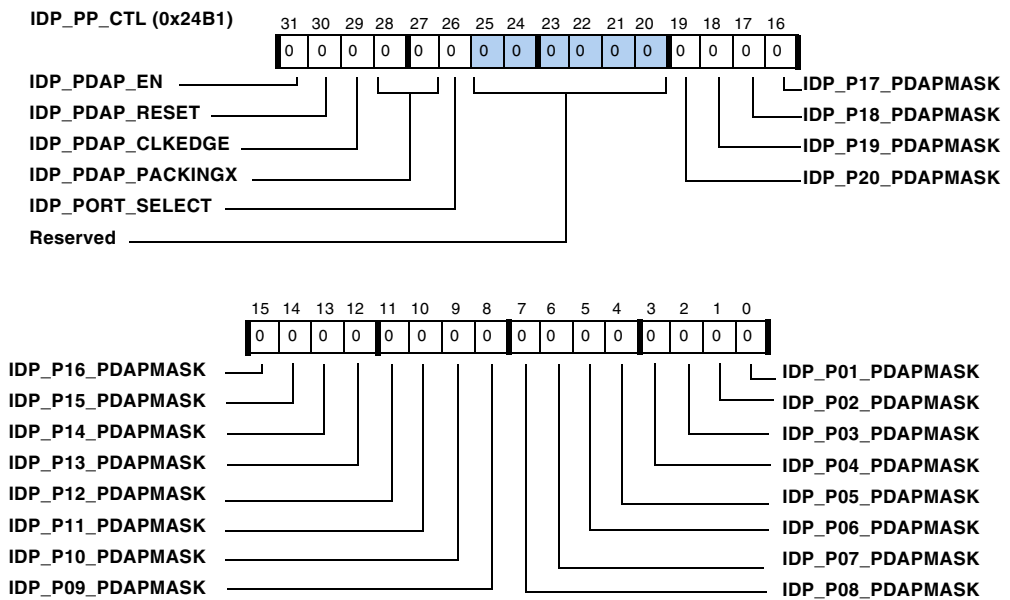


Figure A-30. IDP_PP_CTL Register

Table A-24. IDP_PP_CTL Register Bit Descriptions

Bit	Name	Description
0	IDP_P01_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_01 is masked 1 = Input data from DAI_01 is unmasked
1	IDP_P02_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_02 is masked 1 = Input data from DAI_02 is unmasked
2	IDP_P03_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_03 is masked 1 = Input data from DAI_03 is unmasked
3	IDP_P04_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_04 is masked 1 = Input data from DAI_04 is unmasked

Input Data Port Registers

Table A-24. IDP_PP_CTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
4	IDP_P05_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_05/DATA0 is masked 1 = Input data from DAI_05/DATA0 is unmasked
5	IDP_P06_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_06/DATA1 is masked 1 = Input data from DAI_06/DATA1 is unmasked
6	IDP_P07_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_07/DATA2 is masked 1 = Input data from DAI_07/DATA2 is unmasked
7	IDP_P08_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_08/DATA3 is masked 1 = Input data from DAI_08/DATA3 is unmasked
8	IDP_P09_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_09/DATA4 is masked 1 = Input data from DAI_09/DATA4 is unmasked
9	IDP_P10_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_10/DATA5 is masked 1 = Input data from DAI_10/DATA5 is unmasked
10	IDP_P11_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_11/DATA6 is masked 1 = Input data from DAI_11/DATA6 is unmasked
11	IDP_P12_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_12/DATA7 is masked 1 = Input data from DAI_12/DATA7 is unmasked
12	IDP_P13_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_13/ADDR0 is masked 1 = Input data from DAI_13/ADDR0 is unmasked
13	IDP_P14_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_14/ADDR1 is masked 1 = Input data from DAI_14/ADDR1 is unmasked
14	IDP_P15_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_15/ADDR2 is masked 1 = Input data from DAI_15/ADDR2 is unmasked

Table A-24. IDP_PP_CTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
15	IDP_P16_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_16/ADDR3 is masked 1 = Input data from DAI_16/ADDR3 is unmasked
16	IDP_P17_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_17/ADDR4 is masked 1 = Input data from DAI_17/ADDR4 is unmasked
17	IDP_P18_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_18/ADDR5 is masked 1 = Input data from DAI_18/ADDR5 is unmasked
18	IDP_P19_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_19/ADDR6 is masked 1 = Input data from DAI_19/ADDR6 is unmasked
19	IDP_P20_PDAPMASK	Parallel Data Acquisition Port Mask. 0 = Input data from DAI_20/ADDR7 is masked 1 = Input data from DAI_20/ADDR7 is unmasked
25–20	Reserved	
26	IDP_PORT_SELECT	Port Select: Input Pins Select. 0 = Data bits are read from DAI_P20–1 1 = Data bits are read from DATA31–12 and the control signals come from DATA11–8 and the PDAP can be operated through data pins alone (data and controls can be completely routed through DATA pins)
28–27	IDP_PDAP_PACKING	Packing. Selects PDAP packing mode. 00 = 8- to 32-bit packing 01 = (11, 11, 10) to 32-bit packing 10 = 16- to 32-bit packing 11 = 20- to 32-bit packing. 12 LSBs are set to 0
29	IDP_PDAP_CLKEDGE	PDAP (Rising or Falling) Clock Edge. 0 = Data is latched on the rising edge of the clock (IDP0_CLK_I) 1 = Data is latched on the falling edge

Pulse Width Modulation Registers

Table A-24. IDP_PP_CTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
30	IDP_PDAP_RESET	PDAP Reset. Setting this bit (=1) causes the PDAP reset circuit to strobe, then this bit is cleared automatically. This bit always returns a value of zero when read.
31	IDP_PDAP_EN	PDAP Enable. 0 = Disables the 20 DAI pins or the DATA31–8 pins from use as parallel input channels. 1 = Enables either the 20 DAI pins or the DATA31–8 pins to be used as a parallel input channel. IDP channel 0 cannot be used as a serial input port with this setting.

Pulse Width Modulation Registers

The following registers control the operation of pulse width modulation on the ADSP-21367/8/9 and ADSP-2137x processors.

PWM Global Control Register (PWMGCTL)

Use this register, shown in [Figure A-31](#), to enable or disable the four PWM groups in any combination. This provides synchronization across the groups. This 16-bit, read/write register is located at address 0x3800.

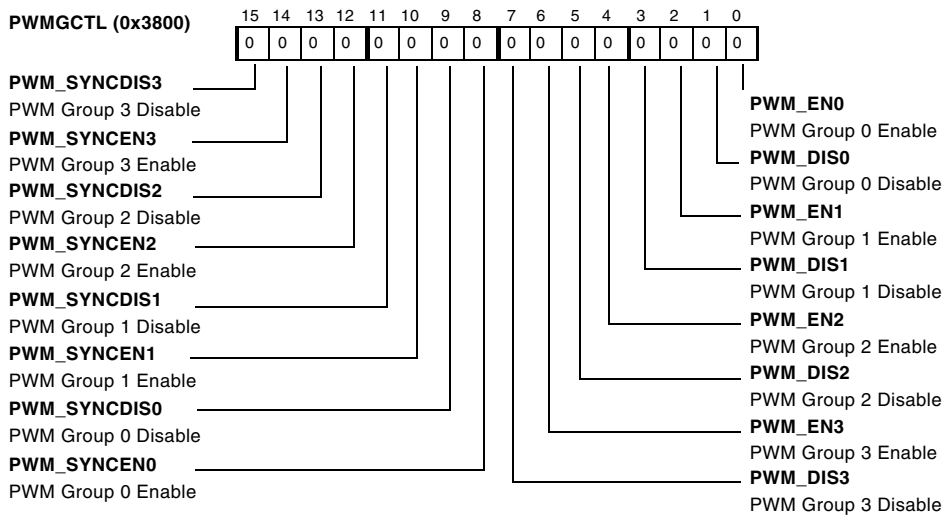


Figure A-31. PWMGCTL Register

PWM Global Status Register (PWMGSTAT)

This register, described in [Table A-25](#), provides the status of each PWM group and is located at address 0x3801. The bits in this register are W1C-type (write one-to-clear).

Table A-25. PWMGSTAT Register Bit Descriptions

Bit	Name	Description
0	PWM_STAT0	PWM Group 0 Period Completion Status
1	PWM_STAT1	PWM Group 1 Period Completion Status
2	PWM_STAT2	PWM Group 2 Period Completion Status
3	PWM_STAT3	PWM Group 3 Period Completion Status

PWM Control Register (PWMCTLx)

These registers, described in [Table A-26](#), are used to set the operating modes of each PWM block. These registers also allow programs to disable interrupts from individual groups. These registers addresses are:

PWMCTL0 — 0x3000

PWMCTL1 — 0x3010

PWMCTL2 — 0x3400

PWMCTL3 — 0x3410

Table A-26. PWMCTLx Register Bit Descriptions

Bit	Name	Function
0	PWM_ALIGN	Align Mode. 0 = Edge-aligned. The PWM waveform is left-justified in the period window. 1 = Center-aligned. The PWM waveform is symmetrical.
1	PWM_PAIR	Pair Mode. 0 = Non-paired mode. The PWM generates independent signals 1 = Paired mode. The PWM generates complementary signals on two outputs.
2	PWM_UPDATE	Update Mode. 0 = Single-update mode. The duty cycle values are programmable only once per PWM period. The resulting PWM patterns are symmetrical about the midpoint of the PWM period. 1 = Double-update mode. A second update of the PWM registers is implemented at the midpoint of the PWM period.
5	PWM_IRQEN	Enable PWM Interrupts. Enables interrupts. 0 = Interrupts not enabled 1 = Interrupts enabled

PWM Status Registers (PWMSTATx)

These 16-bit read-only registers, described in [Table A-27](#), report the status of the phase and mode for each PWM group. The addresses for these registers are:

PWMSTAT0 — 0x3001

PWMSTAT1 — 0x3011

PWMSTAT2 — 0x3401

PWMSTAT3 — 0x3411

Table A-27. PWMSTATx Register Bit Descriptions

Bit	Name	Description
0	PWM_PHASE	PWM Phase Status. Set during operation in the second half of each PWM period. Allows programs to determine the particular half-cycle during implementation of the PWMSYNC interrupt service routine, if required. 0 = First half 1 = Second half
2	PWM_PAIRSTAT	PWM Paired Mode Status. 0 = Inactive paired mode 1 = Active paired mode

PWM Period Registers (PWMPERIODx)

These 16-bit read/write registers control the period of the four PWM groups. The value written to the PWMPERIODx register is effectively the number of t_{CK} clock increments in one-half a PWM period. The addresses for these registers are:

PWMPERIOD0 — 0x3002

PWMPERIOD1 — 0x3012

PWMPERIOD2 — 0x3402

PWMPERIOD3 — 0x3412

PWM Output Disable Registers (PWMSEGx)

These 16-bit read/write registers, described in [Table A-28](#), control the output signals of the four PWM groups. The addresses for these registers are:

PWMSEG0 — 0x3008

PWMSEG1 — 0x3018

PWMSEG2 — 0x3408

PWMSEG3 — 0x3418

Table A-28. PWMSEGx Register Bit Descriptions

Bit	Name	Description
0	PWM_BH	Channel B High Disable. Enables or disables the channel B output signal. 0 = Enable 1 = Disable
1	PWM_BL	Channel B Low Disable. Enables or disables the channel B output signal. 0 = Enable 1 = Disable
2	PWM_AH	Channel A High Disable. Enables or disables the channel A output signal. 0 = Enable 1 = Disable
3	PWM_AL	Channel A Low Disable. Enables or disables the channel A output signal. 0 = Enable 1 = Disable

PWM Polarity Select Registers (PWMPOLx)

These registers, described in [Table A-29](#), control the polarity of the four PWM groups which can be set to either active hi or active lo. The addresses for these registers are:

PWMPOL0 — 0x300F

PWMPOL1 — 0x301F

PWMPOL2 — 0x340F

PWMPOL3 — 0x341F

Table A-29. PWMPOLx Register Bit Descriptions

Bit	Name	Description
0	PWM_POL1AL	Write to Set Channel A Low Polarity 1
1	PWM_POL0AL	Write to Set Channel A Low Polarity 0
2	PWM_POL1AH	Write to Set Channel A High Polarity 1
3	PWM_POL0AH	Write to Set Channel A High Polarity 0
4	PWM_POL1BL	Write to Set Channel B Low Polarity 1
5	PWM_POL0BL	Write to Set Channel B Low Polarity 0
6	PWM_POL1BH	Write to set channel B High Polarity 1
7	PWM_POL0BH	Write to set channel B High Polarity 0

PWM Channel Duty Control Registers (PWMAx, PWMBx)

The duty cycle control registers, described in [Table A-30](#), directly control the duty cycles of the two pairs of PWM signals. These 16-bit, read/write registers are located at addresses:

PWMA0 — 0x3005	PWMB0 — 0x3006
PWMA1 — 0x3015	PWMB1 — 0x3016
PWMA2 — 0x3405	PWMB2 — 0x3406
PWMA3 — 0x3415	PWMB3 — 0x3416

Table A-30. PWMAx/PWMBx Register Bit Descriptions

Bit	Name	Description
15–0	PWMAx	Channel A Duty Cycle. Program a two's complement duty cycle with a value of 0x0000 through 0xFFFF. Default = 0
15–0	PWMBx	Channel B Duty Cycle. Program a two's complement duty cycle with a value of 0x0000 through 0xFFFF. Default = 0

PWM Channel Low Duty Control Registers (PWMALx, PWMBLx)

In non-paired mode, these registers, described in [Table A-31](#) are used to program the low side duty cycle. These can be different then the high-side cycles. These 16-bit read/write registers are located at addresses:

PWMAL0 — 0x300A	PWMBL0 — 0x300B
PWMAL1 — 0x301A	PWMBL1 — 0x301B
PWMAL2 — 0x340A	PWMBL2 — 0x340B
PWMAL3 — 0x341A	PWMBL3 — 0x341B

Table A-31. PWMALx/PWMBLx Register Bit Descriptions

Bit	Name	Description
15–0	PWMALx	Channel AL Duty Cycle. Program a two's complement duty cycle with a value of 0x0000 through 0xFFFF. Default = 0
15–0	PWMBLx	Channel BL Duty Cycle. Program a two's complement duty cycle with a value of 0x0000 through 0xFFFF. Default = 0

PWM Dead Time Registers (PWMDTx)

These registers, described in [Table A-32](#), set up a short time delay between turning off one PWM signal and turning on its complementary signal. These 10-bit, read/write registers are located at addresses:

PWMDT0—0x3003 PWMDT2—0x3403
PWMDT1—0x0013 PWMDT3—0x3413

Table A-32. PWMDTx Register Bit Descriptions

Bit	Name	Description
9–0	PWMDT	PWM Dead Time (unsigned). Program a time delay setting of 0x0000 to 0x03FF. Default = 0
15–10	Reserved	

Sony/Philips Digital Interface Registers

The following sections describe the registers that are used to configure, enable and report status information for the S/PDIF transceiver.

Transmitter Control Register (DITCTL)

The S/PDIF transmit control register (DITCTL) is a 32-bit, read/write register located at address 0x24A0. The register's bits are shown in [Figure A-32](#), [Figure A-33](#) and described in [Table A-33](#).

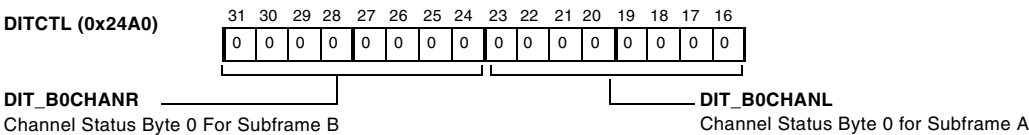


Figure A-32. DITCTL Register (Bits 31–16)

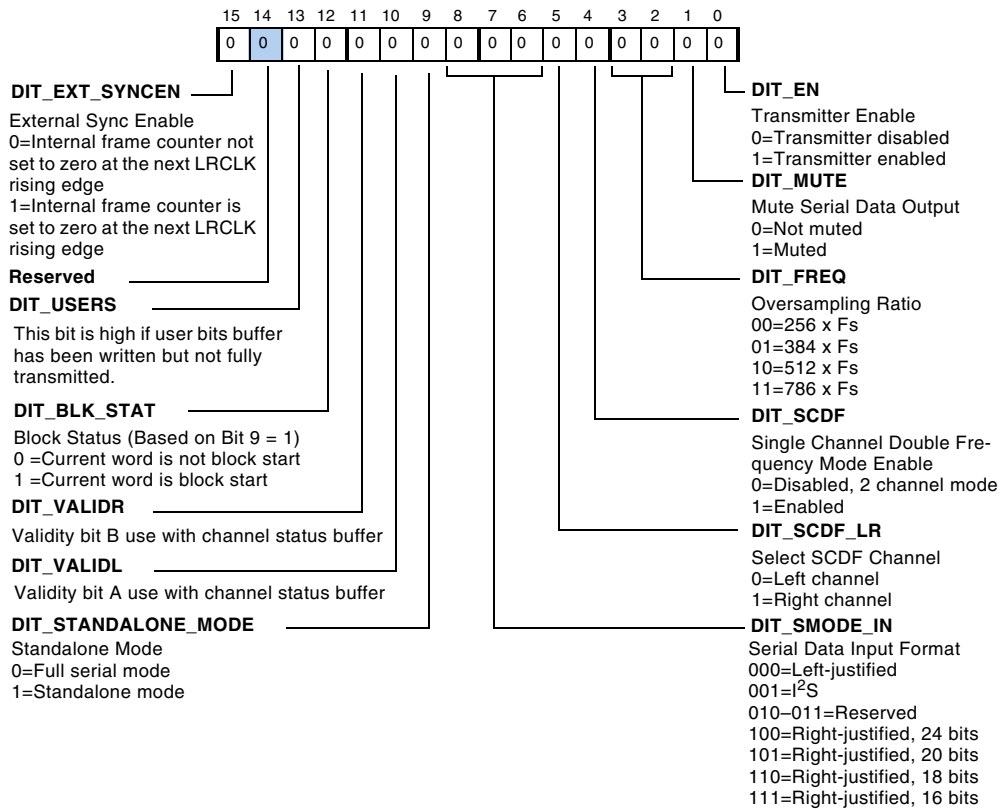


Figure A-33. DITCTL Register (Bits 15–0)

Table A-33. DITCTL Register Bit Descriptions

Bit	Name	Description
0	DIT_EN	Transmitter Enable. Enables the transmitter and resets the control registers to their defaults. 0 = Transmitter disabled 1 = Transmitter enabled
1	DIT_MUTE	Mute. Mutes the serial data output.

Sony/Philips Digital Interface Registers

Table A-33. DITCTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
3–2	DIT_FREQ	Frequency Multiplier. Sets the oversampling ratio to the following: 00 = 256 x Frame sync 01 = 384 x Frame sync 10 = 512 x Frame sync 11 = 768 x Frame sync
4	DIT_SCDF	Transmit Single-Channel, Double-Frequency Enable.
5	DIT_SCDF_LR	Select single-channel, double-frequency mode channel. (L = 0, R = 1)
8–6	DIT_SMODEIN	Serial Data Input Format. Select the input format as follows: 000 = Left-justified 001 = I ² S 010 = Reserved 011 = Reserved 100 = Right-justified, 24-bits 101 = Right-justified, 20-bits 110 = Right-justified, 18-bits 111 = Right-justified, 16-bits
9	DIT_STANDALONE_MODE	Standalone Mode Enable. When this bit is set (=1), the transmitter is in standalone mode. In this mode the transmitter inserts block start bits automatically. Channel status bits and user bits are taken from the respective buffers. Valid bit is set based on bits 10 and 11 in this DITCTL register for channel A and B respectively. Bits received with serial data are ignored. When this bit is cleared (=0), the transmitter is in full serial mode. In this mode, all the status bits, including the block start bit, (indicating start of a frame) come through the SDATA pin along with audio data. The transmitter should be enabled after or with all other control bits.
10	DIT_VALIDL	Validity Bit A. Use with channel status buffer.
11	DIT_VALIDR	Validity Bit B. Use with channel status buffer.
12	DIT_BLK_STAT	Block Status (read-only). Status bit that indicates block start (when bit 9, DIT_AUTO, = 1). 0 = Current word is not block start 1 = Current word is block start

Table A-33. DITCTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
13	DIT_USERS	User Bits Status. This bit is high if user bits buffer has been written but data has not been transmitted completely.
14	Reserved	
15	DIT_EXT_SYNCEN	External Sync Enable. When this bit is set, the internal frame counter is set to zero at the next LRCLK rising edge.
23–16	DIT_B0CHANL	Channel status byte 0 for subframe A
31–24	DIT_B0CHANR	Channel status byte 0 for subframe B

Left Channel Status for Subframe A Registers (DITCHANAx)

There are six channel status buffer registers associated with subframe A. These registers are listed with their locations in [Table A-34](#).

Table A-34. DITCHANAx Registers

Register (Address)	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DITCHANA0 (0x24A1)	BYTE1	BYTE2	BYTE3	BYTE4
DITCHANA1 (0x24D4)	BYTE5	BYTE6	BYTE7	BYTE8
DITCHANA2 (0x24D5)	BYTE9	BYTE10	BYTE11	BYTE12
DITCHANA3 (0x24D6)	BYTE13	BYTE14	BYTE15	BYTE16
DITCHANA4 (0x24D7)	BYTE17	BYTE18	BYTE19	BYTE20
DITCHANA5 (0x24D8)	BYTE21	BYTE22	BYTE23	Reserved

Right Channel Status for Subframe B Registers (DITCHANBx)

There are six channel status buffer registers associated with subframe B. These registers are listed with their locations in [Table A-35](#).

Table A-35. DITCHANBx Registers

Register (Address)	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DITCHANB0 (0x24A2)	BYTE1	BYTE2	BYTE3	BYTE4
DITCHANB1 (0x24DA)	BYTE5	BYTE6	BYTE7	BYTE8
DITCHANB2 (0x24DB)	BYTE9	BYTE10	BYTE11	BYTE12
DITCHANB3 (0x24DC)	BYTE13	BYTE14	BYTE15	BYTE16
DITCHANB4 (0x24DD)	BYTE17	BYTE18	BYTE19	BYTE20
DITCHANB5 (0x24DE)	BYTE21	BYTE22	BYTE23	Reserved

User Bits Buffer Registers for Subframe A Registers (DITUSRBITAx)

There are six user bits buffer registers associated with subframe A (left channel). These registers are listed with their locations in [Table A-36](#).

Table A-36. DITUSRBITAx Registers

Register (Address)	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DITUSRBITA0 (0x24E0)	BYTE0	BYTE1	BYTE2	BYTE3
DITUSRBITA1 (0x24E1)	BYTE4	BYTE5	BYTE6	BYTE7
DITUSRBITA2 (0x24E2)	BYTE8	BYTE9	BYTE10	BYTE11
DITUSRBITA3 (0x24E3)	BYTE12	BYTE13	BYTE14	BYTE15
DITUSRBITA4 (0x24E4)	BYTE16	BYTE17	BYTE18	BYTE19
DITUSRBITA5 (0x24E5)	BYTE20	BYTE21	BYTE22	BYTE23

User Bits Buffer Registers for Subframe B Registers (DITUSRBITBx)

There are six user bits buffer registers associated with subframe B (right channel). These registers are listed with their locations in [Table A-37](#).

Table A-37. DITUSRBITBx Registers

Register (Address)	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DITUSRBITB0 (0x24E8)	BYTE0	BYTE1	BYTE2	BYTE3
DITUSRBITB1 (0x24E9)	BYTE4	BYTE5	BYTE6	BYTE7
DITUSRBITB2 (0x24EA)	BYTE8	BYTE9	BYTE10	BYTE11
DITUSRBITB3 (0x24EB)	BYTE12	BYTE13	BYTE14	BYTE15
DITUSRBITB4 (0x24EC)	BYTE16	BYTE17	BYTE18	BYTE19
DITUSRBITB5 (0x24ED)	BYTE20	BYTE21	BYTE22	BYTE23

Receiver Control Register (DIRCTL)

This 32-bit read/write register is used to set up error control and single-channel, double-frequency mode. The register is located at address 0x24A8. The register's bits are shown in [Figure A-34](#) and described in [Table A-38](#).

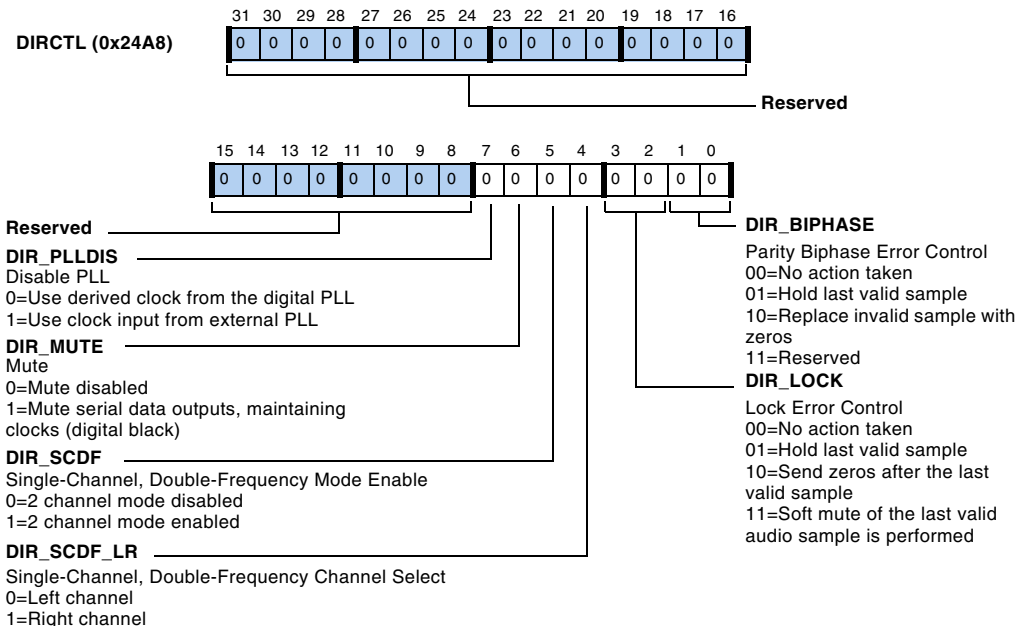


Figure A-34. DIRCTL Register

Table A-38. DIRCTL Register Bit Descriptions

Bit	Name	Description
1–0	DIR_BIPHASE	Parity Biphase Error Control. 00 = No action taken 01 = Hold last valid sample 10 = Replace invalid sample with zeros 11 = Reserved
3–2	DIR_LOCK	Lock Error Control. 00 = No action taken 01 = Hold last valid sample 10 = Send zeros after the last valid sample 11 = Soft mute of the last valid audio sample is performed as if NOSTREAM is asserted. This is valid only when linear PCM audio data is in the stream. With non-linear audio data, this mode defaults to LOCKERROR_CTL = 10.
4	DIR_SCDF_LR	Single-Channel, Double-Frequency Channel Select. 0 = Left channel 1 = Right channel
5	DIR_SCDF	Single-Channel, Double-Frequency Mode Enable. 0 = 2 channel mode disabled 1 = 2 channel mode enabled
6	DIR_MUTE	Mute. 0 = Mute disabled 1 = Mute serial data outputs, maintaining clocks (digital black)
7	DIR_PLLDIS	Disable PLL. Determines clock input. 0 = Use derived clock from the digital PLL 1 = Use clock input from external PLL
31–8	Reserved	

Receiver Status Register (DIRSTAT)

This 32-bit, read-only register is used to store the error bits. The error bits are sticky on read. Once they are set, they remain set until the register is read. This register also contains the lower byte of the 40-bit channel status information. The register is located at address 0x24A9. The register's bits are shown in [Figure A-35](#) and described in [Table A-39](#).

DIRSTAT (0x24A9)

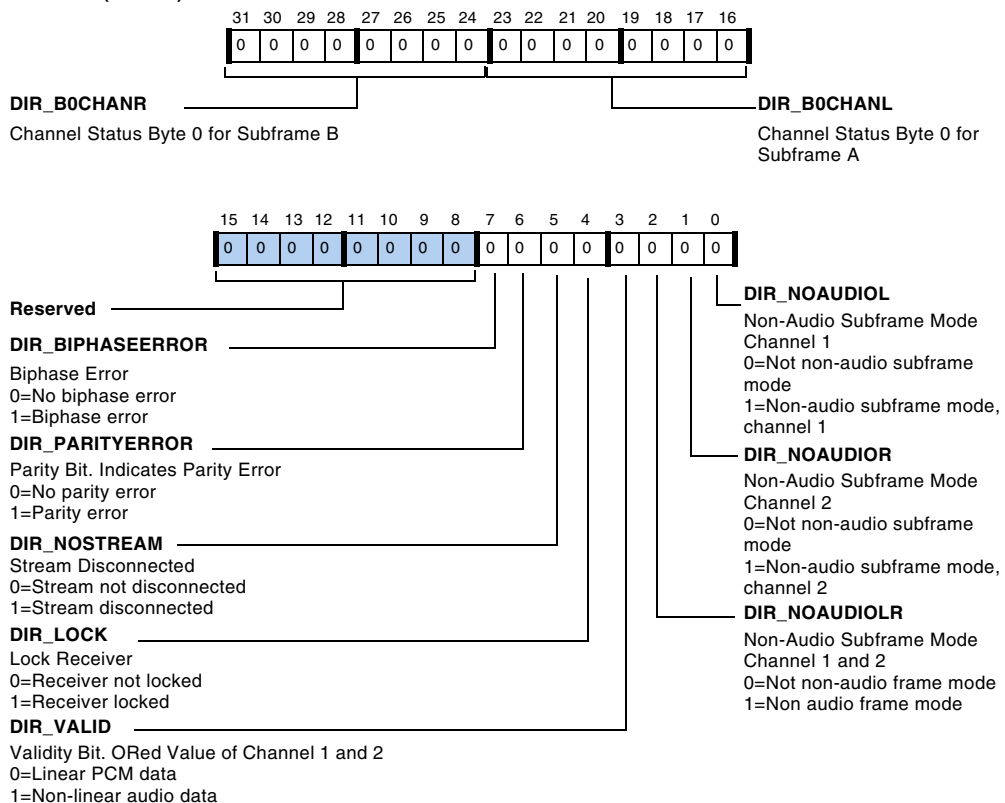


Figure A-35. DIRSTAT Register

Table A-39. DIRSTAT Register Bit Descriptions

Bit	Name	Description
0	DIR_NOAUDIOL	Non-Audio Subframe Mode Channel 1. Based on SMPTE 337M. 0 = Not non-audio subframe mode 1 = Non-audio subframe mode, Channel 1
1	DIR_NOAUDIOR	Non-Audio Subframe Mode Channel 2. Based on SMPTE 337M. 0 = Not non-audio subframe mode 1 = Non-audio subframe mode, Channel 2
2	DIR_NOAUDIOLR	Non-Audio Frame Mode Channel 1 and 2. Based on SMPTE 337M. 0 = Not non-audio frame mode 1 = Non audio frame mode
3	DIR_VALID	Validity Bit. ORed value of channel 1 and 2. 0 = Linear PCM data 1 = Non-linear audio data
4	DIR_LOCK	Lock Receiver. 0 = Receiver not locked 1 = Receiver locked
5	DIR_NOSTREAM	Stream Disconnected. Indicates that the data stream is disconnected. 0 = Stream not disconnected 1 = Stream disconnected
6	DIR_PARITYERROR	Parity Bit. Indicates parity error. 0 = No parity error 1 = Parity error
7	DIR_BIPHASEERROR	Biphase Error Indicates biphase error. 0 = No biphase error 1 = Biphase error
15–8	Reserved	
23–16	DIR_B0CHANL	Channel Status Byte 0 for Subframe A.
31–24	DIR_B0CHANR	Channel Status Byte 0 for Subframe B.

Left Channel Status for Subframe A Register (DIRCHANL)

This register (`DIRCHANL`, described in [Table A-40](#)) is a 32-bit, read/write register located at address 0x24AA.

Table A-40. DIRCHANL Register

Bit	Name	Description
7–0	DIR_B1CHANL	Channel status byte 1 for subframe A
15–8	DIR_B2CHANL	Channel status byte 2 for subframe A
23–16	DIR_B3CHANL	Channel status byte 3 for subframe A
31–24	DIR_B4CHANL	Channel status byte 4 for subframe A

Right Channel Status for Subframe B Register (DIRCHANR)

This register (`DIRCHANR`, described in [Table A-41](#)) is a 32-bit, read/write register located at address 0x24AB.

Table A-41. DIRCHANR Register

Bit	Name	Description
7–0	DIR_B1CHANR	Channel status byte 1 for subframe B
15–8	DIR_B2CHANR	Channel status byte 2 for subframe B
23–16	DIR_B3CHANR	Channel status byte 3 for subframe B
31–24	DIR_B4CHANR	Channel status byte 4 for subframe B

Sample Rate Converter Registers

The sample rate converter (SRC) is composed of five registers which are described in the following sections.

SRC Control Registers (SRCCTLx)

These registers (read/write) control the operating modes, filters, and data formats used in the SRCs and are shown in [Figure A-36](#) through [Figure A-39](#) and described in [Table A-42](#) and [Table A-43](#). The SRCCTL0 register controls the SRC0 and SRC1 modules and SRCCTL1 register controls the SRC2 and SRC3 modules.

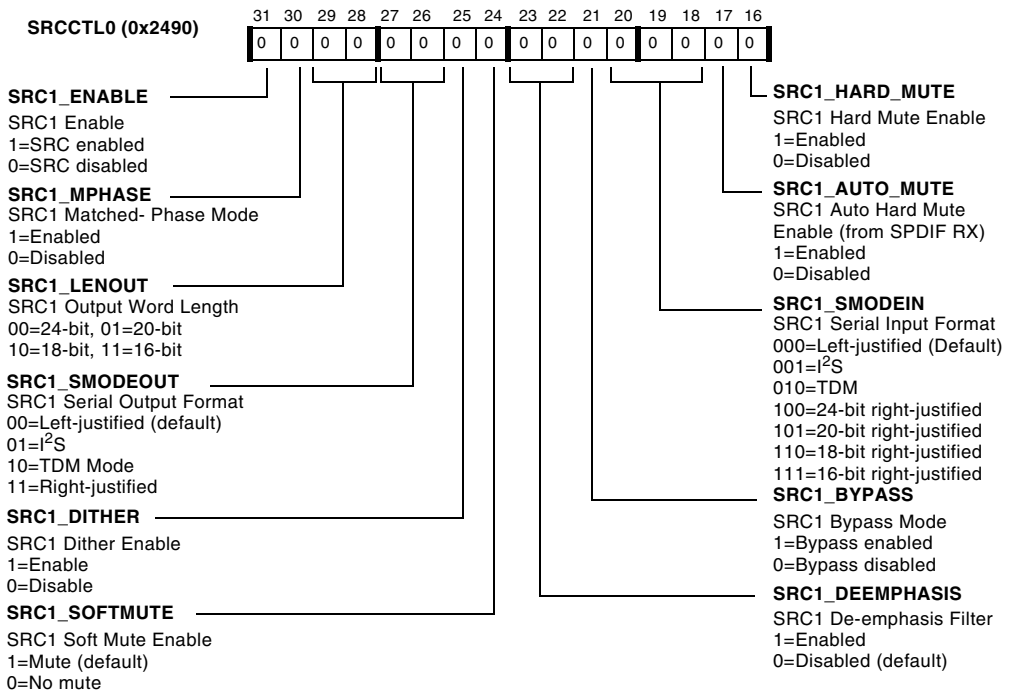


Figure A-36. SRCCTL0 Register (Bits 16–31)

Sample Rate Converter Registers

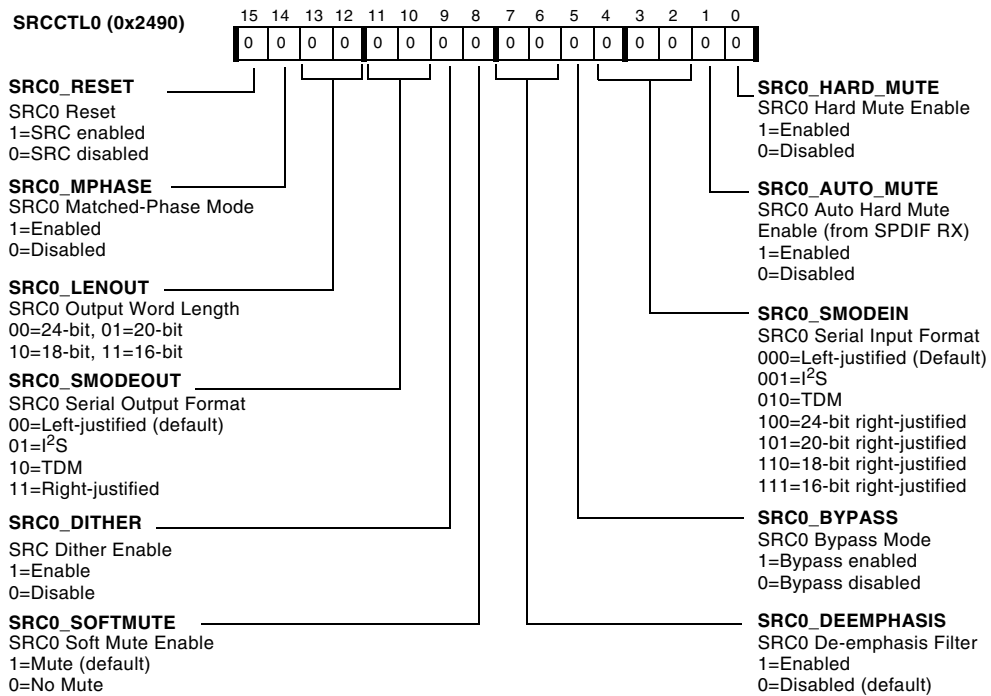


Figure A-37. SRCCTL0 Register (Bits 0–15)

Table A-42. SRCCTL0 Register Bit Descriptions

Bit	Name	Description
0	SRC0_HARD_MUTE	Hard Mute. Hard mutes SRC 0. 1 = Mute (default)
1	SRC0_AUTO_MUTE	Auto Hard Mute. Auto hard mutes SRC 0 when one of the non-audio bits is asserted by the SPDIF receiver. See Table A-39 on page A-95 . 0 = No mute 1 = Mute (default)

Table A-42. SRCCTL0 Register Bit Descriptions (Cont'd)

Bit	Name	Description
4–2	SRC0_SMODEIN	Serial Input Format. Selects the serial input format for SRC 0 as follows. 000 = Default, format is left-justified 001 = I ² S 010 = TDM 100 = 24-bit right-justified 101 = 20-bit right-justified 110 = 18-bit right-justified 111 = 16-bit right-justified
5	SRC0_BYPASS	Bypass SRC0. Output of SRC 0 is the same as the input.
7–6	SRC0_DEEMPHASIS	De-emphasis Filter Select. Enables de-emphasis on incoming audio data for SRC 0. 00 = No Deemphasis 01 = 33 kHz 10 = 44.1 kHz 11 = 48 kHz
8	SRC0_SOFTMUTE	Soft Mute. Enables soft mute on SRC 0. 0 = No mute 1 = Mute (default)
9	SRC0_DITHER	Dither Select. Enables dithering on SRC 0 when a word length less than 24 bits is selected. 0 = Dithering is disabled (default) 1 = Dithering is enabled
11–10	SRC0_SMODEOUT	Serial Output Format. Selects the serial output format on SRC 0, as follows: 00 = Left-justified (default) 01 = I ² S 10 = TDM mode 11 = Right-justified

Sample Rate Converter Registers

Table A-42. SRCCTL0 Register Bit Descriptions (Cont'd)

Bit	Name	Description
13–12	SRC0_LENOUT	Output Word Length Select. Selects the serial output word length on SRC 0 as follows: 00 = 24 bits 01 = 20 bits 10 = 18 bits 11 = 16 bits Any word length less than 24 bits will have dither added to the unused LSBs if SRCx_DITHER is enabled (= 1).
14	SRC0_MPHASE	Match-Phase Mode Select. Configures the SRC 0 modules to not use their own internally-generated sample rate ratio but use an externally-generated ratio. Used with TDM data. 0 = Matched-phase mode disabled (default) 1 = Matched-phase mode enabled
15	SRC0_ENABLE	SRC0 Enable. Enables SRC 0. 0 = Disabled 1 = Enabled
16	SRC1_HARD_MUTE	Hard Mute. Hard mutes SRC 1. 1 = Mute (default)
17	SRC1_AUTO_MUTE	Auto Hard Mute. Auto hard mutes SRC 1 when one of the non-audio bits is asserted by the SPDIF receiver. See Table A-39 on page A-95 . 0 = No mute 1 = Mute (default)
20–18	SRC1_SMODEIN	Serial Input Format. Selects the serial input format for SRC 1 as follows. 000 = Default, format is left-justified 001 = I ² S 010 = TDM 100 = 24-bit right-justified 101 = 20-bit right-justified 110 = 18-bit right-justified 111 = 16-bit right-justified
21	SRC1_BYPASS	Bypass Mode Enable. Output of SRC 1 is the same as input.

Table A-42. SRCCTL0 Register Bit Descriptions (Cont'd)

Bit	Name	Description
23–22	SRC1_DEEMPHASIS	De-emphasis Filter Select. Enables deemphasis on incoming audio data for SRC 1. 00 = No de-emphasis 01 = 33 kHz 10 = 44.1 kHz 11 = 48 kHz
24	SRC1_SOFTMUTE	Soft Mute. Enables soft mute on SRC 1. 0 = No mute 1 = Mute (default)
25	SRC1_DITHER	Dither Select. Disables dithering on SRC 1 when a word length less than 24 bits is selected. 0 = Dithering is disabled (default) 1 = Dithering is enabled
27–26	SRC1_SMODEOUT	Serial Output Format. Selects the serial output format for SRC 1 as follows. 00 = Left-justified (default) 01 = I ² S 10 = TDM mode 11 = Right-justified
29–28	SRC1_LENOUT	Output Word Length Select. Selects the serial output word length for SRC 1 as follows. 00 = 24 bits 01 = 20 bits 10 = 18 bits 11 = 16 bits Any word length less than 24 bits will have dither added to the unused LSBs if SRCx_DITHER is enabled (= 1).
30	SRC1_MPHASE	Match-Phase Mode Select. Configures the SRC 1 modules to not use their own internally-generated sample rate ratio but use an externally-generated ratio. Used with TDM data. 0 = Matched-phase disabled (default) 1 = Matched-phase enabled
31	SRC1_ENABLE	SRC Enable. Enables SRC 1. 0 = Disabled 1 = Enabled

Sample Rate Converter Registers

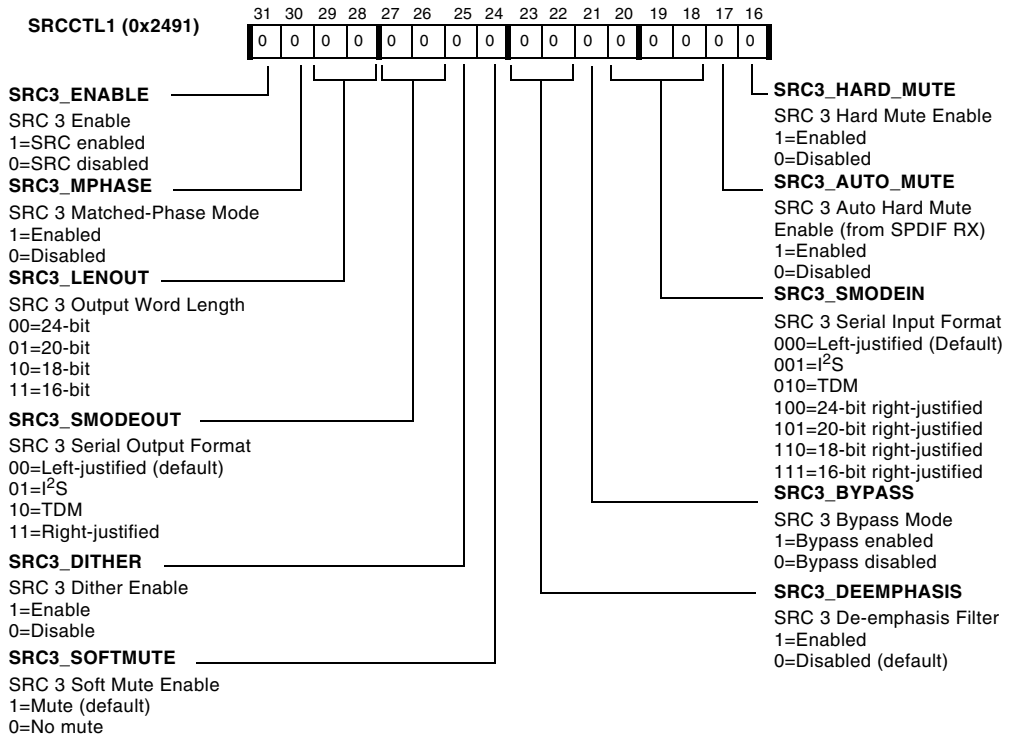


Figure A-38. SRCCTL1 Register (Bits 16–31)

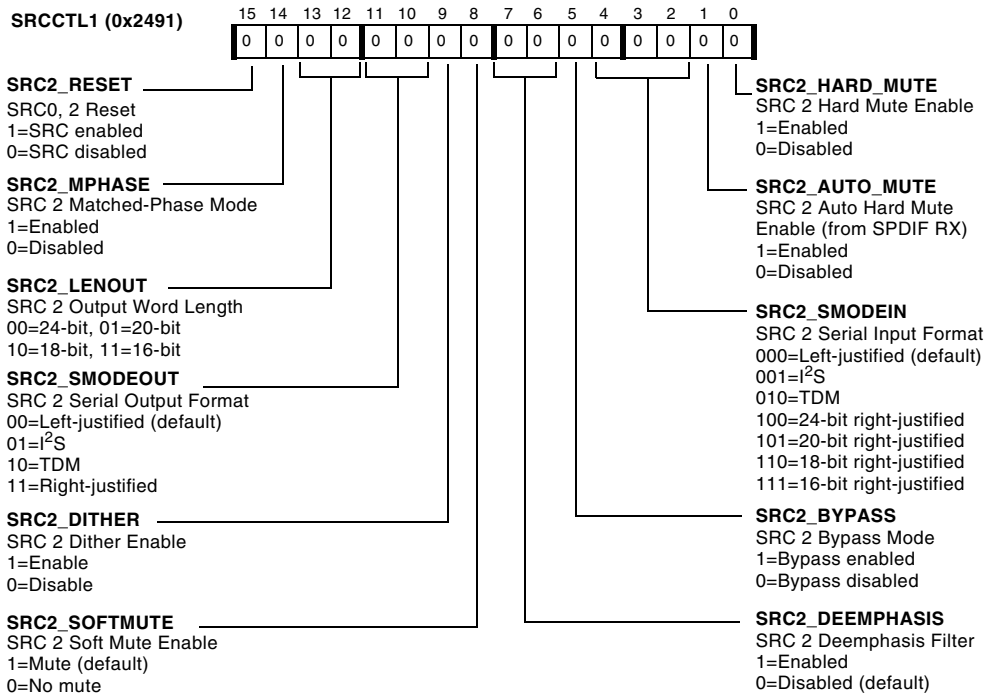


Figure A-39. SRCCTL1 Register (Bits 0–15)

Table A-43. SRCCTL1 Register Bit Descriptions

Bit	Name	Description
0	SRC2_HARD_MUTE	Hard Mute. Hard mutes SRC 2. 1 = Mute (default)
1	SRC2_AUTO_MUTE	Auto Hard Mute. Auto hard mutes SRC 2 when one of the non-audio bits is asserted by the SPDIF receiver. See Table A-39 on page A-95 . 0 = No mute 1 = Mute (default)

Sample Rate Converter Registers

Table A-43. SRCCTL1 Register Bit Descriptions (Cont'd)

Bit	Name	Description
4–2	SRC2_SMODEIN	Serial Input Format. Selects the serial input format for SRC 2 as follows. 000 = Default, format is left-justified 001 = I ² S 010 = TDM 100 = 24-bit right-justified 101 = 20-bit right-justified 110 = 18-bit right-justified 111 = 16-bit right-justified
5	SRC2_BYPASS	Bypass SRCx. Output of SRC 2 is the same as input
7–6	SRC2_DEEMPHASIS	De-emphasis Filter Select. Enables de-emphasis on incoming audio data for SRC 2. 00 = No de-emphasis 01 = 33 kHz 10 = 44.1 kHz 11 = 48 kHz
8	SRC2_SOFTMUTE	Soft Mute. Enables soft mute on SRC 2. 0 = No mute 1 = Mute (default)
9	SRC2_DITHER	Dither Select. Enables dithering on SRC 2 when a word length less than 24 bits is selected. 0 = Dithering is disabled (default) 1 = Dithering is enabled
11–10	SRC2_SMODEOUT	Serial Output Format. Selects the serial output format on SRC 2 as follows: 00 = Left-justified (default) 01 = I ² S 10 = TDM mode 11 = Right-justified

Table A-43. SRCCTL1 Register Bit Descriptions (Cont'd)

Bit	Name	Description
13–12	SRC2_LENOUT	Output Word Length Select. Selects the serial output word length on SRC 2 as follows: 00 = 24 bits 01 = 20 bits 10 = 18 bits 11 = 16 bits Any word length less than 24 bits will have dither added to the unused LSBs if SRCx_DITHER is enabled (= 1).
14	SRC2_MPHASE	Match-Phase Mode Select. Configures the SRC 2 modules to not use their own internally-generated sample rate ratio but use an externally-generated ratio. Used with TDM data. 0 = Matched-phase disabled (default) 1 = Matched-phase enabled
15	SRC2_ENABLE	SRCx Enable. Enables SRC 2. 0 = Disabled 1 = Enabled
16	SRC3_HARD_MUTE	Hard Mute. Hard mutes SRC 3. 1 = Mute (default)
17	SRC3_AUTO_MUTE	Auto Hard Mute. Auto hard mutes SRC 3 when one of the non-audio bits is asserted by the SPDIF receiver. See Table A-39 on page A-95 . 0 = No mute 1 = Mute (default)
20–18	SRC3_SMODEIN	Serial Input Format. Selects the serial input format for SRC 3 as follows: 000 = Default, format is left-justified 001 = I ² S 010 = TDM 100 = 24-bit right-justified 101 = 20-bit right-justified 110 = 18-bit right-justified 111 = 16-bit right-justified
21	SRC3_BYPASS	Bypass Mode Enable. Output of SRC 3 is the same as input.

Sample Rate Converter Registers

Table A-43. SRCCTL1 Register Bit Descriptions (Cont'd)

Bit	Name	Description
23–22	SRC3_DEEMPHASIS	De-emphasis Filter Select. Enables de-emphasis on incoming audio data for SRC 3. 00 = No de-emphasis 01 = 33 kHz 10 = 44.1 kHz 11 = 48 kHz
24	SRC3_SOFTMUTE	Soft Mute. Enables soft mute on SRC 3. 0 = No mute 1 = Mute (default)
25	SRC3_DITHER	Dither Select. Disables dithering on SRC 3 when a word length less than 24 bits is selected. 0 = Dithering is disabled (default) 1 = Dithering is enabled
27–26	SRC3_SMODEOUT	Serial Output Format. Selects the serial output format for SRC 3 as follows: 00 = Left-justified (default) 01 = I ² S 10 = TDM mode 11 = Right-justified
29–28	SRC3_LENOUT	Output Word Length Select. Selects the serial output word length for SRC 3 as follows: 00 = 24 bits 01 = 20 bits 10 = 18 bits 11 = 16 bits Any word length less than 24 bits will have dither added to the unused LSBs if SRCx_DITHER is enabled (= 1).
30	SRC3_MPHASE	Match-Phase Mode Select. Configures the SRC 3 modules to not use their own internally-generated sample rate ratio but use an externally-generated ratio. Used with TDM data. 0 = Matched-phase disabled (default) 1 = Matched-phase enabled
31	SRC3_ENABLE	SRC Enable. Enables SRC 3. 0 = Disabled 1 = Enabled

SRC Mute Register (SRCMUTE)

This read-write register, described in [Table A-44](#), connects an SRCx mute input and output (when cleared) to automatically mute input while the SRC is initializing. Bit 0 controls SRC0, bit 1 controls SRC1, bit 2 controls SRC2, and bit 3 controls SRC3. This register is located at address 0x2492. Note that the SRC mute interrupts are latched in the DAI_IRPTL_H and DAI_IRPTL_L registers.

Table A-44. SRCMUTE Register Bit Descriptions

Bit	Name	Description
0	SRC0_MUTE_EN	0 = Connect SRC0 mute input and output 1 = Do not connect SRC0 mute input and output
1	SRC1_MUTE_EN	0 = Connect SRC1 mute input and output 1 = Do not connect SRC1 mute input and output
2	SRC2_MUTE_EN	0 = Connect SRC2 mute input and output 1 = Do not connect SRC2 mute input and output
3	SRC3_MUTE_EN	0 = Connect SRC3 mute input and output 1 = Do not connect SRC3 mute input and output

SRC Ratio Registers (SRCRATx)

These read-only status registers (shown in [Figure A-40](#)) report the mute and I/O sample ratio as follows: the SRCRAT0 register reports for SRC0 and 1 and the SRCRAT1 register reports the mute and I/O sample ratio for SRC2 and 3.

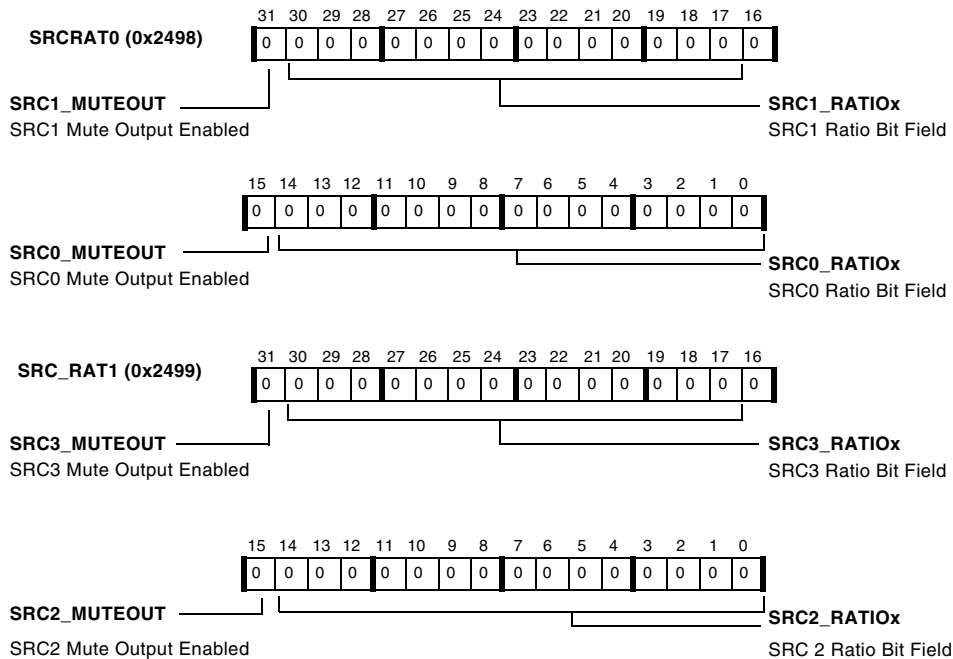


Figure A-40. SRC Ratio Register (SRCRAT0)

DAI/DPI Registers

The registers that are described in the following sections are contained within the digital audio and digital peripheral interfaces. The bits in these registers are used to enable the connection of peripherals and to view status of data transfers. For complete information on using the DAI/DPI, see [Chapter 4, Digital Audio/Digital Peripheral Interfaces](#).

Digital Audio Interface Status Register (DAI_STAT)

The DAI_STAT register is a read-only register and is shown in [Figure A-41](#) and described in [Table A-45](#). The state of all eight DMA channels is reflected in IDP_DMAx_STAT (bits 24-17 of the DAI_STAT register). These bits are set once the IDP_DMA_EN bit is set and remains set till the last data of that channel is transferred (see “[Input Data Port Control Register 0 \(IDP_CTL0\)](#)” on [page A-66](#)). Even if the IDP_DMA_EN bit is set, it goes low once the required number of data transfers occurs. And even when DMA through some channel is not intended, its IDP_DMAx_STAT bit goes high.

DAI/DPI Registers

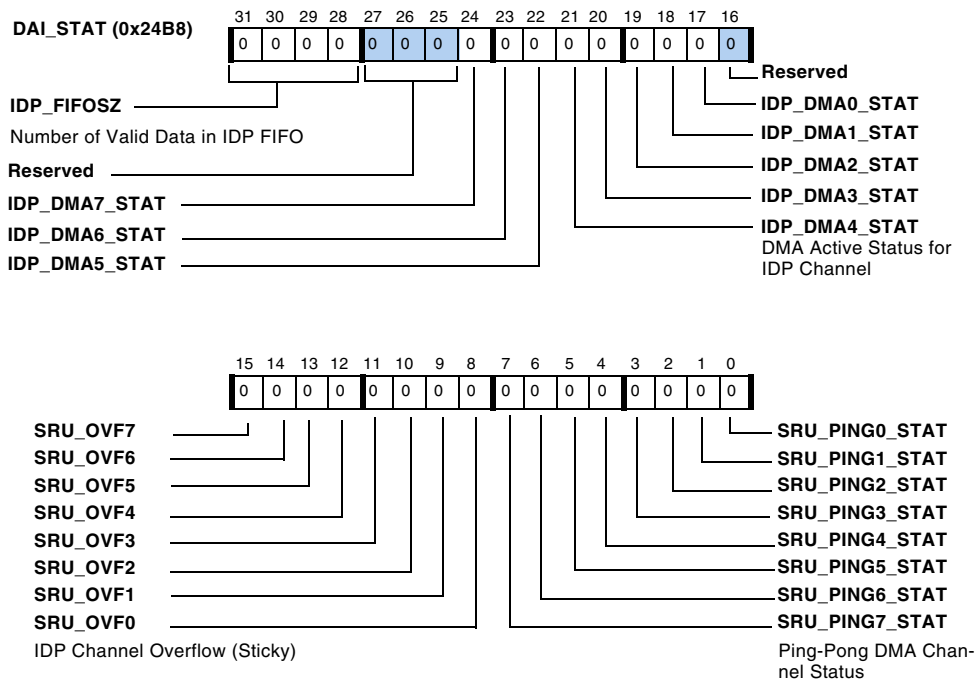


Figure A-41. DAI_STAT Register

Table A-45. DAI_STAT Register Bit Descriptions

Bit	Name	Description
7–0	SRU_PING _x _STAT	Ping-pong DMA Status (Channel). Indicates the status of ping-pong DMA in each respective channel (channel 0–7). 0 = DMA is not active 1 = DMA is active
15–8	SRU_OVF _x	Sticky Overflow Status (Channel). Provides overflow status information for each channel (bit 8 for channel 0 through bit 15 for channel 7). 0 = No overflow 1 = Overflow has occurred
16	Reserved	

Table A-45. DAI_STAT Register Bit Descriptions (Cont'd)

Bit	Name	Description
24–17	IDP_DMAx_STAT	Input Data Port DMA Status. 0 = DMA is not active 1 = DMA is active
27–25	Reserved	
31–28	IDP_FIFOSZ	IDP FIFO Size. Number of samples in the IDP FIFO.

DAI Resistor Pull-up Enable Register (DAI_PIN_PULLUP)

This 20-bit, read/write register is shown in [Figure A-42](#). Bits 19–0 of this register control the enabling/disabling 22.5 K Ω pull-up resistor on DAI_PO[19:0]. Setting a bit to 1 enables a pull-up resistor on the corresponding pin. After RESET, the value of this register is 0xFFFFF, which means pull-ups are enabled on all 20 DAI pins.

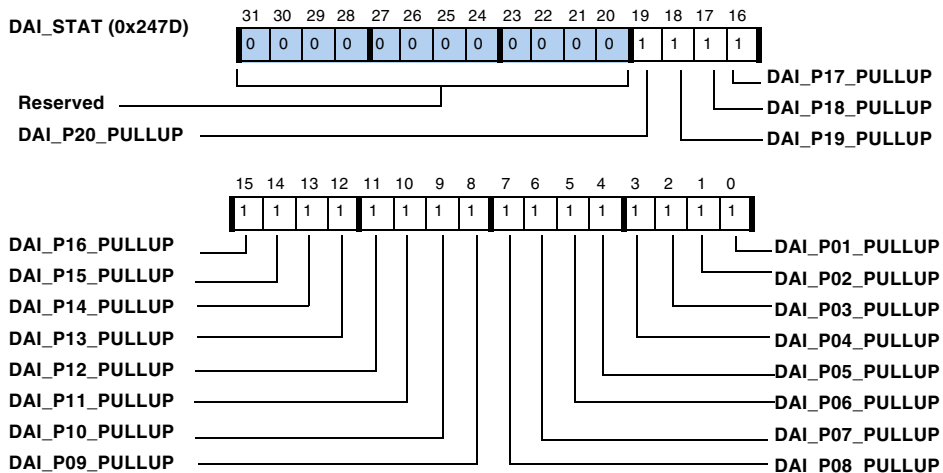


Figure A-42. DAI_PIN_PULLUP Register

DAI Pin Buffer Status Register (DAI_PIN_STAT)

This 20-bit, read-only register is shown in [Figure A-43](#). Bits 19–0 of this register indicate the status of `DAI_PB[20:1]`. Reads from bits 31–20 always return 0. This register is updated at one-half the core clock rate.

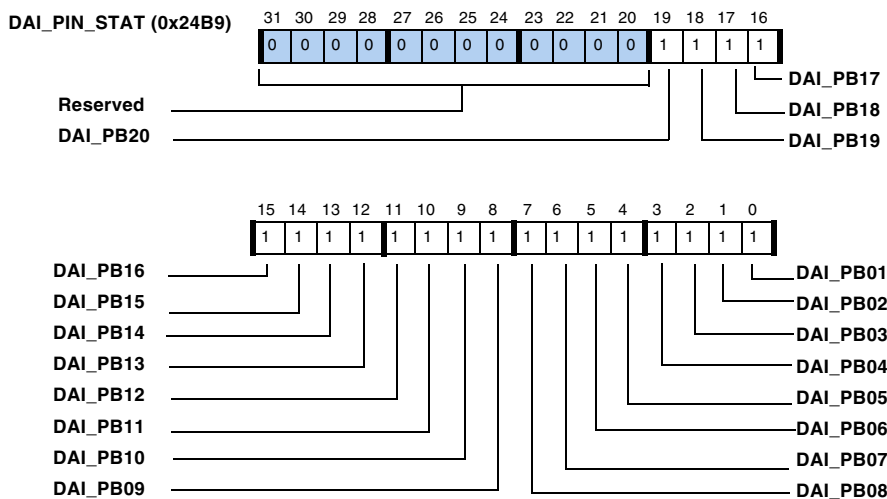


Figure A-43. DAI_PIN_STAT Register

DAI Interrupt Controller Registers

The DAI contains its own interrupt controller that indicates to the core when DAI audio peripheral related events have occurred. Since audio events generally occur infrequently relative to the SHARC core, the DAI interrupt controller reduces all of its interrupts onto two interrupt signals within the core's primary interrupt systems—one mapped with low priority and one mapped with high priority. This architecture allows programs to indicate priority broadly. In this way, the DAI interrupt controller registers provide 32 independently configurable interrupts labeled `DAI_INT[31:0]`, respectively.

The DAI interrupt controller is configured using three registers. Each of the 32 interrupt lines can be independently configured to trigger based on the incoming signal's rising edge, falling edge, both, or neither. Setting a bit in `DAI_IRPTL_RE` or `DAI_IRPTL_FE` enables that interrupt level on the rising and falling edges, respectively.

The 32 interrupt signals within the DAI are mapped to two interrupt signals in the primary interrupt controller of the SHARC core. The `DAI_IRPTL_PRI` register selects if the DAI interrupt is mapped to the high priority or low priority core interrupt (1=high priority, 0 =low priority).

The `DAI_IRPTL_H` register is a read-only register that has bits set for every DAI interrupt latched for the high priority core interrupt. The `DAI_IRPTL_L` register is a read-only register that has bits set for every DAI interrupt latched for the low priority core interrupt. When a DAI interrupt occurs, the low or high priority core ISR queries its corresponding register to determine which of the 32 interrupt sources requires service. When `DAI_IRPTL_H` is read, the high priority latched interrupts are cleared. When `DAI_IRPTL_L` is read, the low priority latched interrupts are cleared.

DMA overflow greater than N interrupts can be sensed only at rising edges. Falling edges are not used for these ten interrupts (eight DMA, one overflow, and one FIFO valid data greater than N).

The `IDP_FIFO_GTN_INT` interrupt is not cleared when the `DAI_IRPTL_H/L` register is read. This gets cleared when cause of this interrupt is zero.

A read resets the value to 0, except under the following condition:

The `IDP_FIFO_GTN_INT` interrupt is not cleared when the `DAI_IRPTL_H/L` registers are read. This register is cleared when the cause of this interrupt is zero.

All of the DAI interrupt registers are used primarily to provide the status of the resident interrupt controller. These registers are shown in [Figure A-44](#) described in [Table A-46](#). Note that for each of these registers the bit names and numbers are the same.

DAI/DPI Registers

DAI_IRPTL_RE(0x2480) DAI_IRPTL_FE(0x2481)
 DAI_IRPTL_PRI(0x2484) DAI_IRPTL_H(0x2488)
 DAI_IRPTL_L(0x2489) DAI_IRPTL_HS(0x248C)
 DAI_IRPTL_LS(0x248D)

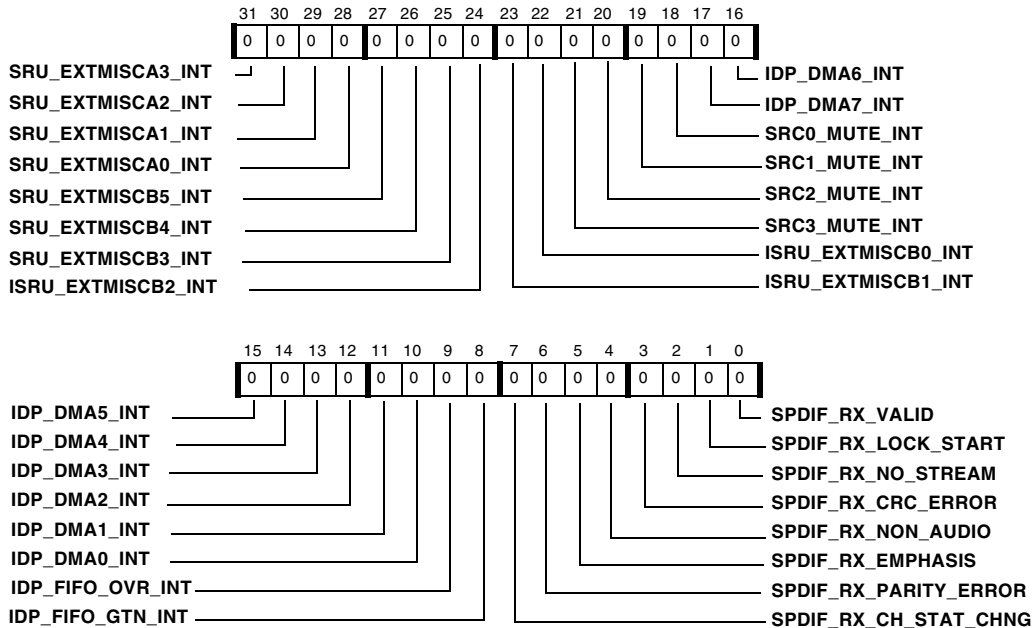


Figure A-44. DAI Interrupt Latch Register

Table A-46. DAI Interrupt Registers

Register	Description	Address
DAI_IRPTL_H DAI_IRPTL_HS	High Priority Interrupt Latch Register Shadow High Priority Interrupt Latch Register	0x2488 0x248C
DAI_IRPTL_L DAI_IRPTL_LS	Low Priority Interrupt Latch Register Shadow Low Priority Interrupt Latch Register	0x2489 0x248D
DAI_IRPTL_PRI	Core Interrupt Priority Assignment Register	0x2484

Table A-46. DAI Interrupt Registers (Cont'd)

Register	Description	Address
DAI_IRPTL_RE	Rising Edge Interrupt Mask Register	0x2481
DAI_IRPTL_FE	Falling Edge Interrupt Mask Register	0x2480

DPI Resistor Pull-up Enable Register (DPI_PIN_PULLUP)

This 16-bit read/write register is shown in [Figure A-45](#). Bits 13–0 of this register control the enabling/disabling 22.5 K Ω pull-up resistor on DPI_PO[13:0]. Setting a bit to 1 enables a pull-up resistor on the corresponding pin. After RESET, the value of this register is 0xFFFF, which means pull-ups are enabled on all 14 DPI pins.

DPI_PIN_PULLUP (0x1C30)

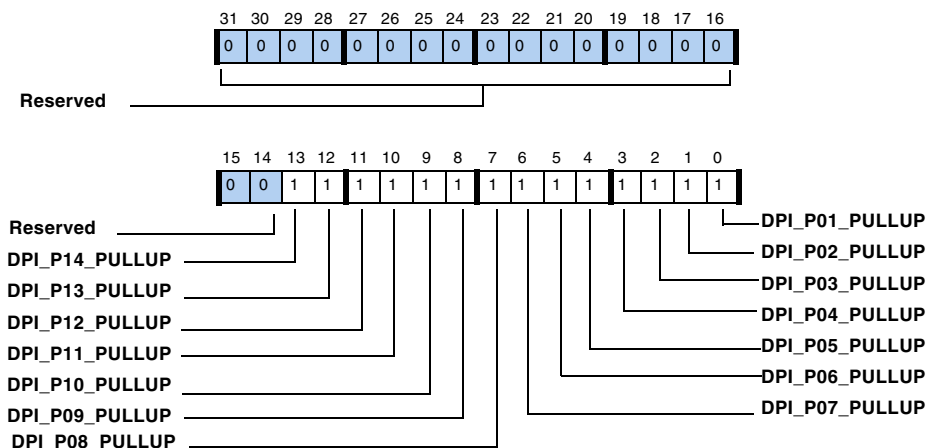


Figure A-45. DPI_PIN_PULLUP Register

DPI Pin Buffer Status Register (DPI_PIN_STAT)

This 16-bit, read-only register is shown in [Figure A-46](#). Bits 13–0 of this register indicate the status of `DPI_PB[14:1]`. Reads from bits 15–14 always return 0. This register is updated at one-half the core clock rate.

DPI_PIN_STAT (0x1C31)

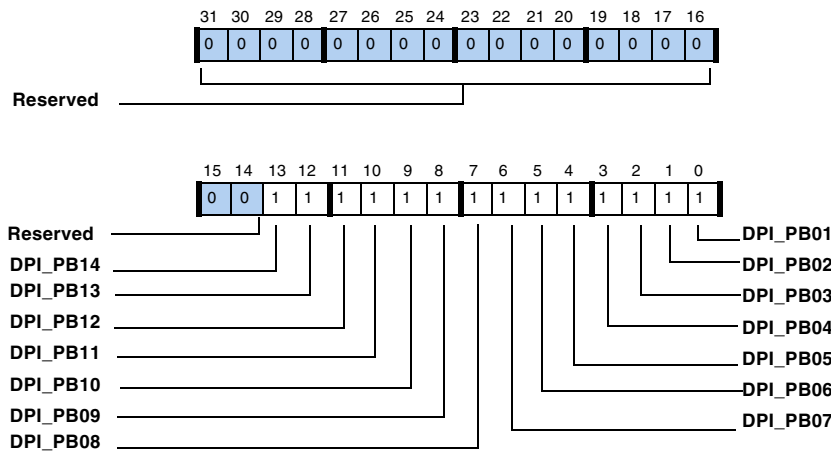


Figure A-46. DPI_PIN_STAT Register

DPI Interrupt Controller Registers

The digital peripheral interface (DPI) also has an interrupt controller, similar to that in the DAI. All of these interrupts are combined into a single interrupt, namely `DPI_INT`. The `DPI_IRPTL` register located at address `0x1C32`, contains the status on individual interrupts. Apart from the `DPI_IRPTL` register, there are two additional registers, `DPI_IRPTL_RE` and `DPI_IRPTL_FE` that are used for interrupt latching.

All of the DPI interrupt registers are used primarily to provide the status of the interrupt controller. These registers are shown in [Figure A-47](#) and listed in [Table A-47](#). Note that for each of these registers the bit names and numbers are the same.

Table A-47. DPI Interrupt Registers

Register	Description	Address
DPI_IRPTL_RE	Rising Edge Interrupt Mask Register	0x1C35
DPI_IRPTL_FE	Falling Edge Interrupt Mask Register	0x1C34
DPI_IRPTL_SH	DPI_IRPTL Shadow Register. Reads of this register returns data in DPI_IRPTL without clearing contents of the register.	0x1C33

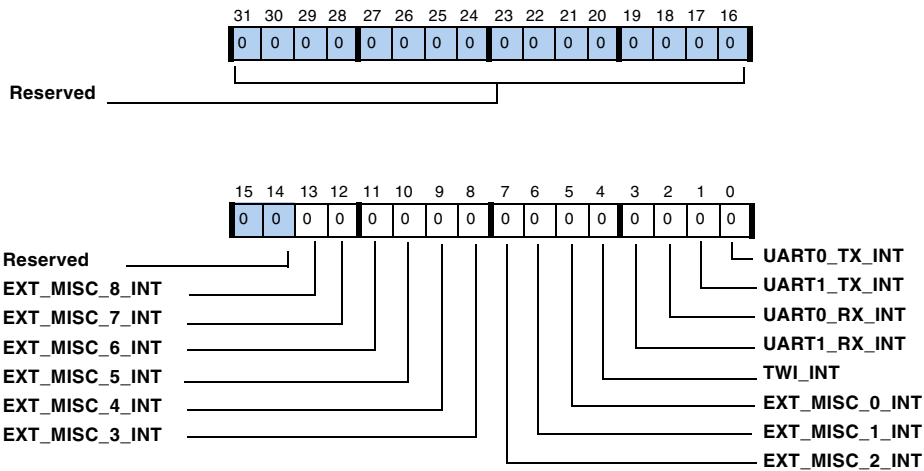


Figure A-47. DPI Interrupt Latch Register

UART Control and Status Registers

The processor provides a set of PC-style, industry-standard control and status registers for each UART. These memory-mapped registers (MMRs) are byte-wide registers that are mapped as half-words with the most significant byte zero-filled. Transmit and receive channels are buffered. The `UARTxTHR` register buffers the transmit shift register (`UARTxTSR`) and the `UARTxRBR` register buffers the receive shift register (`UARTxLSR`). The shift registers are not directly accessible by software.

Line Control Registers (UARTxLCR)

The UART line control registers (`UARTxLCR`, shown in [Figure A-48](#)) control the format of received and transmitted character frames. The `UARTSB` bit functions even when the UART clock is disabled. Since the transmit pin normally drives high, it can be used as a flag out pin, if the UART is not used. In 9-bit mode, the word length is always 8 and the 9th bit is transmitted instead of the parity bit.

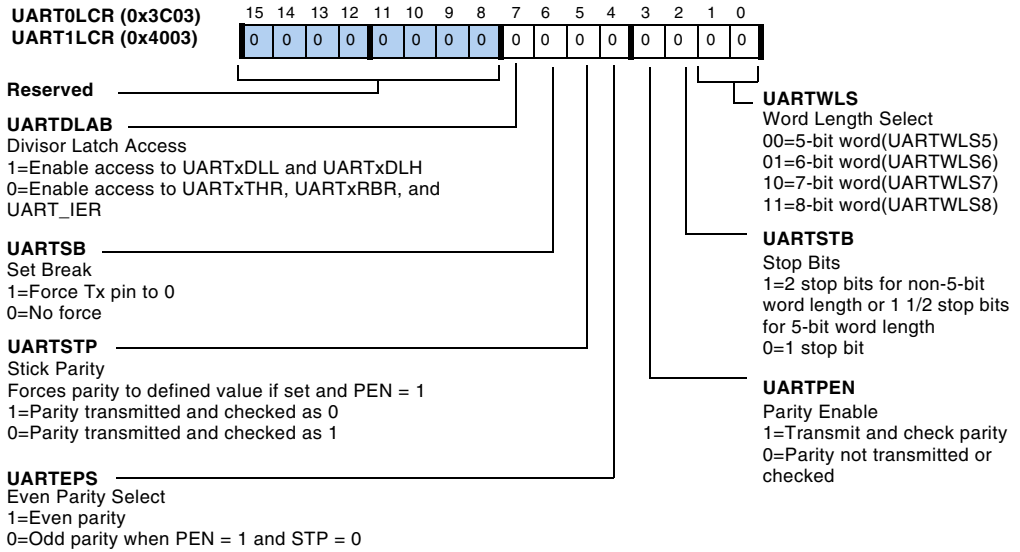


Figure A-48. UART Line Control Registers

Line Status Registers (UARTxLSR)

The UART line status registers (UARTxLSR) contain UART status information as shown in [Figure A-49](#). There are also shadow registers, UARTxLSRSH, with the following addresses: UART0LSRSH (0x3C0A) and UART1LSRSH (0x400A). These registers allow programs to read the contents of the corresponding main register without affecting the status the UART.

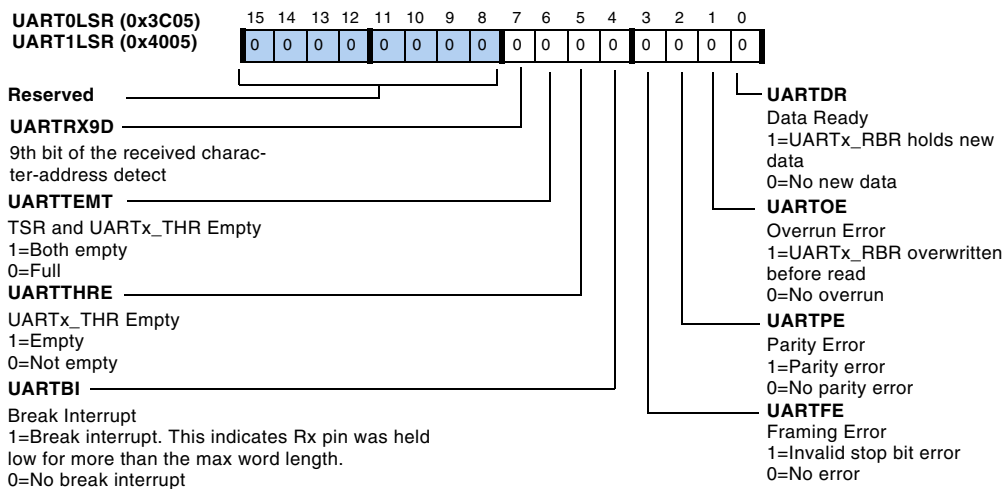


Figure A-49. UART Line Status Registers

Transmit Hold Registers (UARTxTHR)

In no pack mode (default), only the lower byte of these registers is used—all other bits are zero-filled. However in pack mode, both the high and low bytes are used. The TX9D and RX9D are the 9th bit in 9-bit transmission mode. These registers are mapped to the same address as the UARTxRBR and UARTxDLL registers. A write to the UART transmit holding registers (UARTxTHR) initiates the transmit operation.

To access UARTxTHR (shown in [Figure A-50](#)), the UARDDLAB bit in UARTxLCR must be cleared. When the UARDDLAB bit is cleared, writes to this address target the UARTxTHR registers, and reads from this address return the UARTxRBR registers.

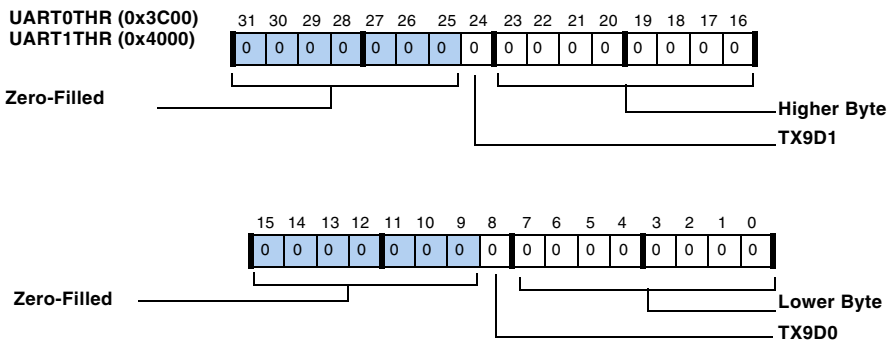


Figure A-50. UART Transmit Holding Registers (Packing Enabled)

Receive Buffer Registers (UARTxRBR)

These read-only registers (shown in [Figure A-51](#)) are mapped to the same address as the write-only UARTxTHR and DLL registers. To access UARTxRBR, the UARTDLAB bit in the UARTxLCR register must be cleared. When the UARTDLAB bit is cleared, writes to this address target the UARTxTHR registers, while reads from this address return the UARTxRBR registers.

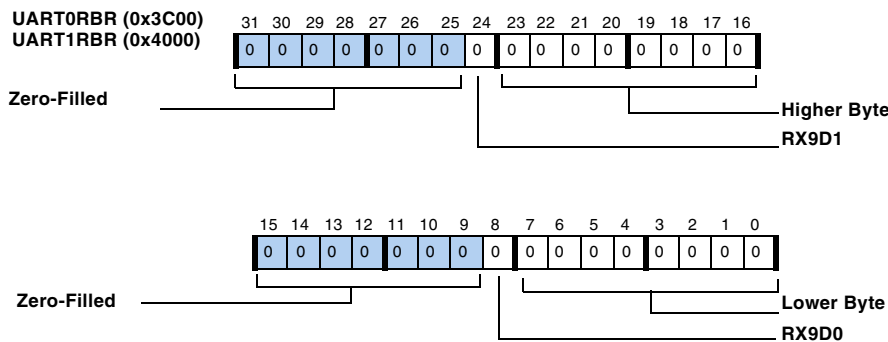


Figure A-51. UART Receive Buffer Registers

There are also shadow registers, UARTxRBRSH, with the following addresses: UART0RBRSH (0x3C08) and UART1RBRSH (0x4008). These registers allow programs to read the contents of the corresponding main register without affecting the status of the UART.

Interrupt Enable Registers (UARTxIER)

The UART interrupt enable registers (UARTxIER) are used to enable requests for system handling of empty or full states of UART data registers. Unless polling is used as a means of action, the UARTRBFIE and/or UARTTBEIE bits in these registers are normally set.

The UARTxIER register (shown in Figure A-52) is mapped to the same address as the UARTxDLH register. To access the UARTxIER register, the UARTDLAB bit in the UARTxLCR register must be cleared.

UART0IER (0x3C01)

UART1IER (0x4001)

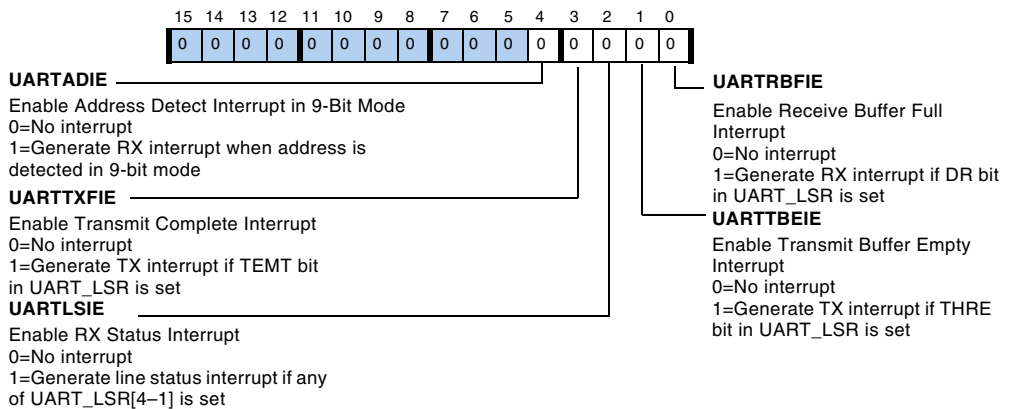


Figure A-52. UART Interrupt Enable Register

Interrupt Identification Registers (UARTxIIR)

For legacy reasons, the UART interrupt identification registers (UARTxIIR, shown in [Figure A-53](#)) still reflect the UART interrupt status. Legacy operation may require bundling all UART interrupt sources to a single interrupt channel and servicing them all by the same software routine. This can be established by globally assigning all UART interrupts to the same interrupt priority. For more information, see “Peripheral Interrupt Priority Control Registers” on page A-164.

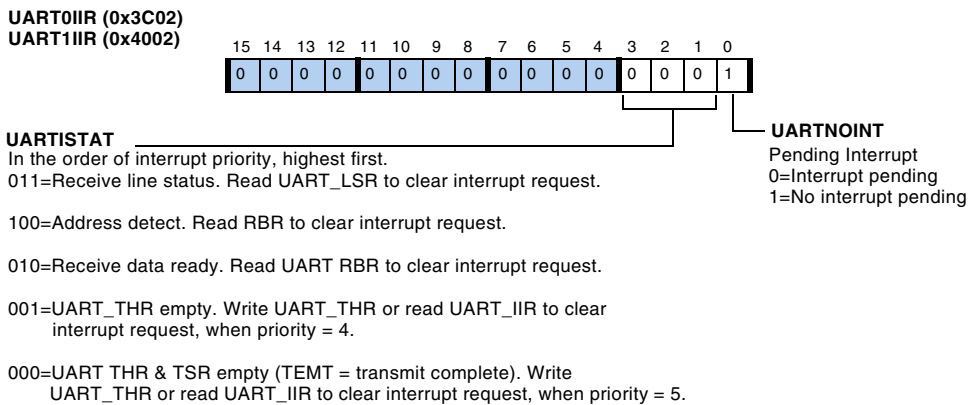


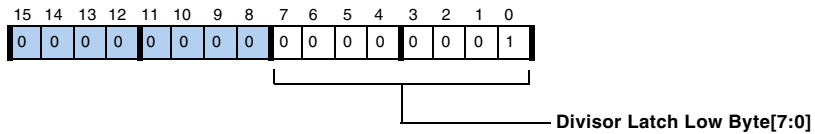
Figure A-53. UART Interrupt Identification Register

There are also shadow registers, UARTxIIRSH, with the following addresses: UART0IIRSH (0x3C09) and UART1IIRSH (0x4009). These registers allow programs to read the contents of the corresponding main register without affecting the status of the UART.

Divisor Latch Registers (UARTxDLL, UARTxDLH)

The bit rate is characterized by the system clock (SCLK) and the 16-bit divisor. The divisor is split into the UART divisor latch low byte register (UARTxDLL) and the UART divisor latch high byte register (UARTxDLH), both shown in [Figure A-54](#).

UART0DLL (0x3C00)
UART1DLL (0x4000)



UART0DLL (0x3C01)
UART1DLL (0x4001)

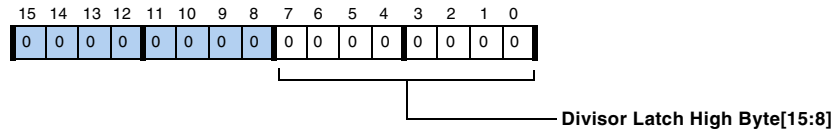


Figure A-54. UART Divisor Latch Registers

The UARTxDLL registers are mapped to the same address as the UARTxTHR and UARTxRBR registers. The UARTxDLH registers are mapped to the same address as the interrupt enable registers (UARTxIER). The UARTDLAB bit in the UARTxLCR register must be set before the UART divisor latch registers can be accessed.

Scratch Registers (UARTxSCR)

The contents of the 8-bit UART scratch registers (UARTxSCR shown in [Figure A-55](#)) is reset to 0x00. It is used for general-purpose data storage and does not control the UART hardware in any way.

UART0SCR (0x3C07)
UART1SCR (0x4007)

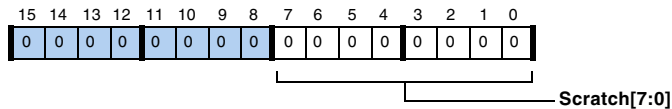


Figure A-55. UART Scratch Registers

Mode Registers (UARTxMODE)

The UART mode registers control miscellaneous settings as shown in [Figure A-56](#) and described [Table A-48](#).

UART0MODE (0x3C04)
UART1MODE (0x4004)

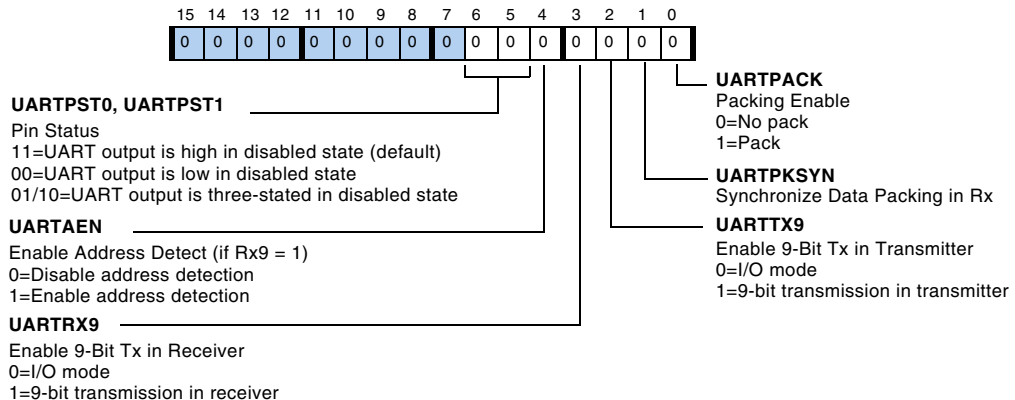


Figure A-56. UART Mode Registers

Table A-48. UART Mode Register Bit Descriptions

Bit	Name	Description
0	UARTPACK	Packing Enable. 0 = No pack 1 = Packing enabled. Consecutive data words (example 0xAB and 0xCD) are packed as 0x00CD 00AB in the receiver, and 0x00CD 00AB is transmitted as two words of 0xAB and 0xCD successively from the transmitter. For more information, see “Packing Mode” on page 11-15.
1	UARTPKSYN	Synchronize Data Packing in RX. When written with a 1, the next data byte goes to the lower byte position of the RBR register. This is a write-only bit and always returns zero on reads.
2	UARTTX9	Enable 9-Bit Tx in Transmitter. 0 = I/O mode 1 = 9-bit transmission in transmitter
3	UARTRX9	Enable 9-Bit Tx in Receiver. 0 = I/O mode 1 = 9-bit transmission in receiver
4	UARTAEN	Enable Address Detect (If RX9 = 1). 0 = Disable address detection; all bytes are received 1 = Enable address detection; interrupt and load of RBR when RX9D is set
6–5	UARTPST1, UARTPST0	Pin Status. 11 = UART output is high in disabled state (default) 00 = UART output is low in disabled state 01/10 = UART output is three-stated in the disabled state

UART DMA Registers

The UART supports full-duplex DMA. Separate receive and transmit DMA channels are responsible for moving data between the peripheral and memory in this mode. The UART uses eight registers to control and provide status on DMA. The following sections provide descriptions of these registers.

UART Control and Status Registers

DMA Control Registers (UARTxTXCTL, UARTxRXCTL)

Use these registers (described in [Table A-49](#) and [Table A-50](#)) to enable DMA, DMA chaining, and to clear the transmit and receive buffers. The transmit and receive registers are read-write registers and their addresses are:

UART0TXCTL – 0x3F04

UART1TXCTL – 0x4304

UART0RXCTL – 0x3E04

UART1RXCTL – 0x4204

Table A-49. UARTxTXCTL Register Descriptions

Bit	Name	Description
0	UARTEN	DMA Transmit Buffer Enable. When set (=1), enables the transmit buffer. When cleared, clears the transmit buffer.
1	UARTDEN	DMA Enable. When set (=1), enables DMA on the specified channel. When cleared, disables DMA.
2	UARTCHEN	Chain Pointer DMA Enable. When set (=1), enables chain pointer DMA on the specified channel. When cleared, disables chained DMA.

Table A-50. UARTxRXCTL Register Descriptions

Bit	Name	Description
0	UARTEN	DMA Receive Buffer Enable. When set (=1), enables the receive buffer. When cleared, clears the receive buffer.
1	UARTDEN	DMA Enable. When set (=1), enables DMA on the specified channel. When cleared, disables DMA.
2	UARTCHEN	Chain Pointer DMA Enable. When set (=1), enables chain pointer DMA on the specified channel. When cleared, disables chained DMA.

DMA Status Registers (UARTxTXSTAT, UARTxRXSTAT)

These read-only registers (described in [Table A-51](#) and [Table A-52](#)) provide DMA status information and their addresses are:

UART0TXSTAT – 0x3F05

UART1TXSTAT – 0x4305

UART0RXSTAT – 0x3E05

UART1RXSTAT – 0x4205

Table A-51. UARTxTXSTAT Register Bit Descriptions

Bit	Name	Description
0	Reserved	
1	UARTDMASTAT	DMA Status. Provides DMA status. 0 = TX DMA is inactive 1 = TX DMA is active
2	UARTCHSTAT	DMA Chaining Status. Provides DMA chaining status. 0 = TX DMA chain loading is inactive 1 = TX DMA chain loading is active

Table A-52. UARTxRXSTAT Register Bit Descriptions

Bit	Name	Description
0	UARTERRIRQ	Receive Channel Error Interrupt. 0 = No error interrupt 1 = Error interrupt generated due to receive error (parity/overflow/framing). This bit is cleared on a read of the LSR register.
1	UARTDMASTAT	DMA Status. Provides DMA status. 0 = RX DMA is inactive 1 = RX DMA is active
2	UARTCHSTAT	DMA Chaining Status. Provides DMA chaining status. 0 = RX DMA chain loading is inactive 1 = RX DMA chain loading is active

Two Wire Interface Registers



Apart from the DMA control and status registers there are index, modifier, count, and chain pointer registers for both the transmit and receive DMA channels. For more information on these registers, see [“Port, Buffer, and DMA Control Registers” on page 2-26](#), and [Table 2-13, “UART DMA Registers,” on page 2-44](#).

Two Wire Interface Registers

The two wire interface (TWI) registers (described in [Table A-53](#)) provide all control and status bits for this peripheral. Status bits can be updated by their respective functional blocks.

Table A-53. TWI Register Descriptions

Address	Name	Description
0x4400	TWIDIV	SCL Clock Divider
0x4404	TWIMITR	Master Internal Time Reference
0x4408	TWISCTL	Slave Mode Control
0x440C	TWISSTAT	Slave Mode Status
0x4410	TWISADDR	Slave Mode Address
0x4414	TWIMCTL	Master Mode Control
0x4418	TWIMSTAT	Master Mode Status
0x441C	TWIMADDR	Master Mode Address
0x4420	TWIIRPTL	Interrupt Latch
0x4424	TWIIMASK	Interrupt Mask
0x4428	TWIFIFOCTL	FIFO Control
0x442C	TWIFIFOSTAT	FIFO Status
0x4480	TXTWI 8	8-bit FIFO Transmit Register
0x4484	TXTWI 16	16-bit FIFO Transmit Register

Table A-53. TWI Register Descriptions (Cont'd)

Address	Name	Description
0x4488	RXTWI8	8-Bit FIFO Receive Register
0x448C	RXTWI16	16-Bit FIFO Receive Register

Master Internal Time Register (TWIMITR)

The TWI control register (TWIMITR, shown in [Figure A-57](#) and described in [Table A-54](#)) is used to enable the TWI module as well as to establish a relationship between the peripheral clock (PCLK) and the TWI controller's internally-timed events. The internal time reference is derived from PCLK using the prescaled value:

$$\text{PRESCALE} = f_{\text{PCLK}} / 10 \text{ MHz}$$

TWIMITR (0x4404)

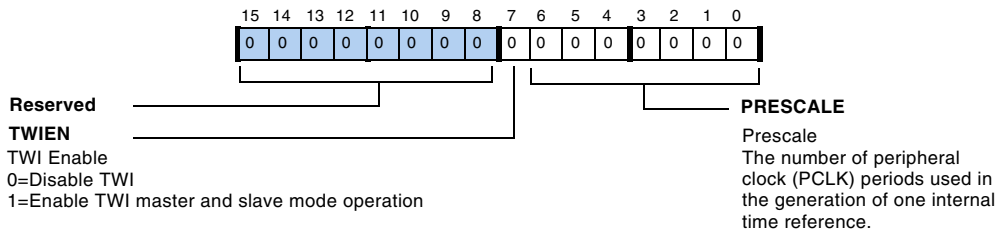


Figure A-57. Master Internal Time Register

Two Wire Interface Registers

Table A-54. Master Internal Time Register Bit Descriptions

Bit	Name	Description
0–6	PRESCALE	Prescale. The number of peripheral clock (PCLK) periods used in the generation of one internal time reference. The value of PRESCALE must be set to create an internal time reference with a period of 10 MHz. This is represented as a 7-bit binary value.
7	TWIEN	TWI Enable. This bit must be set for slave or master mode operation. It is recommended that this bit be set at the time PRESCALE is initialized and remain set. This guarantees accurate operation of bus busy detection logic. 0 = Disable TWI 1 = Enable TWI master and slave mode operation.

Clock Divider Register (TWIDIV)

During master mode operation, the SCL clock divider register (TWIDIV shown in [Figure A-58](#) and described in [Table A-55](#)) values are used to create the high and low durations of the serial clock (SCL). Serial clock frequencies can vary from 400 KHz to less than 20 KHz. The resolution of the clock generated is 1/10 MHz or 100 ns.

TWIDIV (0x4400)

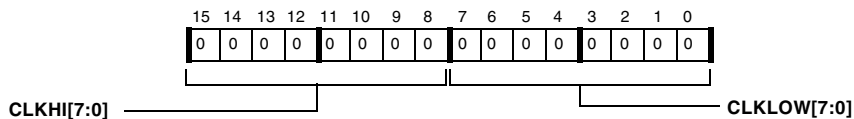


Figure A-58. Clock Divider Register

Table A-55. Clock Divider Register Bit Descriptions

Bit	Name	Description
7–0	CLKLOW	Clock Low. Number of internal time reference periods the serial clock (SCL) is held low. Represented as an 8-bit binary value.
15–8	CLKHI	Clock High. Number of internal time reference periods the serial clock (SCL) waits before a new clock low period begins (assuming a single master). Represented as an 8-bit binary value.

Slave Mode Control Register (TWISCTL)

The TWI slave mode control register (TWISCTL) shown in [Figure A-59](#) and described in [Table A-56](#), controls the logic associated with slave mode operation. Settings in this register do not affect master mode operation and should not be modified to control master mode functionality.

TWISCTL (0x4408)

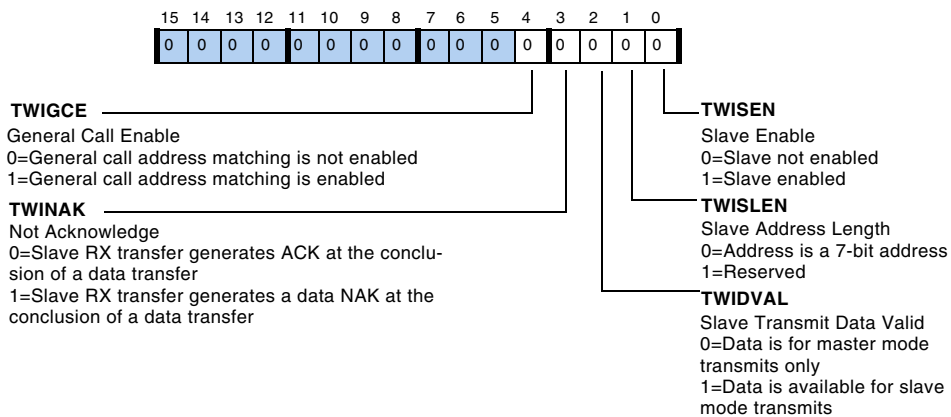


Figure A-59. Slave Mode Control Register

Two Wire Interface Registers

Table A-56. Slave Mode Control Register Bit Descriptions

Bit	Name	Description
0	TWISEN	Slave Enable. 0 = The slave is not enabled. No attempt is made to identify a valid address. If cleared during a valid transfer, clock stretching ceases, the serial data line is released and the current byte is not acknowledged. 1 = The slave is enabled. Enabling slave and master modes of operation concurrently is allowed.
1	TWISLEN	Slave Address Length. 0 = Address is a 7-bit address 1 = Reserved. Setting this bit to 1 causes unpredictable behavior.
2	TWIDVAL	Slave Transmit Data Valid. 0 = Data in the transmit FIFO is for master mode transmits and is not allowed to be used during a slave transmit, and the transmit FIFO is treated as if it is empty. 1 = Data in the transmit FIFO is available for a slave transmission.
3	TWINAK	Not Acknowledged. 0 = Slave receive transfer generates an ACK at the conclusion of a data transfer. 1 = Slave receive transfer generates a data NAK at the conclusion of a data transfer. The slave is still considered to be addressed.
4	TWIGCE	General Call Enable. General call address detection is available only when slave mode is enabled. 0 = General call address matching is not enabled 1 = General call address matching is enabled. Regardless of the selected address length of slave address, a general call slave receive transfer is accepted. All status and interrupt source bits associated with transfers are updated.

Slave Address Register (TWISADDR)

The TWI slave mode address register (TWISADDR, shown in [Figure A-60](#)) holds the slave mode address, which is the valid address that the slave-enabled TWI controller responds to. The TWI controller compares this value with the received address during the addressing phase of a transfer.

TWISADDR (0x4410)

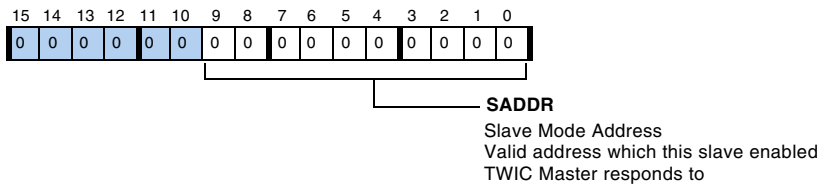


Figure A-60. Slave Mode Address Register

Slave Status Register (TWISSTAT)

During and at the conclusion of slave mode transfers, the TWI slave mode status register (TWISSTAT, shown in [Figure A-61](#)) holds information on the current transfer. Generally slave mode status bits are not associated with the generation of interrupts. Master mode operation does not affect the slave mode status bits.

TWISSTAT (0x440C)

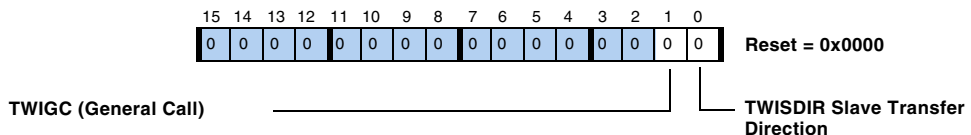


Figure A-61. Slave Mode Status Register

Master Control Register (TWIMCTL)

The TWI master mode control register (TWIMCTL, shown in [Figure A-62](#) and described in [Table A-57](#)) controls the logic associated with master mode operation. Bits in this register do not affect slave mode operation and should not be modified to control slave mode functionality.

TWIMCTL (0x4414)

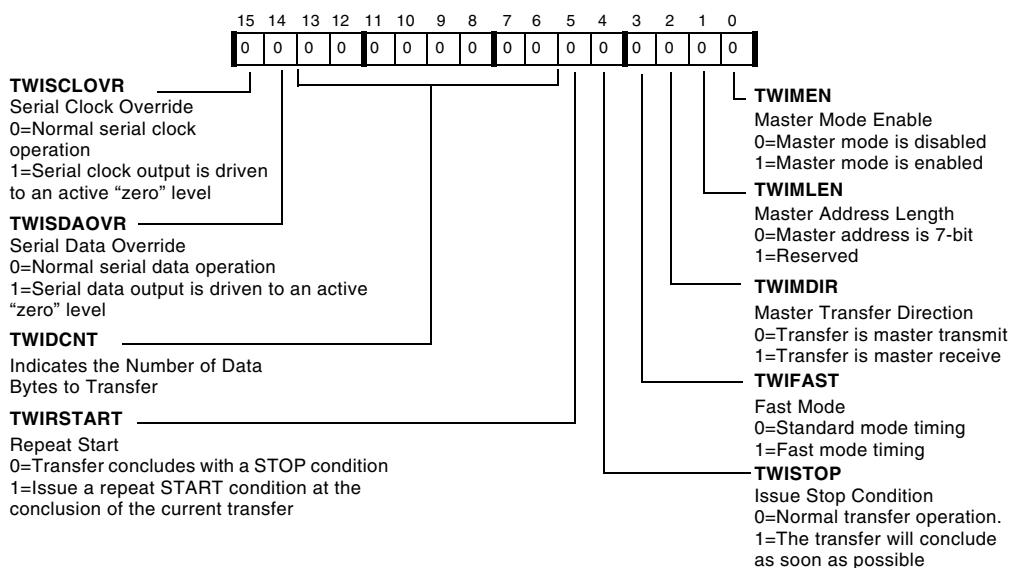


Figure A-62. Master Mode Control Register

Table A-57. Master Control Register Bit Descriptions

Bit	Name	Description
0	TWIMEN	Master Mode Enable. Clears itself at the completion of a transfer. This includes transfers terminated due to errors. 0 = Master mode functionality is disabled. If MEN is cleared during operation, the transfer is aborted and all logic associated with master mode transfers are reset. Serial data and serial clock (SDA, SCL) are no longer driven. Write 1-to-clear status bits are not effected. 1 = Master mode functionality is enabled. A START condition is generated if the bus is idle.
1	TWIMLEN	Master Address Length. 0 = Address is 7-bit 1 = Reserved. Setting this bit to one causes unpredictable behavior
2	TWIMDIR	Master Transfer Direction. 0 = The initiated transfer is master transmit 1 = The initiated transfer is master receive
3	TWIFAST	Fast Mode. 0 = Standard mode timing specifications in use 1 = Fast mode timing specifications in use
4	TWISTOP	Issue STOP Condition. 0 = Normal transfer operation 1 = The transfer concludes as soon as possible avoiding any error conditions (as if data transfer count had been reached) and at that time the interrupt source register is updated along with any associated status bits.
5	TWIRSTART	Repeat START. 0 = Transfer concludes with a STOP condition 1 = Issue a repeat START condition at the conclusion of the current transfer (DCNT = 0) and begin the next transfer. The current transfer is concluded with updates to the appropriate status and interrupt bits. If errors occurred during the previous transfer, a repeat START does not occur. In the absence of any errors, master enable (MEN) does not clear itself on a repeat start.

Two Wire Interface Registers

Table A-57. Master Control Register Bit Descriptions (Cont'd)

Bit	Name	Description
13–6	TWIDCNT	Data Transfer Count. Indicates the number of data bytes to transfer. As each data word is transferred, the data transfer count is decremented. When DCNT is zero, a STOP (or restart condition) is issued. Setting DCNT to 0xFF disables the counter. In this transfer mode, data continues to be transferred until it is concluded by setting the STOP bit.
14	TWISDAOVR	Serial Data (SDA) Override. For use when direct control of the Serial Data line is required. Normal master and slave mode operation should not require override operation. 0 = Normal serial data operation under the control of the transmit shift register and acknowledge logic 1 = Serial data output is driven to an active “zero” level, overriding all other logic. This state is held until the bit location is cleared.
15	TWISCLOVR	Serial Clock (SCL) Override. For use when direct control of the serial clock line is required. Normal master and slave mode operation should not require override operation. 0 = Normal serial clock operation under the control of master mode clock generation and slave mode clock stretching logic 1 = Serial clock output is driven to an active “zero” level, overriding all other logic. This state is held until the bit location is cleared.

Master Address Register (TWIMADDR)

During the addressing phase of a transfer, the TWI controller, with its master enabled, transmits the contents of the TWI master mode address register (TWIMADDR, shown in [Figure A-63](#)). When programming this register, omit the read/write bit. That is, only the upper 7 bits that make up the slave address should be written to this register. For example, if the slave address is 1010000X, then TWIMADDR is programmed with 1010000, which corresponds to 0x50. When sending out the address on the bus, the TWI controller appends the read/write bit as appropriate, based on the state of the MDIR bit in the master mode control register.

TWIMADDR(0x441C)

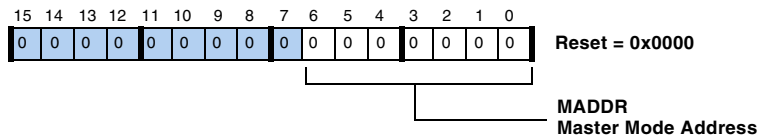


Figure A-63. Master Mode Address Register

Master Status Register (TWIMSTAT)

The TWI master mode status register (TWIMSTAT, shown in [Figure A-64](#) and described in [Table A-58](#)) holds information during master mode transfers and at their conclusions. Generally, master mode status bits are not directly associated with the generation of interrupts but offer information on the current transfer. Slave mode operation does not affect master mode status bits. This is a read-only register.

TWIMSTAT (0x4418)

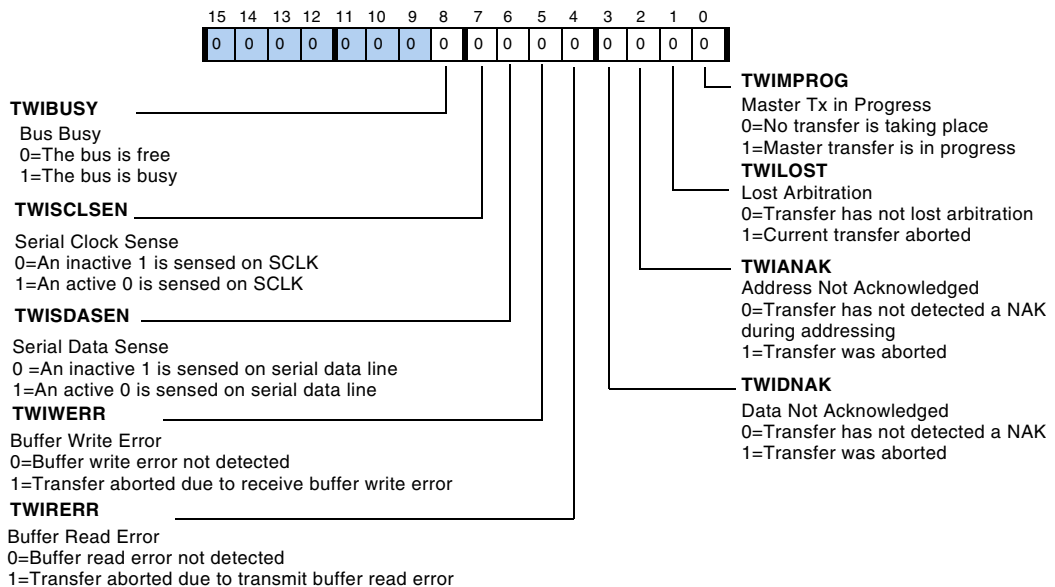


Figure A-64. Master Mode Status Register

Table A-58. Master Status Register Bit Descriptions

Bit	Name	Description
0	TWIMPROG	Master Transfer In Progress. 0 = Currently no transfer is taking place. This can occur once a transfer is complete or while an enabled master is waiting for an idle bus. 1 = A master transfer is in progress.
1	TWILOST	Lost Arbitration. 0 = The current transfer has not lost arbitration with another master. 1 = The current transfer was aborted due to the loss of arbitration with another master. This bit is cleared by writing a 1 to its bit location.
2	TWIANAK	Address Not Acknowledged. 0 = The current master transfer has not detected a NAK during addressing. 1 = The current master transfer was aborted due to the detection of a NAK during the address phase of the transfer. This bit is cleared by writing a 1 to its bit location.
3	TWIDNAK	Data Not Acknowledged. 0 = The current master transfer has not detected a NAK during data transmission. 1 = The current master transfer was aborted due to the detection of a NAK during data transmission. This bit is cleared by writing a 1 to its bit location.
4	TWIRERR	Buffer Read Error. 0 = The current master transmit has not detected a buffer read error. 1 = The current master transfer was aborted due to a transmit buffer read error. At the time data was required by the transmit shift register, the buffer was empty. This bit is cleared by writing a 1 to its bit location.

Two Wire Interface Registers

Table A-58. Master Status Register Bit Descriptions (Cont'd)

Bit	Name	Description
5	TWIWERR	Buffer Write Error. 0 = The current master receive has not detected a receive buffer write error. 1 = The current master transfer was aborted due to a receive buffer write error. The receive buffer and receive shift register were both full at the same time. This bit is cleared by writing a one to its bit location
6	TWISDASEN	Serial Data Sense. For use when direct sensing of the serial data line is required. The register value is delayed due to the input filter (nominally 50 ns). Normal master and slave mode operation should not require this feature. 0 = An inactive “one” is currently being sensed on serial data line. 1 = An active “zero” is currently being sensed on serial data line. The source of the active driver is not known and can be internal or external.
7	TWISCLSEN	Serial Clock Sense. For use when direct sensing of the serial clock line is required. The register value is delayed due to the input filter (nominally 50 ns). Normal master and slave mode operation should not require this feature. 0 = An inactive “one” is currently being sensed on SCLK. 1 = An active “zero” is currently being sensed on SCLK. The source of the active driver is not known and can be internal or external.
8	TWIBUSY	Bus Busy. Indicates whether the bus is currently busy or free. This indication applies to all devices. Upon a START condition, setting the register value is delayed due to the input filtering. Upon a STOP condition, clearing the register value occurs after time t_{BUF} . 0 = The bus is free. The clock and data bus signals have been inactive for the appropriate bus free time. 1 = The bus is busy. Clock and/or data activity has been detected.

FIFO Control Register (TWIFIFOCTL)

The TWI FIFO control register (TWIFIFOCTL, shown in [Figure A-65](#) and described in [Table A-59](#)) affects only the FIFO and is not tied in any way with master or slave mode operation.

TWIFIFOCTL (0x4428)

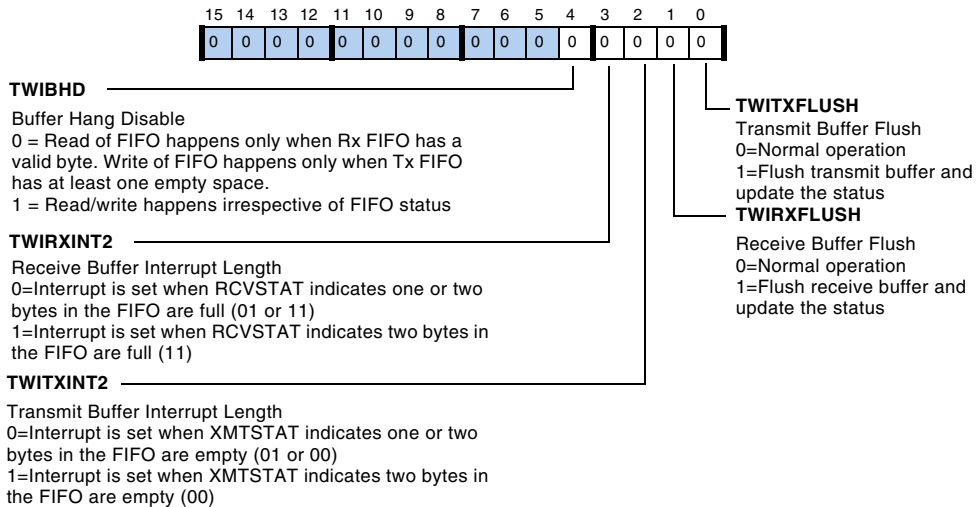


Figure A-65. FIFO Control Register

Two Wire Interface Registers

Table A-59. TWIFIFOCTL Register Bit Descriptions

Bit	Name	Description
0	TWITXFLUSH	Transmit Buffer Flush. 0 = Normal operation of the transmit buffer and its status bits 1 = Flush the contents of the transmit buffer and update the status to indicate the buffer is empty. This state is held until this bit is cleared. During an active transmit, the transmit buffer in this state responds as if the transmit buffer is empty.
1	TWIRXFLUSH	Receive Buffer Flush. 0 = Normal operation of the receive buffer and its status bits. 1 = Flush the contents of the receive buffer and update the status to indicate the buffer is empty. This state is held until this bit is cleared. During an active receive the receive buffer in this state responds to the receive logic as if it is full.
2	TWITXINT2	Transmit Buffer Interrupt Length. Determines the rate at which transmit buffer interrupts are generated. Interrupts may be generated with each byte transmitted or after two bytes are transmitted. 0 = An interrupt (TWITXINT) is set when TWITXS indicates one or two bytes in the FIFO are empty (01 or 00). 1 = An interrupt (TWITXINT) is set when TWITXS indicates two bytes in the FIFO are empty (00).
3	TWIRXINT2	Receive Buffer Interrupt Length. Determines the rate at which receive buffer interrupts are generated. Interrupts may be generated with each byte received or after two bytes are received. 0 = An interrupt (TWIRXINT) is set when TWIRXS indicates one or two bytes in the FIFO are full (01 or 11). 1 = An interrupt (TWIRXINT) is set when TWIRXS indicates two bytes in the FIFO are full (11).
4	TWIBHD	Receive Buffer Hang Disable. 0 = Read of FIFO happens only when Rx FIFO has a valid byte. Write of FIFO happens only when Tx FIFO has at least one empty space. 1 = Read/write happens irrespective of FIFO status

FIFO Status Register (TWIFFOSTAT)

The fields in the TWI FIFO status register (TWIFFOSTAT, shown in [Figure A-66](#) and described in [Table A-60](#)) indicate the state of the FIFO buffers' receive and transmit contents. The FIFO buffers do not discriminate between master data and slave data. By using the status and control bits provided, the FIFO can be managed to allow simultaneous master and slave operation. All bits in this register are read-only.

TWIFFOSTAT (0x442C)

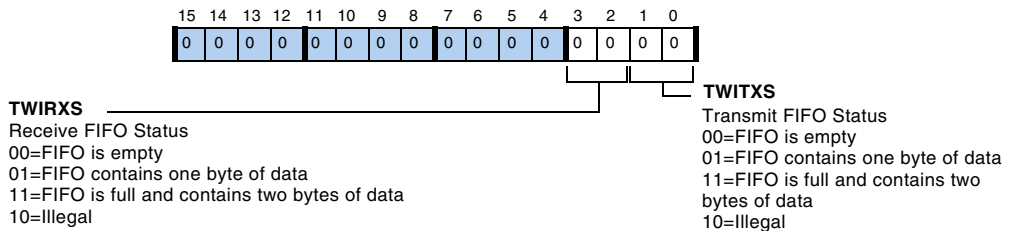


Figure A-66. FIFO Status Register

Two Wire Interface Registers

Table A-60. FIFO Status Register Bit Descriptions

Bit	Name	Description
1–0	TWITXS	Transfer FIFO Status. These read-only bits indicate the number of valid data bytes in the FIFO buffer. The status is updated with each FIFO buffer write using the peripheral data bus or read access by the transmit shift register. Simultaneous accesses are allowed. 00 = FIFO is empty. Either a single- or double-byte peripheral write of the FIFO goes through immediately. 01 = FIFO contains one byte of data. A single byte peripheral write of the FIFO goes through immediately. A double-byte peripheral write waits until the FIFO is empty 11 = FIFO is full and contains two bytes of data. 10 = Illegal
3–2	TWIRXS	Receive FIFO Status. These read-only bits indicate the number of valid data bytes in the receive FIFO buffer. The status is updated with each FIFO buffer read using the peripheral data bus or write access by the receive shift register. Simultaneous accesses are allowed. 00 = FIFO is empty 01 = FIFO contains one byte of data. A single-byte peripheral read of the FIFO goes through immediately. A double-byte peripheral read waits until the FIFO is full. 11 = FIFO is full and contains two bytes of data. Either a single- or double-byte peripheral read of the FIFO is allowed. 10 = Illegal

Interrupt Source Register (TWIIRPTL)

The TWI interrupt source register (TWIIRPTL, shown in [Figure A-67](#) and described in [Table A-61](#)) contains information about functional areas requiring servicing. Many of the bits serve as an indicator to further read and service various status registers. After servicing the interrupt source associated with a bit, the user must clear that interrupt source bit. All bits are sticky and W1C.

TWIIRPTL (0x4420)

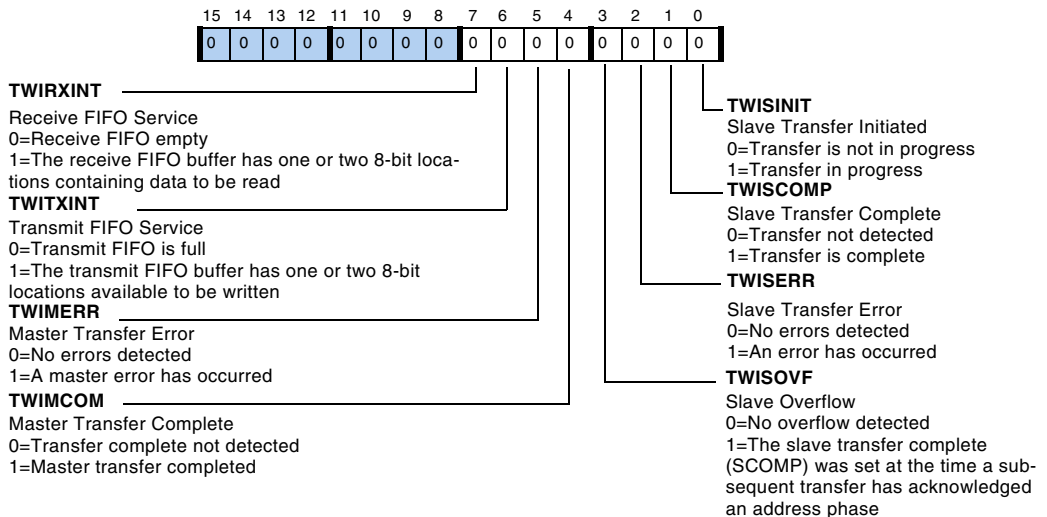


Figure A-67. Interrupt Source Register

Two Wire Interface Registers

Table A-61. Interrupt Source Register Bit Descriptions

Bit	Name	Description
0	TWISINIT	Slave Transfer Initiated. 0 = A transfer is not in progress. An address match has not occurred since the last time this bit was cleared. 1 = The slave has detected an address match and a transfer has been initiated. This bit is sticky and is cleared by writing 1 to its bit location.
1	TWISCOMP	Slave Transfer Complete. 0 = The completion of a transfer not detected 1 = The transfer is complete and either a stop, or a restart was detected. This bit is sticky and is cleared by writing a one to its bit location
2	TWISERR	Slave Transfer Error. 0 = No errors detected. 1 = An error has occurred. A restart or stop condition has occurred during the data receive phase of a transfer. This bit is sticky and is cleared by writing a one to its bit location.
3	TWISOVF	Slave Over Flow. 0 = No overflow detected. 1 = The slave transfer complete (TWISCOMP) was set at the time a subsequent transfer has acknowledged an address phase. The transfer continues, however, it may be difficult to delineate data of one transfer from another. This bit is sticky and is cleared by writing a one to its bit location.
4	TWIMCOM	Master Transfer Complete. 0 = The completion of a transfer not detected. 1 = The initiated master transfer is complete. In the absence of a repeat start, the bus is released. This bit is sticky and is cleared by writing a 1 to its bit location.
5	TWIMERR	Master Transfer Error. 0 = No errors detected. 1 = A master error occurred. The conditions surrounding the error are indicated by the master status register (TWIMSTAT). This bit is sticky and is cleared by writing a 1 to its bit location.

Table A-61. Interrupt Source Register Bit Descriptions (Cont'd)

Bit	Name	Description
6	TWITXINT	Transmit FIFO Service. 0 = No errors detected. 1 = The transmit FIFO buffer has one or two 8-bit locations available to be written. If TWITXINT2 is 0, this bit is set each time TWITXS is updated to either 01 or 00. If TWITXINT2 is 1, this bit is set each time TWITXS is updated to 00. This bit is sticky and is cleared by writing 1 to its bit location.
7	TWIRXINT	Receive FIFO Service. 0 = No errors detected. 1 = The receive FIFO buffer has one or two 8-bit locations containing data to be read. If TWIRXINT2 is 0, this bit is set each time TWIRXS is updated to either 01 or 11. If RTWIRXINT2 is 1, this bit is set each time TWIRXS is updated to 11. This bit is sticky and is cleared by writing 1 to its bit location.

Interrupt Enable Register (TWIIMASK)

The TWI interrupt enable register (TWIIMASK, shown in [Figure A-68](#) and described in [Table A-62](#)) enables interrupt sources to assert the interrupt output. Each enable bit corresponds with one interrupt source bit in the TWI interrupt source register (TWIIRPTL). Reading and writing the TWI interrupt enable register does not affect the contents of the TWI interrupt source register. For all bits, 0 = interrupt generation disabled and 1 = interrupt generation enabled.

TWIIMASK (0x4420)

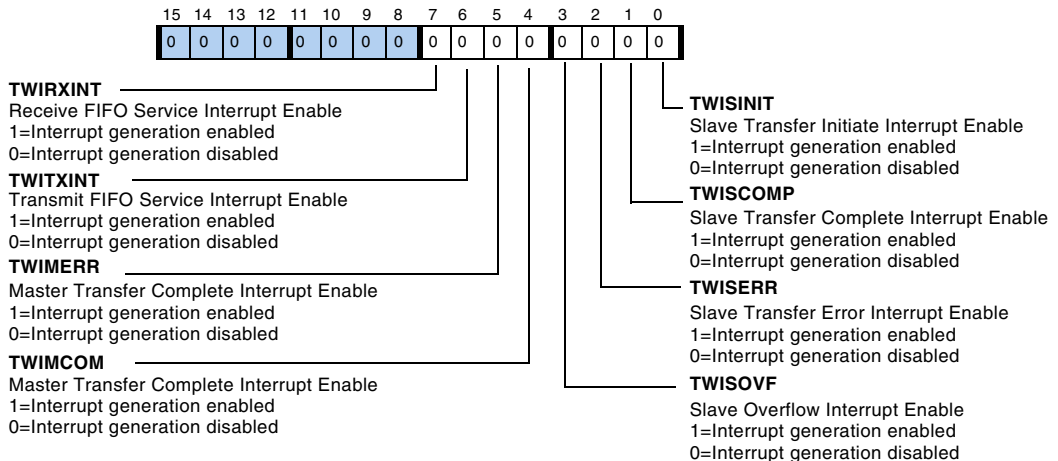


Figure A-68. Interrupt Mask Register

Table A-62. Interrupt Mask Register Bit Descriptions

Bit	Name	Description
0	TWISINIT	Slave Transfer Initiate Interrupt Enable. 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
1	TWISCOMP	Slave Transfer Complete Interrupt. 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
2	TWISERR	Slave Transfer Error Interrupt. 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
3	TWISOVF	Slave Over Flow Interrupt Enable. 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
4	TWIMCOM	Master Transfer Complete Interrupt Enable. 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
5	TWIMERR	Master Transfer Error Interrupt Enable. 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.

Two Wire Interface Registers

Table A-62. Interrupt Mask Register Bit Descriptions (Cont'd)

Bit	Name	Description
6	TWITXINT	Transmit FIFO Service Interrupt Enable. 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
7	TWIRXINT	Receive FIFO Service Interrupt Enable. 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.

8-Bit Transmit FIFO Register (TXTWI8)

The TWI FIFO transmit 8-bit register (TXTWI8, shown in [Figure A-69](#)) holds an 8-bit data value written into the FIFO buffer. Transmit data is entered into the corresponding transmit buffer in a first-in, first-out order. Although peripheral bus writes are 32 bits, a write access to TXTWI8 adds only one transmit data byte to the FIFO buffer. With each access, the transmit status (TWITXS) field in the TWIFIFOSTAT register is updated. If an access is performed while the FIFO buffer is full, the core waits until there is at least one byte space in the transmit FIFO buffer and then completes the write access. The bits in this register are write-only.

TXTWI8 (0x4480)

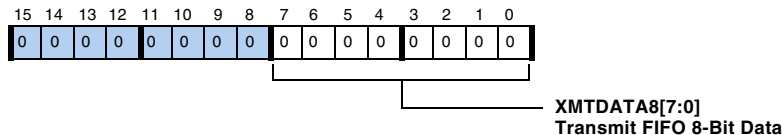


Figure A-69. 8-Bit Transmit FIFO Register

16-Bit Transmit FIFO Register (TXTWI16)

The TWI FIFO transmit 16-bit register (TXTWI16, shown in [Figure A-70](#)) holds a 16-bit data value written into the FIFO buffer. To reduce interrupt output rates and peripheral bus access times, a 16-bit transfer data access can be performed. Two data bytes can be written, effectively filling the transmit FIFO buffer with a single access. The data is written in little-endian byte order as shown in [Figure A-70](#), where byte 0 is the first byte to be transferred and byte 1 is the second byte to be transferred.

TXTWI16 (0x4484)

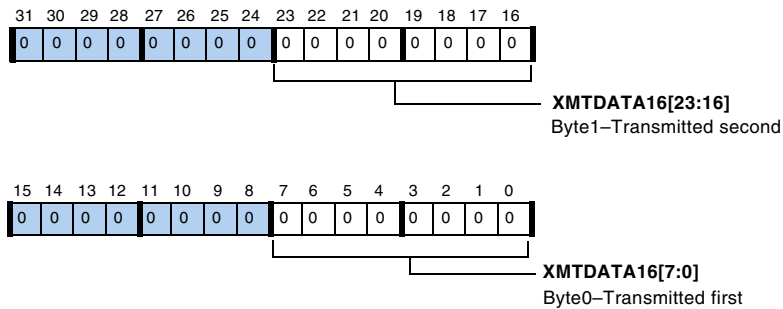


Figure A-70. 16-Bit Transmit FIFO Register

8-Bit Receive FIFO Register (RXTWI8)

The TWI FIFO receive data 8-bit register (RXTWI8, shown in [Figure A-71](#)) holds an 8-bit data value read from the FIFO buffer. Receive data is read from the corresponding receive buffer in a first-in first-out order.

Although peripheral bus reads are 32 bits, a read access to RXTWI8 can only access one receive data byte from the FIFO buffer. With each access, the receive status (TWIRXS) field in the TWIFIFOSTAT register is updated. If an access is performed while the FIFO buffer is empty, the core waits until there is at least one byte in the receive FIFO buffer and then completes the read access. All bits in this register are read-only.

RXTWI8 (0x4488)

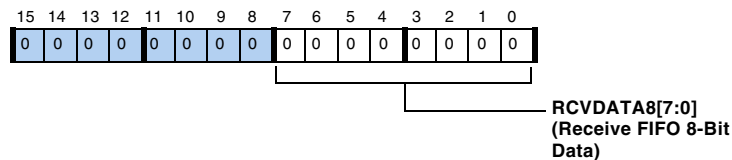


Figure A-71. 8-Bit Receive FIFO Register

16-Bit Receive FIFO Register (RXTWI16)

The TWI FIFO receive data-double byte register (RXTWI16, shown in [Figure A-72](#)) holds a 16-bit data value read from the FIFO buffer. To reduce interrupt output rates and peripheral bus access times, a double-byte receive data access can be performed. Two data bytes can be read, effectively emptying the receive FIFO buffer with a single access. The data is read in little-endian byte order, as shown in [Figure A-72](#), where byte 0 is the first byte received and byte 1 is the second byte received. With each access, the receive status (TWIRXS) field in the TWIFIFOSTAT register is updated to indicate it is empty. If an access is performed while the FIFO buffer is not full, the core waits until the receive FIFO buffer is full and then completes the read access. All bits in this register are read-only.

RXTW16 (0x4484)

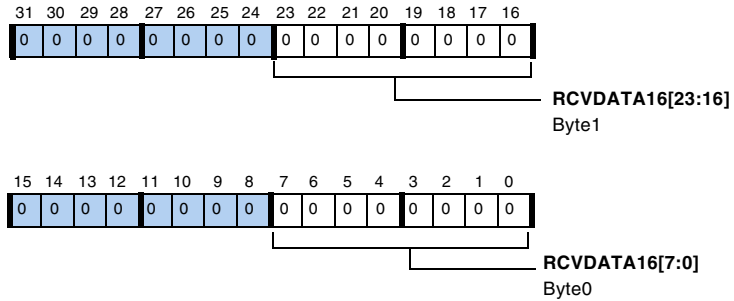


Figure A-72. 16-Bit Receive FIFO Register

Precision Clock Generator Registers

The precision clock generator (PCG) consists of four identical units. Each of these units (A, B, C, and D) generates one clock (CLKA_0, CLKB_0, CLKC_0 or CLKD_0) and one frame sync (FSA_0, FSB_0, FSC_0 or FSD_0) output. These units can take an input clock signal from a crystal oscillator buffer output or any of the sources in Group A of the signal routing unit (SRU).

Control Registers (PCG_CTLxx)

The control registers operate exactly the same for each clock unit. The control registers enable clocks, frame syncs, and select divisors for each clock unit. These registers are shown in [Figure A-73](#) and [Figure A-74](#) and described in [Table A-63](#) and [Table A-64](#). Note that where letters and slashes appear, for example A/B/C/D, any clock unit can be chosen.

Precision Clock Generator Registers

PCG_CTLA0 (0x24C0)

PCG_CTLB0 (0x24C2)

PCG_CTLC0 (0x24C6)

PCG_CTLD0 (0x24C8)

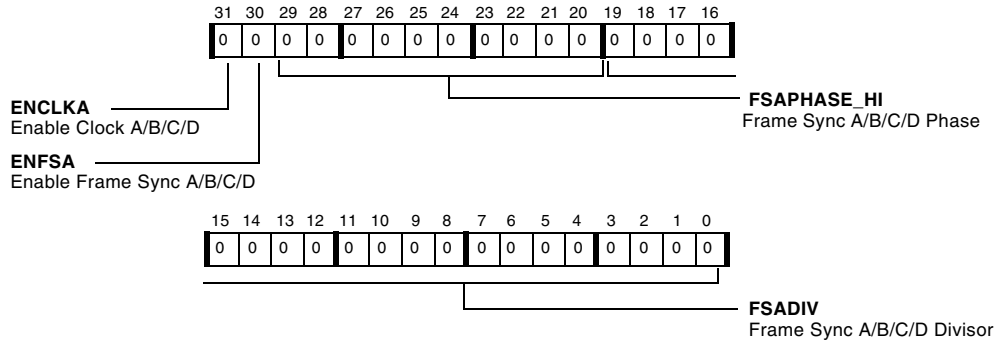


Figure A-73. PCG_CTLx0 Registers

Table A-63. PCG_CTLx0 Register Bit Descriptions

Bit	Name	Description
19–0	FSxDIV	Divisor for Frame Sync A/B/C/D.
29–20	FSxPHASE_HI	Phase for Frame Sync A/B/C/D. This field represents the upper half of the 20-bit value for the channel A/B/C/D frame sync phase. See also FSXPHASE_LO (Bits 29-20) in PCG_CTLX_1 described on page A-157 .
30	ENFSx	Enable Frame Sync A/B/C/D. 0 = Specified frame sync generation disabled 1 = Specified frame sync generation enabled
31	ENCLKx	Enable Clock A/B/C/D. 0 = Specified clock generation disabled 1 = Specified clock generation enabled

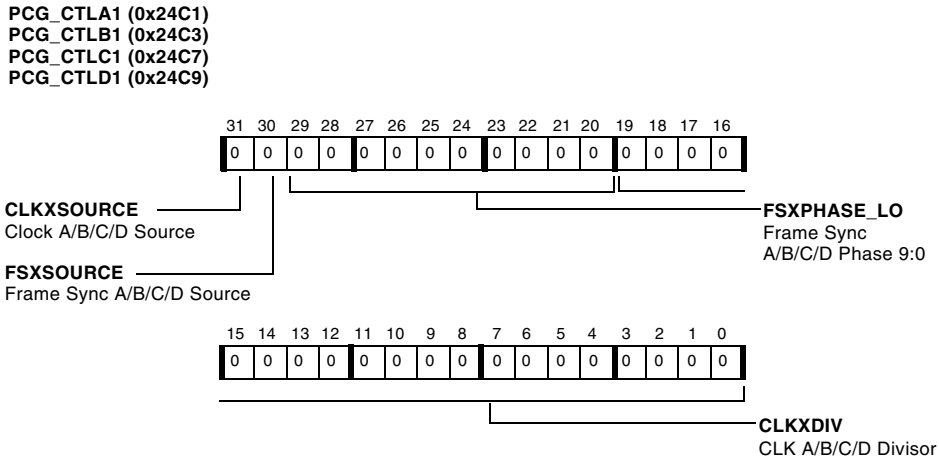


Figure A-74. PCG_CTLx1 Register

Table A-64. PCG_CTLx1 Register Bit Descriptions

Bit	Name	Description
19–0	CLKxDIV	Divisor for Clock A/B/C/D.
29–20	FSxPHASE_LO	Phase for Frame Sync A/B/C/D. This field represents the lower half of the 20-bit value for the channel A/B/C/D frame sync phase. See also FSXPHASE_HI (Bits 29-20) in PCG_CTLx1 described on page A-156 .
30	FSxSOURCE	Frame Sync Source. Master clock source for frame sync A/B/C/D. 0 = XTAL buffer output selected for specified frame sync 1 = PCG_EXTX_I selected for specified frame sync
31	CLKxSOURCE	Clock Source. Master clock source for clock A/B/C/D. 0 = XTAL buffer output selected for specified clock 1 = PCG_EXTx_I selected for specified clock

PCG Pulse Width Registers

Pulse width is the number of input clock periods for which the frame sync output is HIGH. Pulse width should be less than the divisor of the frame sync. The pulse width control registers are shown in [Figure A-75](#) and [Figure A-76](#) and described in [Table A-65](#) and [Table A-66](#). Note that where letters and slashes appear, for example A/B/C/D, any clock unit can be chosen.

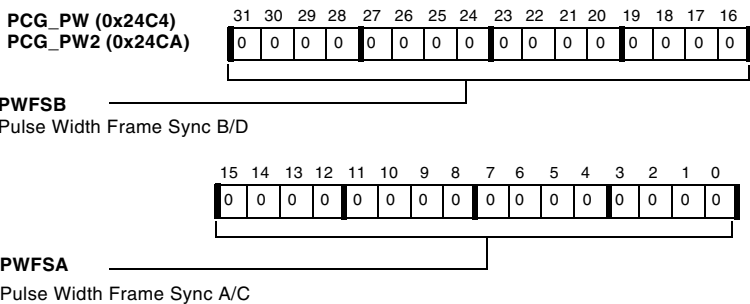


Figure A-75. PCG_PWx Registers (in Normal Mode)

Table A-65. PCG_PWx Register Bit Descriptions (in Normal Mode)

Bit	Name	Description
15–0	PWFSA	Pulse Width for Frame Sync A/C. Note: This is valid when not in bypass mode
31–16	PWFSB	Pulse Width for Frame Sync B/D. Note: This is valid when not in bypass mode

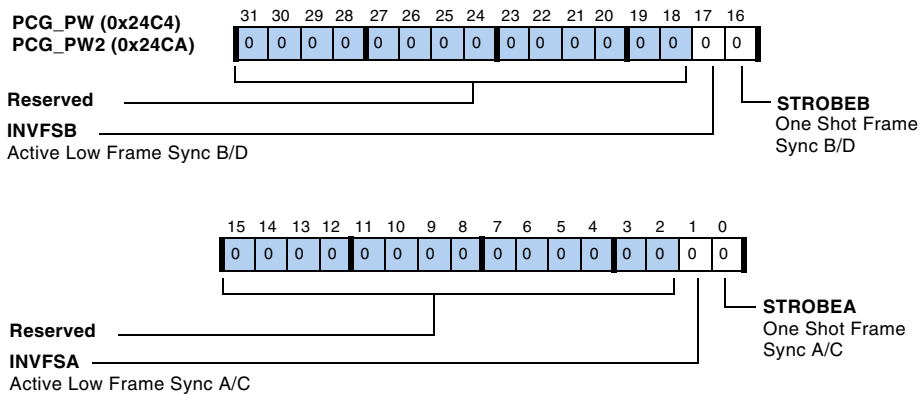


Figure A-76. PCG_PWx Registers (in Bypass Mode)

Table A-66. PCG_PWx Register Bit Descriptions (in Bypass Mode)

Bit	Name	Description
0	STROBEx	One Shot Frame Sync A/C. Frame sync is a pulse with duration equal to one period of the MISCA2_I signal repeating at the beginning of every frame. Note: This is valid in bypass mode only.
1	INVFSx	Active Low Frame Sync Select for Frame Sync A/C. 0 = Active high frame sync 1 = Active low frame sync
15–2	Reserved (In bypass mode, bits 15-2 are ignored.)	
16	STROBEx	One Shot Frame Sync B/D. Frame sync is a pulse with duration equal to one period of the MISCA3_I signal repeating at the beginning of every frame. Note: This is valid in bypass mode only
17	INVFSx	Active Low Frame Sync Select. 0 = Active high frame sync 1 = Active low frame sync
31–18	Reserved (In bypass mode, bits 31–18 are ignored.)	

PCG Frame Synchronization Registers (PCG_SYNCx)

These registers, shown in [Figure A-77](#), and [Figure A-78](#) and described in [Table A-67](#) and [Table A-68](#), allow programs to synchronize the clock frame syncs units with external frame syncs. [For more information, see “Frame Sync” on page 13-8.](#)

PCG_SYNC (0x24C5)

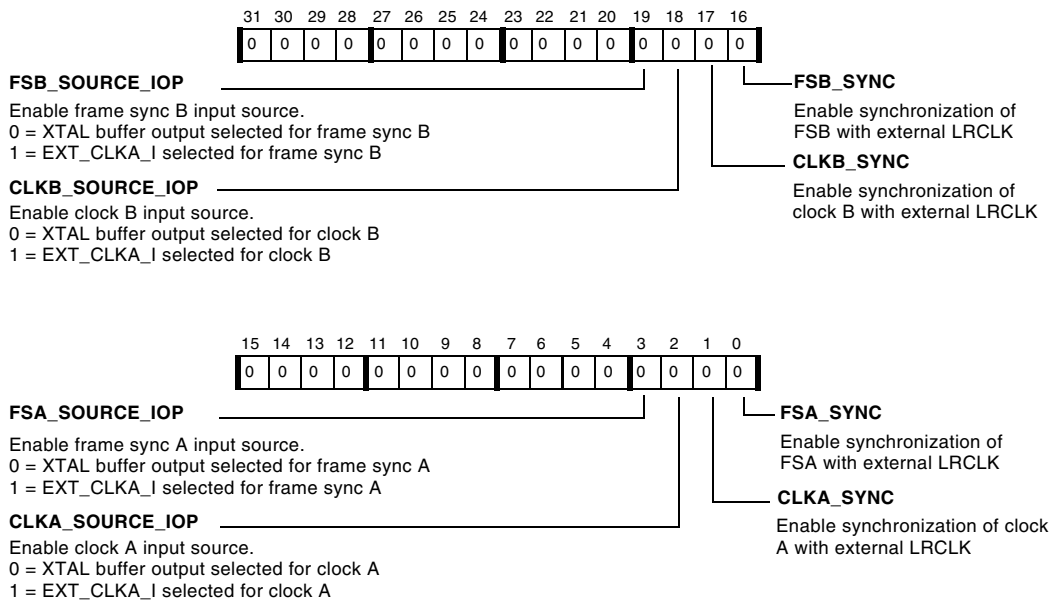


Figure A-77. PCG_SYNC Register

Table A-67. PCG_SYNC Register Bit Descriptions

Bit	Name	Description
0	FSA_SYNC	Enable synchronization of frame sync A with external frame sync. 0 = Frame sync disabled 1 = Frame sync enabled
1	CLKA_SYNC	Enable synchronization of clock A with external frame sync. 0 = Clock disabled 1 = Clock enabled
2	CLKA_SOURCE_IOP	Enable clock A input source. 0 = XTAL buffer output selected for clock A 1 = EXT_CLKA_I selected for clock A
3	FSA_SOURCE_IOP	Enable frame sync A input source. 0 = XTAL buffer output selected for frame sync A 1 = EXT_CLKA_I selected for frame sync A
16	FSB_SYNC	Enable synchronization of frame sync B with external frame sync. 0 = Frame sync disabled 1 = Frame sync enabled
17	CLKB_SYNC	Enable synchronization of clock B with external frame sync. 0 = Clock disabled 1 = Clock enabled
18	CLKB_SOURCE_IOP	Enable clock B input source. 0 = XTAL buffer output selected for clock B 1 = EXT_CLKA_I selected for clock B
19	FSB_SOURCE_IOP	Enable frame sync B input source. 0 = XTAL buffer output selected for frame sync B 1 = EXT_CLKA_I selected for frame sync B

Precision Clock Generator Registers

PCG_SYNC2 (0x24CB)

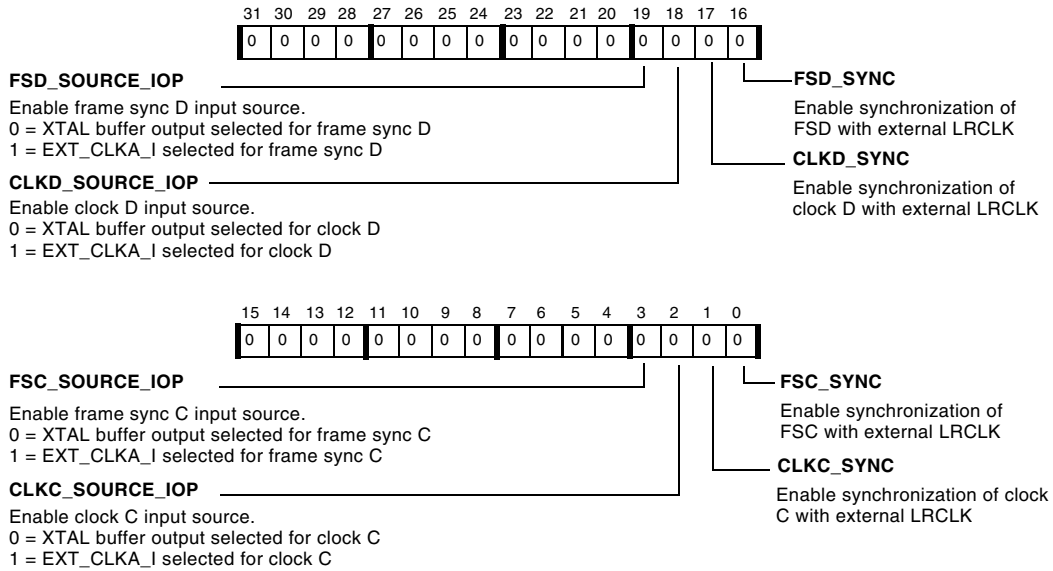


Figure A-78. PCG_SYNC2 Register

Table A-68. PCG_SYNC2 Register Bit Descriptions

Bit	Name	Description
0	FSC_SYNC	Enable synchronization of frame sync C with external frame sync. 0 = Frame sync disabled 1 = Frame sync enabled
1	CLKC_SYNC	Enable synchronization of clock C with external frame sync. 0 = Clock disabled 1 = Clock enabled
2	CLKC_SOURCE_IOP	Enable clock C input source. 0 = XTAL buffer output selected for clock C 1 = EXT_CLKA_I selected for clock C

Table A-68. PCG_SYNC2 Register Bit Descriptions (Cont'd)

Bit	Name	Description
3	FSC_SOURCE_IOP	Enable frame sync C input source. 0 = XTAL buffer output selected for frame sync C 1 = EXT_CLKA_I selected for frame sync C
16	FSD_SYNC	Enable synchronization of frame sync D with external frame sync. 0 = Frame sync disabled 1 = Frame sync enabled
17	CLKD_SYNC	Enable synchronization of clock D with external frame sync. 0 = Clock disabled 1 = Clock enabled
18	CLKD_SOURCE_IOP	Enable clock D input source. 0 = XTAL buffer output selected for clock D 1 = EXT_CLKA_I selected for clock D
19	FSD_SOURCE_IOP	Enable frame sync D input source. 0 = XTAL buffer output selected for frame sync D 1 = EXT_CLKA_I selected for frame sync D

Peripheral Interrupt Priority Control Registers

The following sections provide descriptions of the programmable interrupts that are used in the ADSP-21367/8/9 and ADSP-2137x processors. For information on the interrupt registers and the interrupt vector table, see [Appendix B, Interrupts](#).

Peripheral Interrupt Priority Control Registers (PICRx)

These registers allow programs to substitute the default interrupts for some other interrupt source. [Table A-69](#) lists the locations to program into the IOP programmable interrupt control registers (PICR) to route an IOP interrupt source to a corresponding processor interrupt location.

[Table A-69](#) defines the PICR bits which are programmed to select the source for each peripheral interrupt. Priority programming can be accomplished by changing the sources for each peripheral interrupt. For example, if peripheral x needs high priority, the high priority peripheral interrupt source is set as that peripheral.

Table A-69. Peripheral Interrupt Controller Routing Table

Interrupt Name	Vector Address	Programmable Interrupt Control Register (PICR)	Default Select Value	Default Function	Priority
P0I	0x2C	PICR0[4:0]	0x00	DAI1I interrupt	HIGHEST
P1I	0x30	PICR0[9:5]	0x01	SPIA interrupt	
P2I	0x34	PICR0[14:10]	0x02	IOP GP timer-0 interrupt	
P3I	0x38	PICR0[19:15]	0x03	SPORT1 interrupt	
P4I	0x3C	PICR0[24:20]	0x04	SPORT3 interrupt	
P5I	0x40	PICR0[29:25]	0x05	SPORT5 interrupt	
P6I	0x44	PICR1[4:0]	0x06	SPORT0 interrupt	
P7I	0X48	PICR1[9:5]	0x07	SPORT2 interrupt	
P8I	0X4C	PICR1[14:10]	0x08	SPORT4 interrupt	
P9I	0X50	PICR1[19 :15]	0x09	External port DMA channel0 interrupt	
P10I	0X54	PICR1[24:20]	0x0A	IOP GP timer-1 interrupt	
P11I	0x58	PICR1[29:25]	0x0B	SPORT7 interrupt	
P12I	0x5C	PICR2[4:0]	0x0C	DAI2I interrupt	
P13I	0x60	PICR2[9:5]	0x0D	External port DMA channel1 interrupt	
P14I	0x64	PICR2[14:10]	0x0E	DPI interrupt	
P15I	0x68	PICR2[19:15]	0x0F	MEM/MEM interrupt	
P16I	0x6C	PICR2[24:20]	0x10	SPORT6 interrupt	
P17I	0x70	PICR2[29:25]	0x11	IOP GP timer-2 interrupt	
P18I	0x74	PICR3[4:0]	0x12	SPIB interrupt	
UART0RxI			0x13	UART 0 receive interrupt	
UART1RxI			0x14	UART 1 receive interrupt	

Peripheral Interrupt Priority Control Registers

Table A-69. Peripheral Interrupt Controller Routing Table (Cont'd)

Interrupt Name	Vector Address	Programmable Interrupt Control Register (PICR)	Default Select Value	Default Function	Priority
UART0TxI			0x15	UART 0 transmit interrupt	
UART1TxI			0x16	UART 0 transmit interrupt	
TWII			0x17	Two wire interface interrupt	
PWMI			0x18	PWM interrupt	
Reserved			0x19 – 0x1E	Reserved	
Logic High			0x1F	Software option to set IOP interrupts	LOWEST

Peripheral Interrupt Priority0 Control Register (PICR0)

This 32-bit, read/write register controls programmable peripheral interrupts 0–5 and the default sources shown in [Figure A-79](#). This register is located at address 0x2200. The reset value of this register is 0x0A418820.

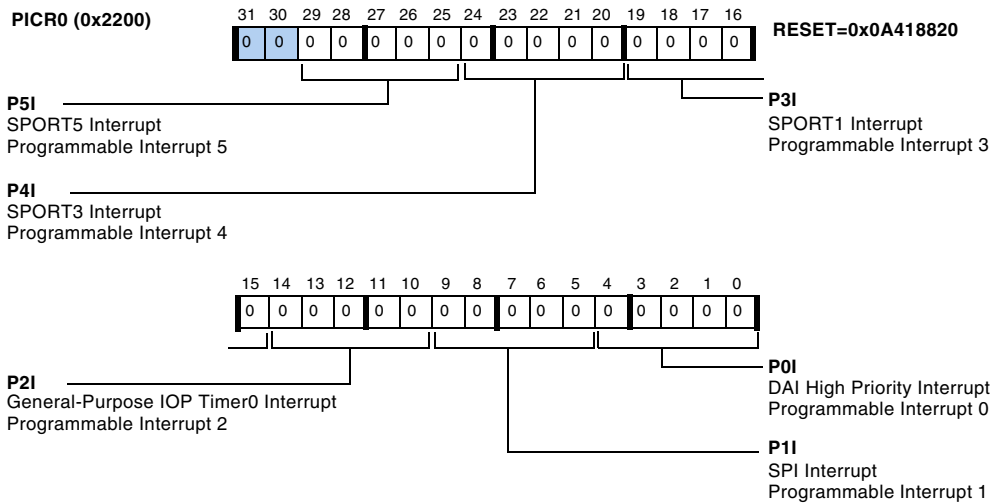


Figure A-79. PICR0 Register

Peripheral Interrupt Priority Control Registers

Peripheral Interrupt Priority1 Control Register (PICR1)

This register controls programmable peripheral interrupts 6–11 and the default sources shown in [Figure A-80](#). This 32-bit, read/write register is located at address 0x2201. The reset value of this register is 0x16A4A0E6.

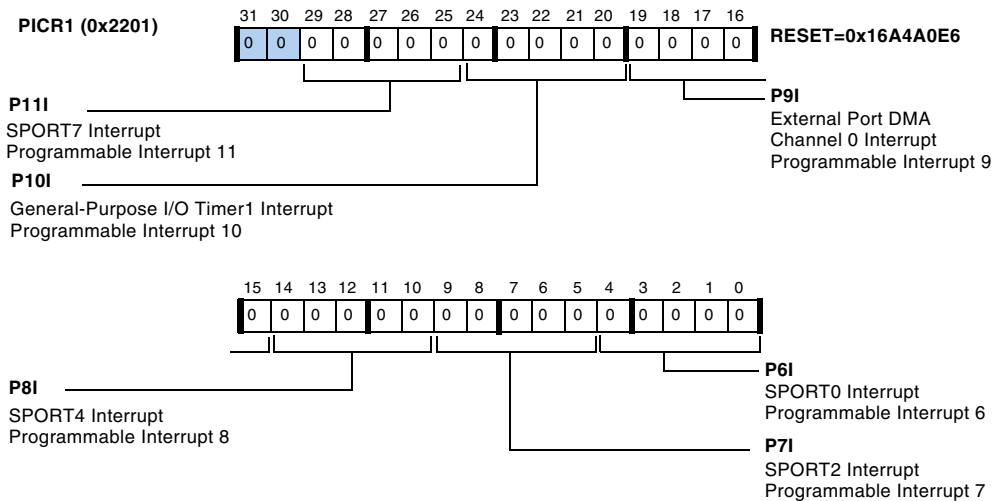


Figure A-80. PICR1 Register

Peripheral Interrupt Priority2 Control Register (PICR2)

This register controls programmable peripheral interrupts 12–17 as well as the default sources shown in [Figure A-81](#). This 32-bit, read/write register is located at address 0x2202. The reset value of this register is 0x2307B9AC.

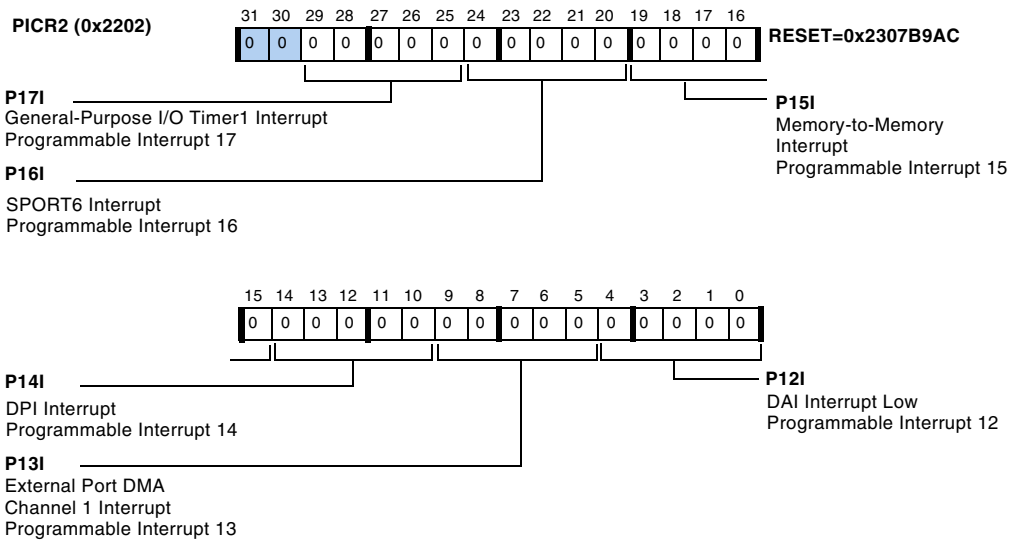


Figure A-81. PICR2 Register

Power Management Control Register (PMCTL)

Peripheral Interrupt Priority3 Control Register (PICR3)

This register controls programmable peripheral interrupt 18 as shown in [Figure A-82](#). This 32-bit, read/write register is located at address 0x2203. The reset value of this register is 0x00000012.

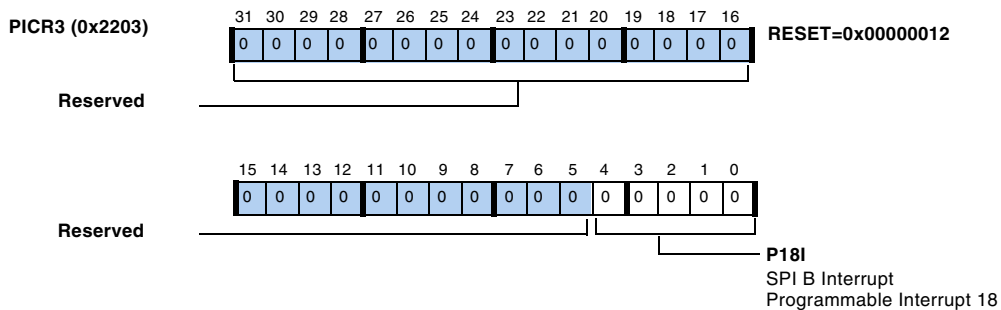


Figure A-82. PICR3 Register

Power Management Control Register (PMCTL)

The power management control register is a 32-bit, memory-mapped register. The PMCTL register's address is 0x2000. This register contains bits to control phase-lock loop (PLL) multiplier and divider (both input and output) values, PLL bypass mode, and clock enabling control for peripherals (see [Figure A-83](#) and [Table A-70](#)). This register also contains status bits, which keep track of the status of the CLK_CFG pins (read-only).

The core can write to all bits except the read-only status bits. The DIVEN bit is a logical bit, that is, it can be set, but on reads it always responds with zero.

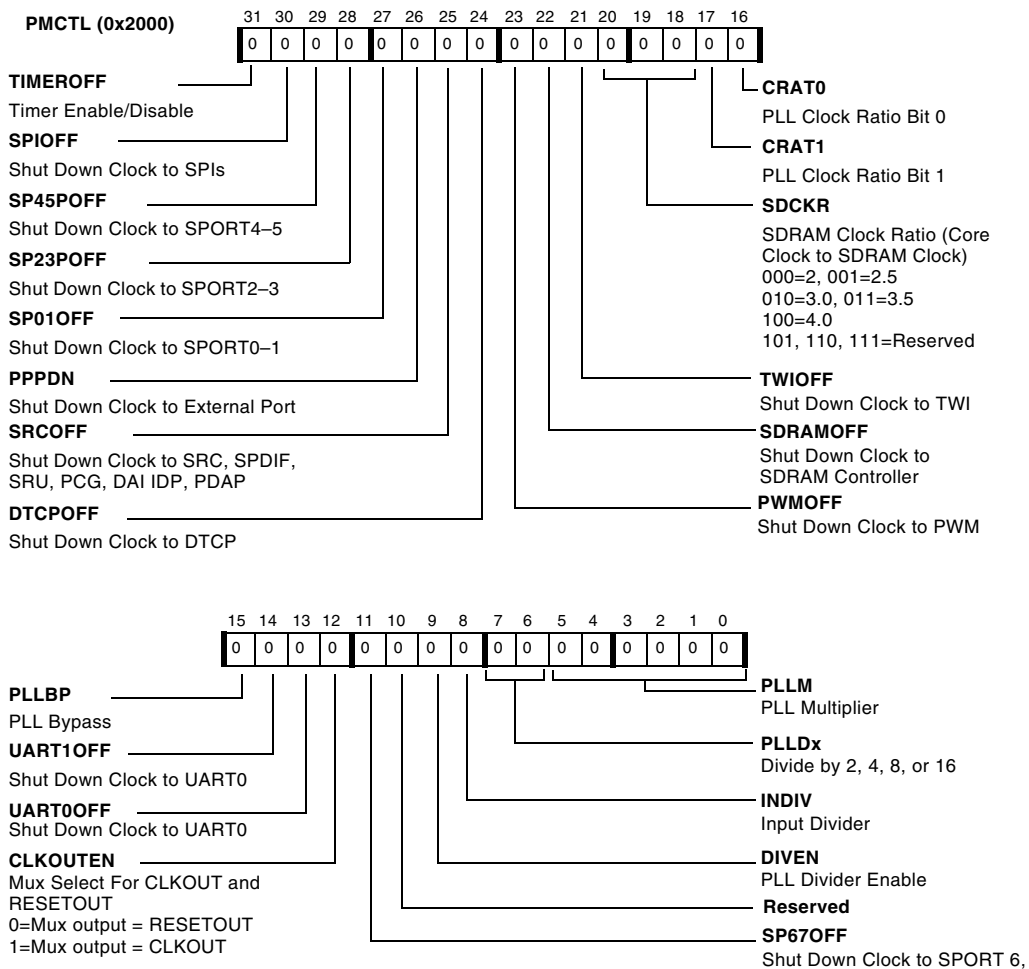


Figure A-83. PMCTL Register

Power Management Control Register (PMCTL)

Table A-70. PMCTL Register Bit Descriptions

Bit	Name	Description
5–0	PLLM	PLL Multiplier (read/write). PLLM = 0 PLL multiplier = 64 0 < PLLM < 63 PLL multiplier = PLLM CLK_CFG1–0 reset value 00 = 0000110 01 = 100000 10 = 010000 11 = 000110
7–6	PLLDx	PLL Divider (read/write). 00 = CK divider = 1 01 = CK divider = 2 10 = CK divider = 4 11 = CK divider = 8 CLK_CFG1–0 reset value x x 00
8	INDIV	Input Divisor (read/write). 0 = divide by 1 1 = divide by 2 Reset Value = 0
9	DIVEN	Enable PLL Divider Value Loading (read/write). 0 = Do not load PLLDx 1 = Load PLLDx Reset value = 0
10	Reserved	
11	SP67OFF	Serial Port 6, 7 Clock Enable. 0 = SPORT 6, 7 in normal mode 1 = Shut down clock to SPORT 6, 7
12	CLKOUTEN	Clockout Enable. Mux select for CLKOUT and RESETOUT 0 = Mux output = RESETOUT 1 = Mux output = CLKOUT Reset value = 0
13	UART0OFF	UART0 Clock Enable. 0 = UART0 is in normal mode 1 = Shut down clock to UART0

Table A-70. PMCTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
14	UART1OFF	UART1 Clock Enable. 0 = UART1 is in normal mode 1 = Shut down clock to UART1
15	PLLBP	PLL Bypass Mode Indication. 0 = PLL is in normal mode 1 = Put PLL in bypass mode Reset value = 0
16	CRAT0	PLL Clock Ratio, CLKIN to CK (read only). Read only. For more detail, see the PLLM and PLLDx bit descriptions in this table. Reset value = CLK_CFG[1:0]
17	CRAT1	PLL Clock Ratio, CLKIN to CK (read only). For more detail, see the PLLM and PLLDx bit descriptions in this table. Reset value = CLK_CFG[1:0]
20–18	SDCKR	SDCLK Ratio. Core clock to SDRAM clock. 000 = RATIO = 2 001 = RATIO = 2.5 010 = RATIO = 3.0 011 = RATIO = 3.5 100 = RATIO = 4.0 101, 110, 111 = Reserved
21	TWIOFF	TWI Clock Enable. 0 = TWI is in normal mode 1 = Shut down clock to TWI
22	SDRAMOFF	SDRAM Clock Enable. 0 = SDRAM is in normal mode 1 = Shut down clock to SDRAM
23	PWMOFF	PWM Clock Enable. 0 = PWM is in normal mode 1 = Shut down clock to PWM
24	DTCPOFF	DTCP Clock Enable. 0 = DTCP is in normal mode 1 = Shut down clock to DTCP

Power Management Control Register (PMCTL)

Table A-70. PMCTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
25	SRCOFF	SRC Off. 0 = SRC, SPDIF, SRU, PCG, DAI, IDP, PDAP blocks in normal mode 1 = Turn OFF clock to SRC, SPDIF, SRU, PCG, DAI, IDP, PDAP
26	PPPDN	External Port Enable/Disable. Shuts down the clock to the asynchronous memory interface as well as SDRAM. 0 = Port is in normal mode 1 = Shut down clock to external port Reset value = 0
27	SP01OFF	SPORT0, 1 Enable/Disable. 0 = SPORTs 0–1 are in normal mode 1 = Shut down clock to SPORTs 0–1 Reset value = 0
28	SP23OFF	SPORT2, 3 Enable/Disable. 0 = SPORTs 2–3 are in normal mode 1 = Shut down clock to SPORTs 2–3 Reset value = 0
29	SP45OFF	SPORT4, 5 Enable/Disable. 0 = SPORTs 4–5 are in normal mode 1 = Shut down clock to SPORTs 4–5 Reset value = 0
30	SPIOFF	SPI Enable/Disable. 0 = SPI is in normal mode 1 = Shut down clock to SPI Note: When this bit is set (= 1), the FLAGx pins cannot be used (via the FLAGS7–0 register bits) because the FLAGx pins are synchronized with the clock. Reset value = 0
31	TMEROFF	Timer Enable/Disable. 0 = Timer is in normal mode 1 = Shutdown clock to Timer Reset value = 0

Hardware Breakpoint Control Register

The **BRKCTL** register controls how breakpoints are used if bit 25, **UMODE**, is set. This user-accessible register, shown in [Figure A-84](#) and [Figure A-85](#) and described in [Table A-71](#), is located at address 0x30025.

The register is a 32-bit, memory-mapped I/O register. The core can write into this register. The bits related to the register are the same as in the “Enhanced Emulation Status Register” on page A-179.



Note that instruction address breakpoint negates such as **NEGPA1** and **NEGDA1** have an effect latency of four core clock cycles.

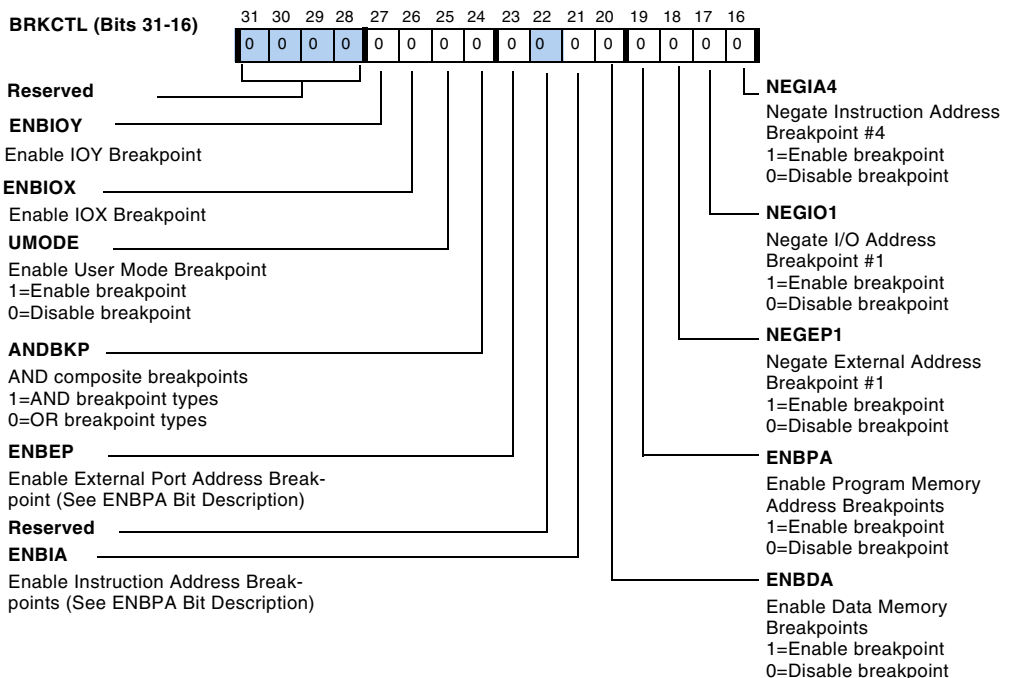


Figure A-84. BRKCTL Register (Bits 16–31)

Hardware Breakpoint Control Register

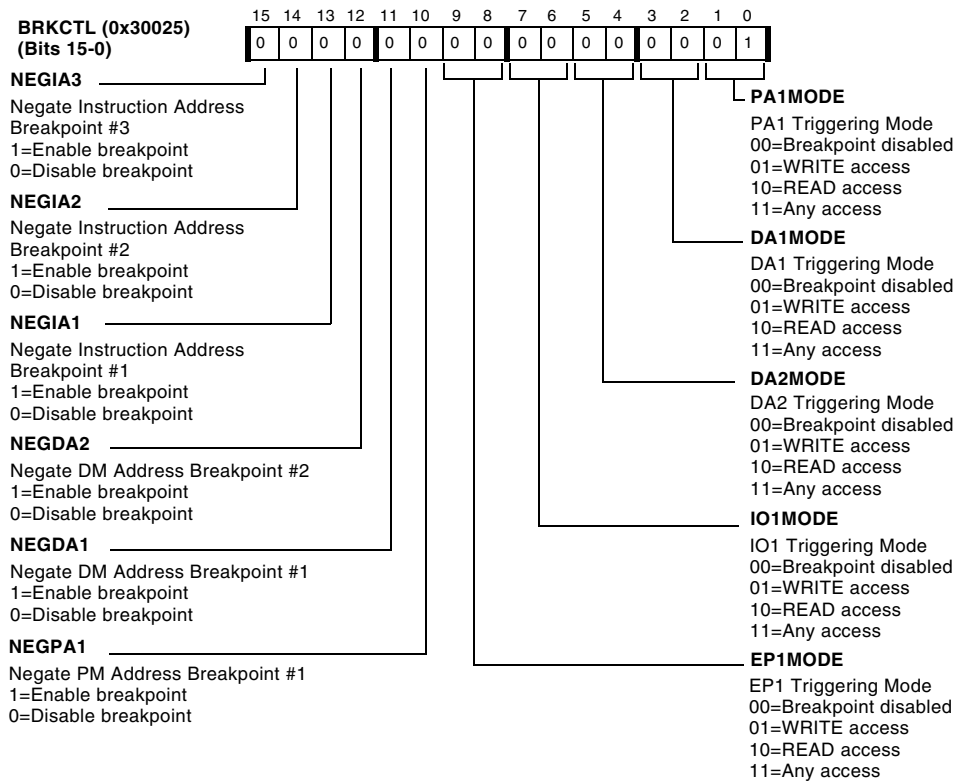


Figure A-85. BRKCTL Register (Bits 0–15)

Table A-71. BRKCTL Register Bit Descriptions

Bit	Name	Description
1–0	PA1MODE	PA1 Triggering Mode. 00 = Breakpoint disabled 01 = WRITE access 10 = READ access 11 = Any access
3–2	DA1MODE	DA1 Triggering Mode. 00 = Breakpoint disabled 01 = WRITE access 10 = READ access 11 = Any access
5–4	DA2MODE	DA2 Triggering Mode. 00 = Breakpoint disabled 01 = WRITE access 10 = READ access 11 = Any access
7–6	IO1MODE	IO1 Triggering Mode. 00 = Breakpoint is disabled 01 = WRITE accesses only 10 = READ accesses only 11 = Any access
9–8	EP1MODE	EP1 Triggering Mode. 00 = Breakpoint disabled 01 = WRITE access 10 = READ access 11 = Any access
10	NEGPA1	Negate Program Memory Data Address Breakpoint. Enable breakpoint events if the address is greater than the end register value OR less than the start register value. This function is useful to detect index range violations in user code. 0 = Do not negate breakpoint 1 = Negate breakpoint
11	NEGDA1	Negate Data Memory Address Breakpoint #1. For more information, see NEGPA1 bit description.
12	NEGDA2	Negate Data Memory Address Breakpoint #2. For more information, see NEGPA1 bit description.

Hardware Breakpoint Control Register

Table A-71. BRKCTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
13	NEGIA1	Negate Instruction Address Breakpoint #1. 0 = Do not negate breakpoint 1 = Negate breakpoint
14	NEGIA2	Negate Instruction Address Breakpoint #2. For more information, see NEGPA1 bit description.
15	NEGIA3	Negate Instruction Address Breakpoint #3. For more information, see NEGPA1 bit description.
16	NEGIA4	Negate Instruction Address Breakpoint #4. For more information, see NEGPA1 bit description.
17	NEGIO1	Negate I/O Address Breakpoint. For more information, see NEGPA1 bit description.
18	NEGEP1	Negate EP Address Breakpoint. For more information, see NEGPA1 bit description.
19	ENBPA	Enable Program Memory Data Address Breakpoints. The ENB bits enable each breakpoint group. Note that when the ANDBKP bit is set, breakpoint types not involved in the generation of the effective breakpoint must be disabled. 0 = Disable breakpoints 1 = Enable breakpoints
20	ENBDA	Enable Data Memory Address Breakpoints. For more information, see ENBPA bit description.
21	ENBIA	Enable Instruction Address Breakpoints. For more information, see ENBPA bit description.
22	Reserved	
23	ENBEP	Enable External Port Address Breakpoint. For more information, see ENBPA bit description.
24	ANDBKP	AND Composite Breakpoints. Enables ANDing of each breakpoint type to generate an effective breakpoint from the composite breakpoint signals. 0 = OR breakpoint types 1 = AND breakpoint types

Table A-71. BRKCTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
25	UMODE	User Mode Breakpoint Functionality Enable. Address Breakpoint 3. 0 = Disable user controlled breakpoint 1 = Enable user controlled breakpoint
26	ENBIOY	IOY Breakpoint Enable. 0 = Disable IOY breakpoint 1 = Enable IOY breakpoint 0
27	ENBIOX	IOX Breakpoint Enable. 0 = Disable IOX breakpoint 1 = Enable IOX breakpoint
31–28	Reserved	

Enhanced Emulation Status Register

The `EEMUSTAT` register reports the breakpoint status of the programs that run on the ADSP-21367/8/9 and ADSP-2137x processors. This register is a memory-mapped IOP register that can be accessed by the core. The `EEMUSTAT` register contains two status bits that report I/O breakpoints, one each for the two I/O buses (IOX and IOY).

When a breakpoint is reached, an interrupt occurs and the breakpoint's status bits are set. When the core returns from an interrupt, the breakpoint's status bits are cleared. This register is shown in [Figure A-86](#) and described in [Table A-72](#).

Enhanced Emulation Status Register

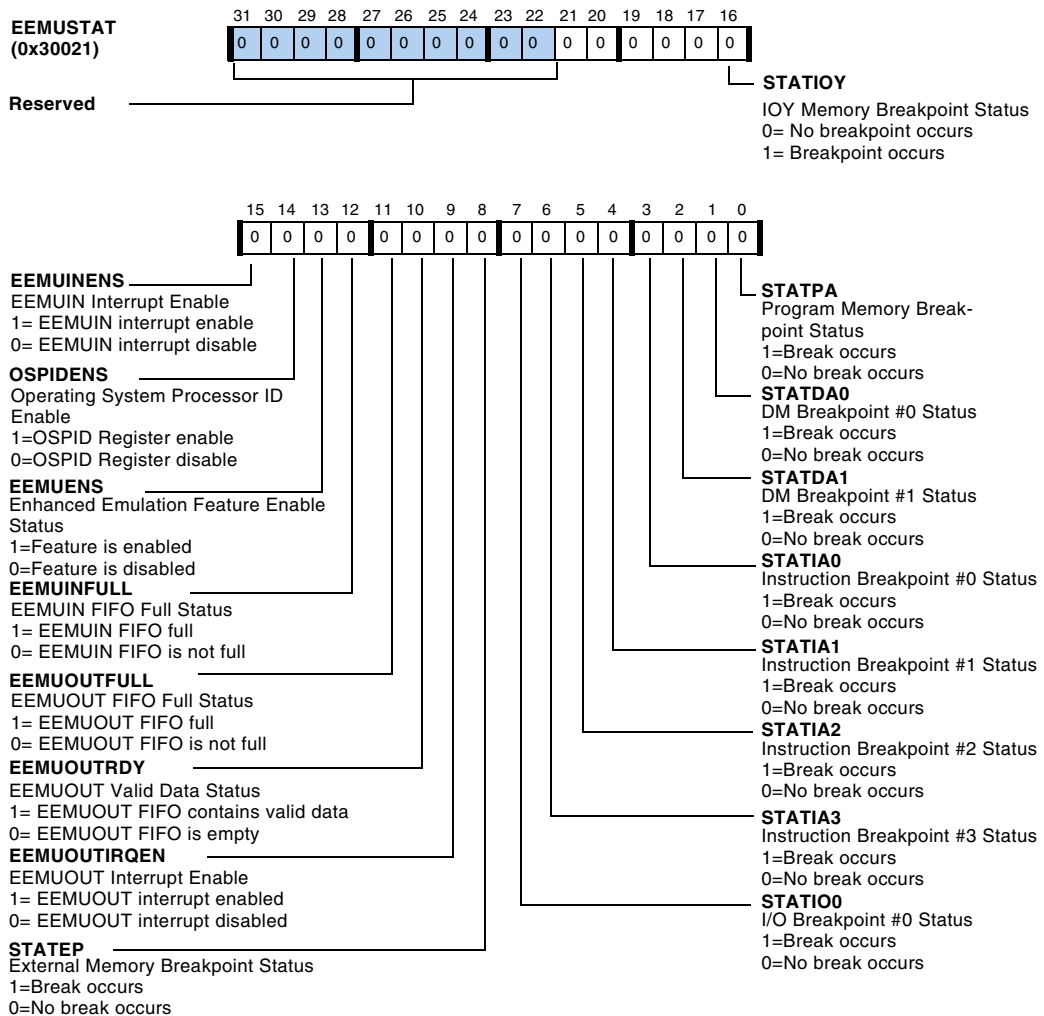


Figure A-86. EEMUSTAT Register

Table A-72. EEMUSTAT Register Bit Descriptions

Bit	Name	Description
0	STATPA	Program Memory Data Breakpoint Hit. ¹ 0 = No program memory breakpoint occurs 1 = Program memory breakpoint occurs
1	STATDA0	Data Memory Breakpoint Hit. ¹ 0 = No data memory #0 breakpoint occurs 1 = Data memory #0 breakpoint occurs
2	STATDA1	Data Memory Breakpoint Hit. ¹ 0 = No data memory #1 breakpoint occurs 1 = Data memory #1 breakpoint occurs
3	STATIA0	Instruction Address Breakpoint Hit. ¹ 0 = No instruction address #0 breakpoint occurs 1 = Instruction address #0 breakpoint occurs
4	STATIA1	Instruction Address Breakpoint Hit. ¹ 0 = No instruction address #1 breakpoint occurs 1 = Instruction address #1 breakpoint occurs
5	STATIA2	Instruction Address Breakpoint Hit. ¹ 0 = No instruction address #2 breakpoint occurs 1 = Instruction address #2 breakpoint occurs
6	STATIA3	Instruction Address Breakpoint Hit. ¹ 0 = No instruction address #3 breakpoint occurs 1 = Instruction address #3 breakpoint occurs
7	STATIO	I/O Address Breakpoint Hit. ¹ 0 = No I/OX address breakpoint occurs 1 = I/OX address breakpoint occurs
8	Reserved ¹	
9	EEMUOUTIRQEN	Enhanced Emulation EEMUOUT Interrupt Enable. ² 0 = EEMUOUT interrupt disable 1 = EEMUOUT interrupt enable Note: Interrupts are of the low priority variety
10	EEMUOUTRDY	Enhanced Emulation EEMUOUT Ready. ³ 1 = EEMUOUT FIFO contains valid data 0 = EEMUOUT FIFO is empty

Enhanced Emulation Status Register

Table A-72. EEMUSTAT Register Bit Descriptions (Cont'd)

Bit	Name	Description
11	EEMUOUTFULL	Enhanced Emulation EEMUOUT FIFO Status. ³ 0 = EEMUOUT FIFO is not full 1 = EEMUOUT FIFO full
12	EEMUINFULL	Enhanced Emulation EEMUIN Register Status. ⁴ 0 = EEMUIN register is empty 1 = EEMUIN register full
13	EEMUENS	Enhanced Emulation Feature Enable. ⁴ 0 = Enhanced emulation feature enable 1 = Enhanced emulation feature disable
14	OSPIDENS	OSPID Register Enable. ⁴ 0 = OSPID register enable 1 = OSPID register disable
15	EEMUINENS	EEMUIN Interrupt Enable. ⁴ 0 = EEMUIN interrupt disable 1 = EEMUIN interrupt enable
16	STATIOY	IOY Memory Breakpoint Status 0 = No breakpoint occurs 1 = Breakpoint occurs
31–17	Reserved	

1 Internal hardware sets this bit.

2 This bit is set and reset by the core.

3 The FIFO controller sets and resets this bit.

4 Internal hardware sets and resets this bit.

B INTERRUPTS

This chapter provides a complete listing of the registers that are used to configure and control interrupts. [Table B-2](#) shows all the processor interrupts, listed according to their bit position in the `IRPTL`, `LIRPTL`, and `IMASK` registers. Also shown are the addresses of the interrupt vectors. Each vector is separated by four memory locations.

Interrupt Vector Tables

The addresses in the vector table represent offsets from a base address. For an interrupt vector table in internal RAM, the base address is `0x9 0000` and for internal ROM, the base address is `0x8 0000` (see [Table B-1](#)). These are 48-bit addresses.

Table B-1. Interrupt Vector Table Base Address

Address ¹	Description
<code>0x0009 0000</code>	Internal RAM (block 0)
<code>0x0008 0000</code>	Internal ROM (block 0)

1 These are 48-bit addresses.

The interrupt name column in [Table B-2](#) lists a mnemonic name for each interrupt as they are defined by the definitions file (`def21369.h`) that comes with the software development tools.

Interrupt Vector Tables

Note that the SPI has only one interrupt for both transmit and receive operations and each serial port (SPORT) has only one interrupt for both transmit and receive.

Table B-2. Interrupt Vector Addresses

Interrupt Number	Register	IRPTL/ LIRPTL/ MASK Bit#	Vector Address	Interrupt Name	Function
0	IRPTL	0	0x00	EMUI	Emulator (read-only, non-maskable) HIGHEST PRIORITY
1	IRPTL	1	0x04	RSTI ¹	Reset (read-only non-maskable)
2	IRPTL	2	0x08	IICDI	Illegal input condition detected
3	IRPTL	3	0x0C	SOVFI	Status loop or mode stack overflow; or PC stack full
4	IRPTL	4	0x10	TMZHI	Timer=0 (high priority option)
5	IRPTL	5	0x14	SPERRI	SP error interrupt
6	IRPTL	6	0x18	BKPI	Hardware breakpoint interrupt
7	IRPTL	7	0x1C	Reserved	
8	IRPTL	8	0x20	IRQ2I	$\overline{\text{IRQ2I}}$ asserted
9	IRPTL	9	0x24	IRQ1I	$\overline{\text{IRQ1I}}$ asserted
10	IRPTL	10	0x28	IRQ0I	$\overline{\text{IRQ0I}}$ asserted
11	IRPTL	11	0x2C	P0I	Programmable interrupt 0 (DAI1)
12	IRPTL	12	0x30	P1I ²	Programmable interrupt 1 (SPIA)
13	IRPTL	13	0x34	P2I	Programmable interrupt 2 (GPTMR0)
14	IRPTL	14	0x38	P3I	Programmable interrupt 3 (SP1)
15	IRPTL	15	0x3C	P4I	Programmable interrupt 4 (SP3)
16	IRPTL	16	0x40	P5I	Programmable interrupt 5 (SP5)
17	LIRPTL	0	0x44	P6I	Programmable interrupt 6 (SP0)
18	LIRPTL	1	0x48	P7I	Programmable interrupt 7 (SP2)

Table B-2. Interrupt Vector Addresses (Cont'd)

Interrupt Number	Register	IRPTL/ LIRPTL/ MASK Bit#	Vector Address	Interrupt Name	Function
19	LIRPTL	2	0x4C	P8I	Programmable interrupt 8 (SP4)
20	LIRPTL	3	0x50	P9I ³	Programmable interrupt 9 (EPDMA0)
21	LIRPTL	4	0x54	P10I	Programmable interrupt 10 (GPTMR1)
22	LIRPTL	5	0x58	P11I ³	Programmable interrupt 11 (SP7)
23	LIRPTL	6	0x5C	P12I	Programmable interrupt 12 (DAI2)
24	LIRPTL	7	0x60	P13I	Programmable interrupt 13 (EPDMA1)
25	IRPTL	17	0x64	P14I ³	Programmable interrupt 14 (DPI)
26	IRPTL	18	0x68	P15I	Programmable interrupt 15 (M2MI)
27	IRPTL	19	0x6C	P16I	Programmable interrupt 16 (SP6)
28	LIRPTL	8	0x70	P17I	Programmable interrupt 17 (GPTMR2)
29	LIRPTL	9	0x74	P18I ³	Programmable interrupt 18 (SPIB)
30	IRPTL	20	0x78	CB7I	Circular buffer 7 overflow
31	IRPTL	21	0x7C	CB15I	Circular buffer 15 overflow
32	IRPTL	22	0x80	TMZLI	Timer=0 (low priority option)
33	IRPTL	23	0x84	FIXI	Fixed-point overflow
34	IRPTL	24	0x88	FLTOI	Floating-point overflow exception
35	IRPTL	25	0x8C	FLTUI	Floating-point underflow exception
36	IRPTL	26	0x90	FLTII	Floating-point invalid exception
37	IRPTL	27	0x94	EMULI	Emulator low priority interrupt
38	IRPTL	28	0x98	SFT0I	User software interrupt 0
39	IRPTL	29	0x9C	SFT1I	User software interrupt 1

Interrupt Vector Tables

Table B-2. Interrupt Vector Addresses (Cont'd)

Interrupt Number	Register	IRPTL/ LIRPTL/ MASK Bit#	Vector Address	Interrupt Name	Function
40	IRPTL	30	0xA0	SFT2I	User software interrupt 2
41	IRPTL	31	0xA4	SFT3I	User software interrupt 3 LOWEST PRIORITY

- 1 If configured for internal ROM boot mode, then the base address for the interrupt vector table is the starting address of internal ROM or 0x00080000.
- 2 These interrupts have options to unmask at reset. Therefore, the peripherals that boot the processor should be allocated these interrupts: P1I, P9I.

Interrupt Priorities

The ADSP-21367/8/9 and ADSP-2137x SHARC processors support 19 prioritized IOP interrupts which are shown in [Table B-3](#). [Table B-3](#) also lists the value corresponding to each interrupt source. To route an IOP interrupt source to a corresponding programmable interrupt location, see “[Peripheral Interrupt Priority Control Registers](#)” on page A-164.

Table B-3. Interrupt Selection Values

No	Interrupt Source	Interrupt Select Value (5-Bits)	Comments
1	DAI1	0x00	DAI high priority interrupt
2	SPIAI	0x01	SPIA high priority interrupt
3	GPTMR0I	0x02	General-purpose IOP timer 0 interrupt
4	SP1I	0x03	Serial port 1 interrupt
5	SP3I	0x04	Serial port 3 interrupt
6	SP5I	0x05	Serial port 5 interrupt
7	SP0I	0x06	Serial port 0 interrupt
8	SP2I	0x07	Serial port 2 interrupt
9	SP4I	0x08	Serial port 4 interrupt
10	EP0I	0x09	External port channel 0 interrupt

Table B-3. Interrupt Selection Values (Cont'd)

No	Interrupt Source	Interrupt Select Value (5-Bits)	Comments
11	GPTMR1I	0x0A	General-purpose Timer 1 interrupt
12	SP7I	0x0B	Serial port 7 interrupt
13	DAI	0x0C	DAI low priority interrupt
14	EP1I	0x0D	External port channel 1 interrupt
15	DPI	0x0E	DPI interrupt
16	MTMDMAI	0x0F	Memory-to-memory DMA interrupt
17	SP6I	0x10	Serial Port 6 interrupt
18	GPTMR2I	0x11	General-purpose timer 2 interrupt
19	SPIBI	0x12	SPI B interrupt
20	UART0RXI	0x13	UART0 RX interrupt ¹
21	UART1RXI	0x14	UART1 RX interrupt ¹
22	UART0TXI	0x15	UART0 TX interrupt ¹
23	UART1TXI	0x16	UART1 TX interrupt ¹
24	TWII	0x17	TWI interrupt ¹
25	PWMI	0x18	PWM interrupt ¹
26	Reserved	0x18–0x1E	Reserved
27	LOGIC HIGH	0x1F	Software option to set IOP interrupts ¹

- ¹ These interrupts are not connected to the processor core interrupts by default. If these interrupts are required, then the PICR registers should be programmed explicitly. [For more information, see “Peripheral Interrupt Priority Control Registers” in Appendix A, Register Reference.](#)

Interrupt Registers

This section provides information on the registers that are used to configure and control interrupts. These registers are:

- “Interrupt Register (LIRPTL)” on page B-6
- “Interrupt Latch Register (IRPTL)” on page B-13
- “Interrupt Mask Register (IMASK)” on page B-18
- “Interrupt Mask Pointer Register (IMASKP)” on page B-22

Interrupt Register (LIRPTL)

The LIRPTL register is a non-memory-mapped, universal, system register (*Ureg* and *Sreg*). The reset value for these registers is 0x0000 0000. The LIRPTL register indicates latch status, select masking, and displays mask pointers for interrupts. [Figure B-1](#), [Figure B-2](#), and [Table B-4](#) provide bit definitions for the LIRPTL register.



The MSKP bits in the LIRPTL register and the entire IMASKP register are for interrupt controller use only. Modifying these bits interferes with the proper operation of the interrupt controller.

The interrupt bits 0 through 19 are programmable through the programmable interrupt controller registers (PICRx). The descriptions provided are their default sources. For information on their optional use, see “[Peripheral Interrupt Priority Control Registers](#)” on page A-164.

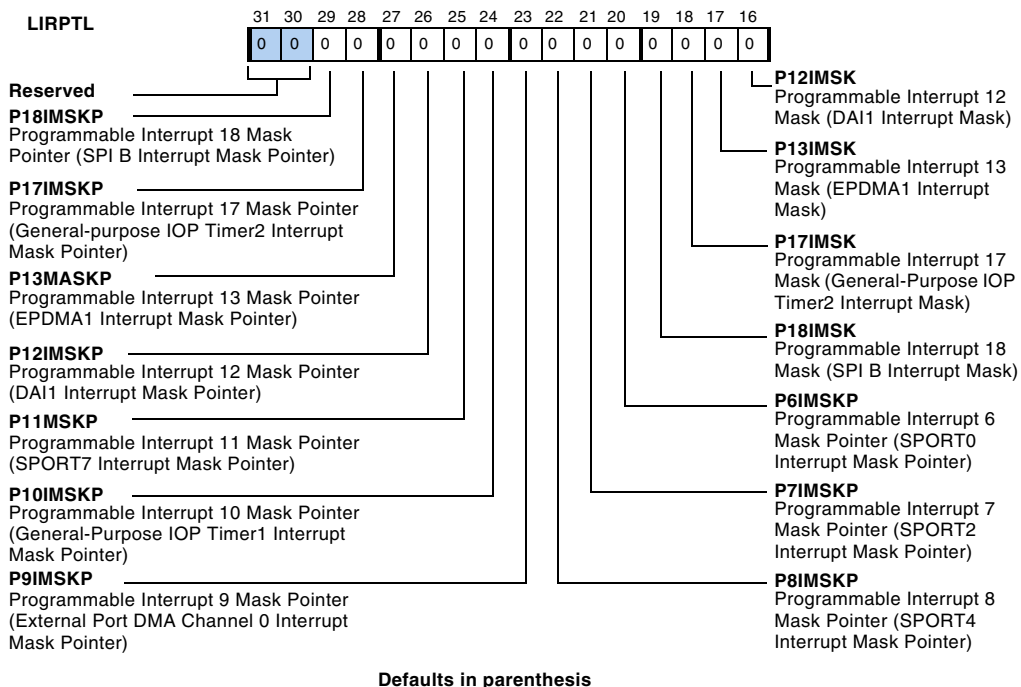


Figure B-1. LIRPTL Register (Bits 16–31)

Interrupt Registers

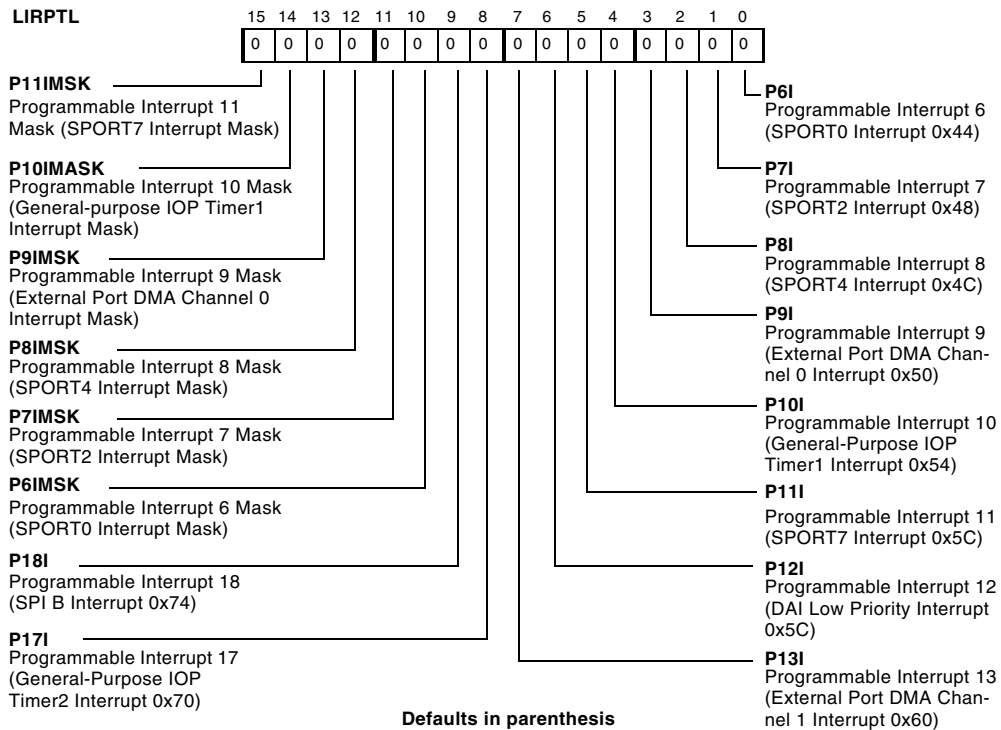


Figure B-2. LIRPTL Register (Bits 0–15)

Table B-4. LIRPTL Register Bit Descriptions

Bit	Name	Description
0	P6I (SP0I)	Programmable Interrupt 6 (SPORT 0 Interrupt). Indicates if an SP0I interrupt is latched and is pending (if set, = 1), or no SP0I is pending (if cleared, = 0). An SP0I interrupt occurs two cycles after the last bit of an input/output serial word is latched into/from RXSP0A/TXSP0A, RXSP0B/TXSP0B.
1	P7I (SP2I)	Programmable Interrupt 7 (SPORT 2 Interrupt). Indicates if an SP2I interrupt is latched and is pending (if set, = 1), or no SP2I is pending (if cleared, = 0). An SP2I interrupt occurs two cycles after the last bit of an input/output serial word is latched into/from RXSP2A/TXSP2A, RXSP2B/TXSP2B.

Table B-4. LIRPTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
2	P8I (SP4I)	Programmable Interrupt 8 (SPORT 4 Interrupt). Indicates if an SP4I interrupt is latched and is pending (if set, = 1), or no SP4I is pending (if cleared, = 0). An SP4I interrupt occurs two cycles after the last bit of an input/output serial word is latched into/from RXSP4A/TXSP4A, RXSP4B/TXSP4B.
3	P9I (EPDMA0I)	Programmable Interrupt 9 (External Port DMA Channel 0 Interrupt). Indicates if an external port interrupt (EPDMA0) is latched and pending (if set, = 1), or that no external port interrupt is pending (if cleared, = 0). An external port interrupt occurs when the DMA block transfer has completed. This interrupt also occurs during core-driven transfers when the Tx buffer is not full or the Rx buffer is not empty.
4	P10I (GPTMR1I)	Programmable Interrupt 10 (General-Purpose IOP Timer 1 Interrupt). Indicates if a GPTMR1I is latched and is pending (if set, = 1), if no GPTMR1I is pending (if cleared, = 0).
5	P11I (SP7I)	Programmable Interrupt 11 (SPORT 7 Interrupt). Indicates if an SP7I interrupt is latched and is pending (if set, = 1), or no SP7I is pending (if cleared, = 0). An SP7I interrupt occurs two cycles after the last bit of an input/output serial word is latched into/from RXSP7A/TXSP7A, RXSP7B/TXSP7B.
6	P12I (DAI2I)	Programmable Interrupt 12 (DAI2 Interrupt). Indicates if a DAI2 interrupt is latched and is pending (if set, = 1) or no DAI2 interrupt is pending (if cleared, = 0). This is the lower priority option.
7	P13I (EPDMA1I)	Programmable Interrupt 13 (External Port DMA Channel 1 Interrupt). Indicates if an external port interrupt (EPDMA1) is latched and pending (if set, = 1), or that no external port interrupt is pending (if cleared, = 0). An external port interrupt occurs when the DMA block transfer has completed. This interrupt also occurs during core-driven transfers when the Tx buffer is not full or the Rx buffer is not empty.
8	P17I (GPTMR2I)	Programmable Interrupt 17 (General-Purpose IOP Timer 2 Interrupt). Indicates if a GPTMR2I is latched and is pending (if set, = 1), or no GPTMR2I is pending (if cleared, = 0).

Interrupt Registers

Table B-4. LIRPTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
9	P18I (SPIB)	Programmable Interrupt 18 (SPI B Interrupt). Indicates if an SPIB interrupt is latched and pending (if set, = 1), or no SPIBI interrupt is pending (if cleared, = 0).
10	P6IMSK (SP0IMSK)	Programmable Interrupt Mask 6 (SPORT0 Interrupt Mask). Unmasks the SP0I interrupt (if set, = 1), or masks the SP0I interrupt (if cleared, = 0).
11	P7IMSK (SP2IMSK)	Programmable Interrupt Mask 7 (SPORT2 Interrupt Mask). Unmasks the SP2I interrupt (if set, = 1), or masks the SP2I interrupt (if cleared, = 0).
12	P8IMSK (SP4IMSK)	Programmable Interrupt Mask 8 (SPORT4 Interrupt Mask). Unmasks the SP4I interrupt (if set, = 1), or masks the SP4I interrupt (if cleared, = 0).
13	P9IMSK (EPDMA0MSK)	Programmable Interrupt Mask 9 (External Port DMA Channel 0 Interrupt Mask). Unmasks the EPDMA0 interrupt (if set, = 1), or masks the EPDMA0 interrupt (if cleared, = 0).
14	P10IMSK (GPTMR1IMSK)	Programmable Interrupt Mask 9 (General-Purpose IOP Timer 1 Interrupt Mask). Unmasks the GPTMR1I interrupt (if set, = 1), or masks the GPTMR1I interrupt (if cleared, = 0).
15	P11IMSK	Programmable Interrupt Mask 11 (SPORT7 Interrupt Mask). Unmasks the SP7I interrupt (if set, = 1), or masks the SP7I interrupt (if cleared, = 0).
16	P12IMSK (DAI2IMSK)	Programmable Interrupt Mask 12 (DAI2 Interrupt Mask). Unmasks the DAI2I (if set, = 1), or masks DAI2I (if cleared, = 0).
17	P13IMSK (EPDMA1IMSK)	Programmable Interrupt Mask 12 (External Port DMA Channel 1 Interrupt Mask). Unmasks the EPDMA1I interrupt (if set, = 1), or masks EPDMA1I interrupt (if set, = 0).
18	P17IMSK (GPTMR2IMSK)	Programmable Interrupt Mask 17 (General-Purpose IOP Timer 2 Interrupt Mask.) Unmasks the GPTMR2I interrupt (if set, = 1), or masks the GPTMR2I interrupt (if cleared, = 0).

Table B-4. LIRPTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
19	P18IMSK (SPIBMSK)	Programmable Interrupt Mask 18 (SPI Interrupt Mask Secondary SPI Port). Unmasks the SPIB interrupt (if set, = 1), or masks the SPIB interrupt (if cleared, = 0).
20	P6IMSKP (SP0IMSKP)	Programmable Interrupt Mask Pointer 9 (SPORT0 Interrupt Mask Pointer). When the processor is servicing another interrupt, this bit indicates if the SP0I interrupt is unmasked (if set, = 1), or the SP0I interrupt is masked (if cleared, = 0).
21	P7IMSKP (SP2IMSKP)	Programmable Interrupt Mask Pointer 7 (SPORT2 Interrupt Mask Pointer). When the processor is servicing another interrupt, this bit indicates if the SP2I interrupt is unmasked (if set, = 1), or the SP2I interrupt is masked (if cleared, = 0).
22	P8IMSKP (SP4IMSKP)	Programmable Interrupt Mask Pointer 8 (SPORT4 Interrupt Mask Pointer). When the processor is servicing another interrupt, this bit indicates if the SP4I interrupt is unmasked (if set, = 1), or the SP4I interrupt is masked (if cleared, = 0).
23	P9IMSKP (EPDMA0IMSKP)	Programmable Interrupt Mask Pointer 9 (External Port DMA Channel 0 Interrupt Mask Pointer). When the processor is servicing another interrupt, this bit indicates if the EPDMA0 interrupt is unmasked (if set, = 1), or masked (if cleared, = 0).
24	P10IMSKP (GPTMR1IMSKP)	Programmable Interrupt Mask Pointer 10 (General-Purpose IOP Timer 1 Interrupt Mask Pointer). When the processor is servicing another interrupt, this bit indicates if the GPTMR1I interrupt is unmasked (if set, = 1), or the GPTMR1I interrupt is masked (if cleared, = 0).
25	P11IMSKP	Programmable Interrupt Mask Pointer 11 (SPORT7 Interrupt Mask Pointer). When the processor is servicing another interrupt, this bit indicates if the SP7I interrupt is unmasked (if set, = 1), or the SP7I interrupt is masked (if cleared, = 0).
26	P12IMSKP (DAI2IMSKP)	Programmable Interrupt Mask Pointer 12 (DAI Low Priority Interrupt Mask Pointer). When the processor is servicing another interrupt, this bit indicates if the DAI2I is unmasked (if set, = 1), or masked (if cleared, = 0).

Interrupt Registers

Table B-4. LIRPTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
27	P13IMASKP (EPDMA1IMSKP)	Programmable Interrupt 13 Mask Pointer (External Port DMA Channel 1 Interrupt Mask Pointer). When the processor is servicing another interrupt, this bit indicates if the EPDMA1I is unmasked (if set, = 1) or masked (if cleared, = 0).
28	P17MSKP (GPTMR2IMSKP)	Programmable Interrupt Mask Pointer 17 (General-Purpose IOP Timer 2 Interrupt Mask Pointer). When the processor is servicing another interrupt, this bit indicates if the GPTMR2I interrupt is unmasked (if set, = 1), or the GPTMR2I interrupt is masked (if cleared, = 0).
29	P18MSKP (SPIBIMSKP)	Programmable Interrupt Mask Pointer 8 (SPIBI Interrupt Mask Pointer). When the processor is servicing another interrupt, this bit indicates if the SPIBI interrupt is unmasked (if set, = 1), or the SPIBI interrupt is masked (if cleared, = 0).
31–30	Reserved	

Interrupt Latch Register (IRPTL)

The **IRPTL** register is a non-memory-mapped, universal, system register (*Ureg* and *Sreg*). The reset value for this register is 0x0000 0000. The **IRPTL** register indicates latch status for interrupts. [Figure B-3](#), [Figure B-4](#) and [Table B-5](#) provide bit definitions for the **IRPTL** register.

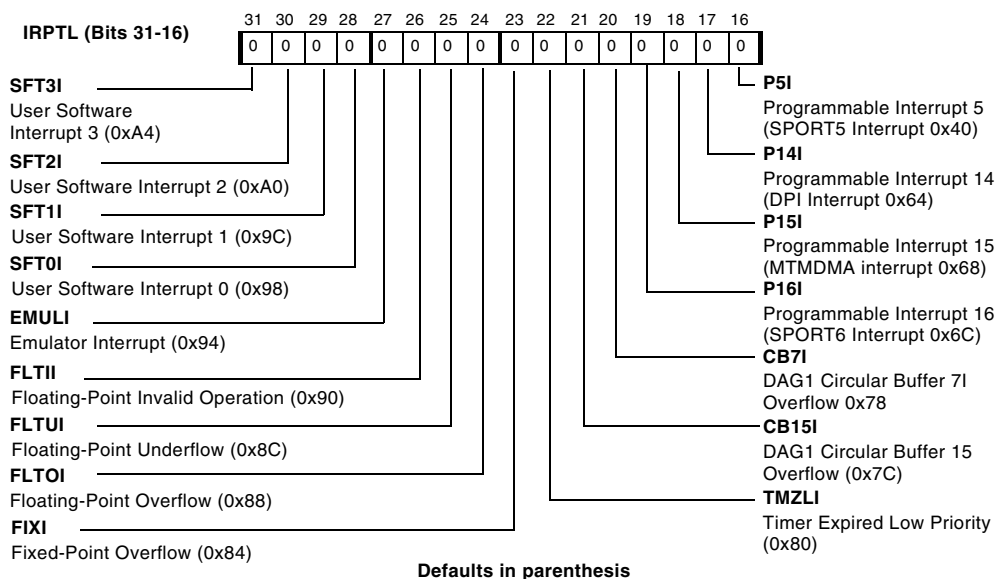
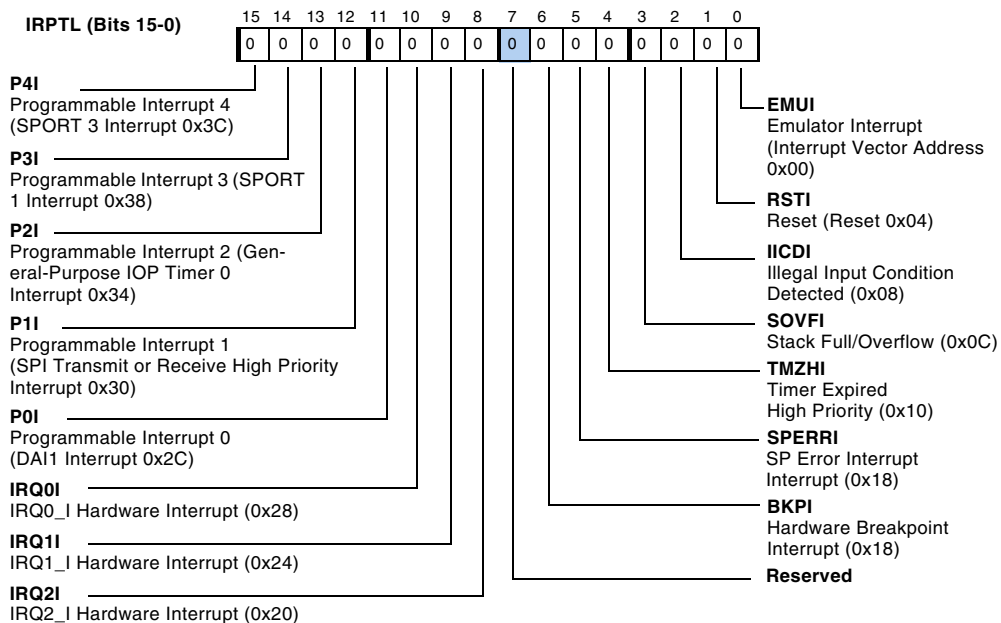


Figure B-3. IRPTL, IMASK, and IMASKP Registers (Bits 31–16)

Interrupt Registers



Defaults in parenthesis

Figure B-4. IRPTL, IMASK, and IMASKP Registers (Bits 15–0)

The interrupt latch bits 11 through 19 are programmable through the programmable interrupt controller register (PICR). The descriptions provided are their default sources. For information on their optional use, see [“Peripheral Interrupt Priority Control Registers” on page A-164](#).

Table B-5. Interrupt Latch (IRPTL) Register Bit Descriptions

Bit	Name	Description
0	EMUI	Emulator Interrupt. Indicates if an EMUI is latched and is pending (if set, = 1,) or no EMUI is pending (if cleared, = 0). An EMUI occurs on reset and when an external device asserts the $\overline{\text{EMU}}$ pin.
1	RSTI	Reset Interrupt. Indicates if an RSTI is latched and is pending (if set, = 1), or no RSTI is pending (if cleared, = 0). An RSTI occurs on reset as an external device asserts the $\overline{\text{RESET}}$ pin.

Table B-5. Interrupt Latch (IRPTL) Register Bit Descriptions (Cont'd)

Bit	Name	Description
2	IICDI	Illegal Input Condition Detected Interrupt. Indicates if an IICDI is latched and is pending (if set, = 1), or no IICDI is pending (if cleared, = 0). An IICDI occurs when a true results from the logical Oring of the illegal I/O processor register access (IIRA) and unaligned 64-bit memory access bits in the STKYx registers.
3	SOVFI	Stack Overflow/Full Interrupt. Indicates if a SOVFI is latched and is pending (if set, = 1), or no SOVFI is pending (if cleared, = 0). An SOVFI occurs when a stack in the program sequencer overflows or is full.
4	TMZHI	<p>Timer Expired High Priority. Indicates if a TMZHI is latched and is pending (if set, = 1), or TMZHI is not pending (if cleared, = 0). A TMZHI occurs when the timer decrements to zero. Note that this event also triggers a TMZLI. The timer operations are controlled as follows:</p> <ul style="list-style-type: none"> • The TCOUNT register contains the timer counter. The timer decrements the TCOUNT register each clock cycle. • The TPERIOD value specifies the frequency of timer interrupts. The number of cycles between interrupts is TPERIOD + 1. The maximum value of TPERIOD is $2^{32} - 1$. • The TIMEN bit in the MODE2 register starts and stops the timer. <p>Since the timer expired event (TCOUNT decrements to zero) generates two interrupts, TMZHI and TMZLI, programs should unmask the timer interrupt with the desired priority and leave the other one masked.</p>
5	SPERRI	SP Error Interrupt.
6	BKPI	Hardware Breakpoint Interrupt. Indicates if an BKPI is latched and is pending (if set, = 1), or no BKPI is pending (if cleared, = 0).
7	Reserved	
8	IRQ2I	IRQ2 Hardware Interrupt. Indicates if an $\overline{\text{IRQ2I}}$ is latched and is pending (if set, = 1), or no IRQ2I is pending (if cleared, = 0). An IRQ2I occurs when an external device asserts the FLAG2 pin configured as $\overline{\text{IRQ2}}$.

Interrupt Registers

Table B-5. Interrupt Latch (IRPTL) Register Bit Descriptions (Cont'd)

Bit	Name	Description
9	$\overline{\text{IRQ11}}$	$\overline{\text{IRQ1}}$ Hardware Interrupt. Indicates if an $\overline{\text{IRQ11}}$ is latched and is pending (if set, = 1), or no $\overline{\text{IRQ11}}$ is pending (if cleared, = 0). An $\overline{\text{IRQ11}}$ occurs when an external device asserts the FLAG1 pin configured as $\overline{\text{IRQ1}}$.
10	$\overline{\text{IRQ01}}$	$\overline{\text{IRQ0}}$ Hardware Interrupt. Indicates if an $\overline{\text{IRQ01}}$ is latched and is pending (if set, = 1), or no $\overline{\text{IRQ01}}$ is pending (if cleared, = 0). An $\overline{\text{IRQ01}}$ occurs when an external device asserts the FLAG0 pin configured as $\overline{\text{IRQ0}}$.
11	P0I (DAI1I)	Programmable Interrupt 0 (DAI High Priority Interrupt). Indicates if a DAI1I interrupt is latched and is pending (if set, = 1), or no DAI1I interrupt is pending (if cleared, = 0). This is the higher priority option.
12	P1I (SPIAI)	Programmable Interrupt 1 (SPI Transmit or Receive High Priority Interrupt). Indicates if an interrupt in the primary SPIAI is latched and is pending (if set, = 1), or no interrupt is pending (if cleared, = 0). This is the higher priority option.
13	P2I (GPTMR0I)	Programmable Interrupt 2 (General-Purpose IOP Timer 0 Interrupt). Indicates if a GPTMR0I is latched and is pending (if set, = 1), or no GPTMR0I is pending (if cleared, = 0).
14	P3I (SP1I)	Programmable Interrupt 3 (SPORT 1 Interrupt). Indicates if an SP1I interrupt is latched and is pending (if set, = 1), or no SP1I is pending (if cleared, = 0). An SP1I interrupt occurs two cycles after the last bit of an input/output serial word is latched into/from RXSP1A/TXSP1A, RXSP1B/TXSP1B.
15	P4I (SP3I)	Programmable Interrupt 4 (SPORT 3 Interrupt). Indicates if an SP3I interrupt is latched and is pending (if set, = 1), or no SP3I is pending (if cleared, = 0). An SP3I interrupt occurs two cycles after the last bit of an input/output serial word is latched into/from RXSP3A/TXSP3A, RXSP3B/TXSP3B.
16	P5I (SP5I)	Programmable Interrupt 5 (SPORT 5 Interrupt). Indicates if an SP5I interrupt is latched and is pending (if set, = 1), or no SP5I is pending (if cleared, = 0). An SP5I interrupt occurs two cycles after the last bit of an input/output serial word is latched into/from RXSP5A/TXSP5A, RXSP5B/TXSP5B.

Table B-5. Interrupt Latch (IRPTL) Register Bit Descriptions (Cont'd)

Bit	Name	Description
17	P14I (DPI)	Programmable Interrupt 14. Indicates if a DPI interrupt is latched and is pending (if set, = 1), or no DPI interrupt is pending (if cleared, = 0).
18	P15I (MTMDMA)	Programmable Interrupt 15 (MTMDMA Interrupt).
19	P16I (SP6I)	Programmable Interrupt 16. Indicates if an SP6I interrupt is latched and is pending (if set, = 1), or no SP6I is pending (if cleared, = 0). An SP6I interrupt occurs two cycles after the last bit of an input/output serial word is latched into/from RXSP6A/TXSP6A, RXSP6B/TXSP6B.
20	CB7I	DAG1 Circular Buffer 7 Overflow Interrupt. Indicates if a CB7I is latched and is pending (if set, = 1), or no CB7I interrupt is pending (if cleared, = 0). A circular buffer overflow occurs when the DAG circular buffering operation increments the I register past the end of the buffer.
21	CB15I	DAG2 Circular Buffer 15 Overflow Interrupt. Indicates if a CB15I is latched and is pending (if set, = 1), or no CB15I is pending (if cleared, = 0). A circular buffer overflow occurs when the DAG circular buffering operation increments the I register past the end of the buffer.
22	TMZLI	Timer Expired (Low Priority) Interrupt. Indicates if a TMZLI is latched and is pending (if set, = 1), or no TMZLI is pending (if cleared, = 0). For more information, see “TMZHI” on page B-15.
23	FIXI	Fixed-Point Overflow Interrupt. Indicates if a FIXI is latched and is pending (if set, = 1), or no FIXI is pending (if cleared, = 0).
24	FLTOI	Floating-Point Overflow Interrupt. Indicates if a FLTOI is latched and is pending (if set, = 1), or no FLTOI is pending (if cleared, = 0).
25	FLTUI	Floating-Point Underflow Interrupt. Indicates if a FLTUI is latched and is pending (if set, = 1), or no FLTUI is pending (if cleared, = 0).
26	FLTII	Floating-Point Invalid Operation Interrupt. This bit indicates if a FLTII is latched and is pending (if set, = 1), or no FLTII is pending (if cleared, = 0).

Interrupt Registers

Table B-5. Interrupt Latch (IRPTL) Register Bit Descriptions (Cont'd)

Bit	Name	Description
27	EMULI	Emulator (Lower Priority) Interrupt. Indicates if an EMULI is latched and is pending (if set, = 1), or no EMULI is pending (if cleared, = 0). An EMULI occurs on reset and when an external device asserts the $\overline{\text{EMU}}$ pin. This interrupt has a lower priority than EMULI, but higher priority than software interrupts.
28	SFT0I	User Software Interrupt 0. Indicates if a SFT0I is latched and is pending (if set, = 1), or no SFT0I is pending (if cleared, = 0). An SFT0I interrupt occurs when a program sets (= 1) this bit.
29	SFT1I	User Software Interrupt 1. Indicates if a SFT1I is latched and is pending (if set, = 1), or no SFT1I is pending (if cleared, = 0). An SFT1I interrupt occurs when a program sets (= 1) this bit.
30	SFT2I	User Software Interrupt 2. Indicates if a SFT2I is latched and is pending (if set, = 1), or no SFT2I is pending (if cleared, = 0). An SFT2I interrupt occurs when a program sets (= 1) this bit.
31	SFT3I	User Software Interrupt 3. Indicates if a SFT3I is latched and is pending (if set, = 1), or no SFT3I is pending (if cleared, = 0). An SFT3I interrupt occurs when a program sets (= 1) this bit.

Interrupt Mask Register (IMASK)

The IMASK register is a non-memory-mapped, universal, system register (*Ureg* and *Sreg*). The reset value for this register is 0x0000 0003. Each bit in the IMASK register corresponds to a bit with the same name in the IRPTL registers. The bits in the IMASK register unmask (enable if set, = 1), or mask (disable if cleared, = 0) the interrupts that are latched in the IRPTL register. Except for $\overline{\text{RSTI}}$ and EMULI, all interrupts are maskable.

When the IMASK register masks an interrupt, the masking disables the processor's response to the interrupt. The IRPTL register still latches an interrupt even when masked, and the processor responds to that latched interrupt if it is later unmasked. [Figure B-3](#), [Figure B-4](#) and [Table B-6](#) provide bit definitions for the IMASK register.

Table B-6. IMASK Register Bit Descriptions

Bit	Name	Description
0	EMUI	Emulator Interrupt. This bit is set to 1 (unmasked). An EMUI occurs on reset and when an external device asserts the $\overline{\text{EMU}}$ pin.
1	RSTI	Reset Interrupt. This bit is set to 1 (unmasked). An RSTI occurs on reset as an external device asserts the $\overline{\text{RESET}}$ pin.
2	IICDI	Illegal Input Condition Detected Interrupt. Unmasks the IICDI interrupt (if set, = 1), or masks (if cleared, = 0). An IICDI occurs when a true results from the logical ORing of the illegal I/O processor register access (IIRA) and unaligned 64-bit memory access bits in the STKYx registers.
3	SOVFI	Stack Overflow/Full Interrupt. Unmasks the SOVFI interrupt (if set, = 1), or masks the SOVFI interrupt (if cleared, = 0). An SOVFI occurs when a stack in the program sequencer overflows or is full.
4	TMZHI	<p>Timer Expired High Priority. Unmasks the TMZHI interrupt (if set, = 1), or masks the TMZHI interrupt (if cleared, = 0). A TMZHI occurs when the timer decrements to zero. Note that this event also triggers a TMZLI. The timer operations are controlled as follows:</p> <ul style="list-style-type: none"> • The TCOUNT register contains the timer counter. The timer decrements the TCOUNT register each clock cycle. • The TPERIOD value specifies the frequency of timer interrupts. The number of cycles between interrupts is $\text{TPERIOD} + 1$. The maximum value of TPERIOD is $2^{32} - 1$. • The TIMEN bit in the MODE2 register starts and stops the timer. <p>Since the timer expired event (TCOUNT decrements to zero) generates two interrupts, TMZHI and TMZLI, programs should unmask the timer interrupt with the desired priority and leave the other one masked.</p>
5	Reserved	
6	BKPI	Hardware Breakpoint Interrupt. Unmasks the BKPI interrupt (if set, = 1), or masks the BKPI interrupt (if cleared, = 0).
7	Reserved	
8	IRQ2I	$\overline{\text{IRQ2}}$ Hardware Interrupt. Unmasks the $\overline{\text{IRQ2I}}$ interrupt (if set, = 1), or masks the interrupt (if cleared, = 0). An $\overline{\text{IRQ2I}}$ occurs when an external device asserts the FLAG2 pin configured as $\overline{\text{IRQ2}}$.

Interrupt Registers

Table B-6. IMASK Register Bit Descriptions (Cont'd)

Bit	Name	Description
9	IRQ1I	$\overline{\text{IRQ1}}$ Hardware Interrupt. Unmasks the $\overline{\text{IRQ1I}}$ interrupt (if set, = 1), or masks the $\overline{\text{IRQ1I}}$ interrupt (if cleared, = 0). An $\overline{\text{IRQ1I}}$ occurs when an external device asserts the FLAG1 pin configured as $\overline{\text{IRQ1}}$.
10	IRQ0I	$\overline{\text{IRQ0}}$ Hardware Interrupt. Unmasks the IRQ0I interrupt (if set, = 1), or masks the $\overline{\text{IRQ0I}}$ interrupt (if cleared, = 0). An $\overline{\text{IRQ0I}}$ occurs when an external device asserts the FLAG0 pin configured as $\overline{\text{IRQ0}}$.
11	DAI1I	DAI High Priority Interrupt. Unmasks the DAI1I interrupt (if set, = 1), or masks the DAI1I interrupt (if cleared, = 0). This is the higher priority option.
12	SPIAI	SPI Transmit or Receive High Priority Interrupt. Unmasks the SPIAI interrupt (if set, = 1), or masks the SPIAI interrupt (if cleared, = 0). This is the higher priority option.
13	GPTMR0I	General-Purpose IOP Timer 0 Interrupt. Unmasks the GPTMR0I interrupt (if set, = 1), or masks the GPTMR0I interrupt (if cleared, = 0).
14	SP1I	SPORT 1 Interrupt. Unmasks the SP1I interrupt (if set, = 1), or masks the SP1I interrupt (if cleared, = 0). An SP1I interrupt occurs two cycles after the last bit of an input/output serial word is latched into/from RXSP1A/TXSP1A, or RXSP1B/TXSP1B.
15	SP3I	SPORT 3 Interrupt. Unmasks the SP3I interrupt (if set, = 1), or masks the SP3I interrupt (if cleared, = 0). An SP3I interrupt occurs two cycles after the last bit of an input/output serial word is latched into/from RXSP3A/TXSP3A, or RXSP3B/TXSP3B.
16	SP5I	SPORT 5 Interrupt. Unmasks the SP5I interrupt (if set, = 1), or masks the SP5I interrupt (if cleared, = 0). An SP5I interrupt occurs two cycles after the last bit of an input/output serial word is latched into/from RXSP5A/TXSP5A, or RXSP5B/TXSP5B.
17	Reserved	
18	P15I	Programmable Interrupt 15 (MTMDMA Interrupt).
19	Reserved	

Table B-6. IMASK Register Bit Descriptions (Cont'd)

Bit	Name	Description
20	CB7I	DAG1 Circular Buffer 7 Overflow Interrupt. Unmasks the CB7I interrupt (if set, = 1), or masks the CB7I interrupt (if cleared, = 0). A circular buffer overflow occurs when the DAG circular buffering operation increments the I register past the end of the buffer.
21	CB15I	DAG2 Circular Buffer 15 Overflow Interrupt. Unmasks the CB15I interrupt (if set, = 1), or masks the CB15I interrupt (if cleared, = 0). A circular buffer overflow occurs when the DAG circular buffering operation increments the I register past the end of the buffer.
22	TMZLI	Timer Expired (Low Priority) Interrupt. Unmasks the TMZLI interrupt (if set, = 1), or masks the TMZLI interrupt (if cleared, = 0). For more information, see “TMZHI” on page B-19.
23	FIXI	Fixed-Point Overflow Interrupt. Unmasks the FIXI interrupt (if set, = 1), or masks the FIXI interrupt (if cleared, = 0).
24	FLTOI	Floating-Point Overflow Interrupt. Unmasks the FLTOI interrupt (if set, = 1), or masks the FLTOI interrupt (if cleared, = 0).
25	FLTUI	Floating-Point Underflow Interrupt. Unmasks the FLTUI interrupt (if set, = 1), or masks the FLTUI interrupt (if cleared, = 0).
26	FLTII	Floating-Point Invalid Operation Interrupt. Unmasks the FLTII interrupt (if set, = 1), or masks the FLTII interrupt (if cleared, = 0).
27	EMULI	Emulator (Lower Priority) Interrupt. Unmasks the EMULI interrupt (if set, = 1), or masks the EMULI interrupt (if cleared, = 0). An EMULI occurs on reset and when an external device asserts the $\overline{\text{EMU}}$ pin. This interrupt has a lower priority than EMUI, but higher priority than software interrupts.
28	SFT0I	User Software Interrupt 0. Unmasks the SFT0I interrupt (if set, = 1), or masks the SFT0I interrupt (if cleared, = 0). An SFT0I occurs when a program sets (= 1) this bit.
29	SFT1I	User Software Interrupt 1. Unmasks the SFT1I interrupt (if set, = 1), or masks the SFT1I interrupt (if cleared, = 0).
30	SFT2I	User Software Interrupt 2. Unmasks the SFT2I interrupt (if set, = 1), or masks the SFT2I interrupt (if cleared, = 0).
31	SFT3I	User Software Interrupt 3. Unmasks the SFT3I interrupt (if set, = 1), or masks the SFT3I interrupt (if cleared, = 0).

Interrupt Mask Pointer Register (IMASKP)

The IMASKP register is a non-memory-mapped, universal, system register (*Ureg* and *Sreg*). The reset value for this register is 0x0000 0000. Each bit in the IMASKP register corresponds to a bit with the same name in the IRPTL registers. The IMASKP register field descriptions are shown in [Figure B-3](#) and [Figure B-4](#) and described in [Table B-7](#).

This register supports an interrupt nesting scheme that lets higher priority events interrupt an interrupt service routine (ISR) and keeps lower priority events from interrupting.

When interrupt nesting is enabled, the bits in the IMASKP register mask interrupts with lower priorities than the interrupt that is currently being serviced. Other bits in this register unmask interrupts having higher priority than the interrupt that is currently being serviced. Interrupt nesting is enabled using NESTM in the MODE1 register. The IRPTL register latches a lower priority interrupt even when masked, and the processor responds to that latched interrupt if it is later unmasked.

When interrupt nesting is disabled (NESTM = 0 in the MODE1 register), the bits in the IMASKP register mask all interrupts while an interrupt is currently being serviced. The IRPTL register still latches these interrupts even when masked, and the processor responds to the highest priority latched interrupt after servicing the current interrupt.

Table B-7. IMASKP Register Bit Descriptions

Bit	Name	Description
0	EMUI	Emulator Interrupt. When the processor is servicing another interrupt, this bit indicates if the EMUI interrupt is unmasked (if set, = 1), or masked (if cleared, = 0). An EMUI occurs on reset and when an external device asserts the $\overline{\text{EMU}}$ pin.
1	RSTI	Reset Interrupt. When the processor is servicing another interrupt, this bit indicates if the RSTI interrupt is unmasked (if set, = 1), or masked (if cleared, = 0). An RSTI occurs on reset as an external device asserts the $\overline{\text{RESET}}$ pin.
2	IICDI	Illegal Input Condition Detected Interrupt. When the processor is servicing another interrupt, this bit indicates if the IICDI interrupt is unmasked (if set, = 1), or masked (if cleared, = 0). An IICDI occurs when a TRUE results from the logical ORing of the illegal I/O processor register access (IIRA) and unaligned 64-bit memory access bits in the STKYx registers.
3	SOVFI	Stack Overflow/Full Interrupt. When the processor is servicing another interrupt, this bit indicates if the SOVFI interrupt is unmasked (if set, = 1), or masked (if cleared, = 0). A SOVFI occurs when a stack in the program sequencer overflows or is full.
4	TMZHI	<p>Timer Expired High Priority. When the processor is servicing another interrupt, this bit indicates if the TMZHI interrupt is unmasked (if set, = 1), or masked (if cleared, = 0). A TMZHI occurs when the timer decrements to zero. Note that this event also triggers a TMZLI. Timer operations are controlled as follows:</p> <ul style="list-style-type: none"> • The TCOUNT register contains the timer counter. The timer decrements the TCOUNT register each clock cycle. • The TPERIOD value specifies the frequency of timer interrupts. The number of cycles between interrupts is $\text{TPERIOD} + 1$. The maximum value of TPERIOD is $2^{32} - 1$. • The TIMEN bit in the MODE2 register starts and stops the timer. <p>Since the timer expired event (TCOUNT decrements to zero) generates two interrupts, TMZHI and TMZLI, programs should unmask the timer interrupt with the desired priority and leave the other one masked.</p>
5	Reserved	

Interrupt Registers

Table B-7. IMASKP Register Bit Descriptions (Cont'd)

Bit	Name	Description
6	BKPI	Hardware Breakpoint Interrupt. When the processor is servicing another interrupt, this bit indicates if the BKPI interrupt is unmasked (if set, = 1), or masked (if cleared, = 0).
7	Reserved	
8	IRQ2I	$\overline{\text{IRQ2}}$ Hardware Interrupt. When the processor is servicing another interrupt, this bit indicates if the IRQ2I interrupt is unmasked (if set, = 1), or masked (if cleared, = 0). An IRQ2I occurs when an external device asserts the FLAG2 pin configured as $\overline{\text{IRQ2}}$.
9	IRQ1I	$\overline{\text{IRQ1}}$ Hardware Interrupt. When the processor is servicing another interrupt, this bit indicates if the IRQ1I interrupt is unmasked (if set, = 1), or masked (if cleared, = 0). An IRQ1I occurs when an external device asserts the FLAG1 pin configured as $\overline{\text{IRQ1}}$.
10	IRQ0I	$\overline{\text{IRQ0}}$ Hardware Interrupt. When the processor is servicing another interrupt, this bit indicates if the IRQ0I interrupt is unmasked (if set, = 1), or masked (if cleared, = 0). An IRQ0I occurs when an external device asserts the FLAG0 pin configured as $\overline{\text{IRQ0}}$.
11	DAI1I	DAI High Priority Interrupt. When the processor is servicing another interrupt, this bit indicates if the DAI1I interrupt is unmasked (if set, = 1), or masked (if cleared, = 0). This is the higher priority option.
12	SPIAI	SPI Transmit or Receive High Priority Interrupt. When the processor is servicing another interrupt, this bit indicates if the SPIAI interrupt is unmasked (if set, = 1), or the SPIAI interrupt is masked (if cleared, = 0). This is the higher priority option.
13	GPTMR0I	General-Purpose IOP Timer 0 Interrupt. When the processor is servicing another interrupt, this bit indicates if the GPTMR0I interrupt is unmasked (if set, = 1), or masked (if cleared, = 0).
14	SP1I	SPORT 1 Interrupt. When the processor is servicing another interrupt, this bit indicates if the SP1I interrupt is unmasked (if set, = 1), or masked (if cleared, = 0). An SP1I interrupt occurs two cycles after the last bit of an input/output serial word is latched into/from RXSP1A/TXSP1A, or RXSP1B/TXSP1B.

Table B-7. IMASKP Register Bit Descriptions (Cont'd)

Bit	Name	Description
15	SP3I	SPORT 3 Interrupt. When the processor is servicing another interrupt, this bit indicates if the SP3I interrupt is unmasked (if set, = 1), or masked (if cleared, = 0). An SP3I interrupt occurs two cycles after the last bit of an input/output serial word is latched into/from RXSP3A/TXSP3A, or RXSP3B/TXSP3B.
16	SP5I	SPORT 5 Interrupt. When the processor is servicing another interrupt, this bit indicates if the SP5I interrupt is unmasked (if set, = 1), or masked (if cleared, = 0). An SP5I interrupt occurs two cycles after the last bit of an input/output serial word is latched into/from RXSP5A/TXSP5A, RXSP5B/TXSP5B.
17	Reserved	
18	P15I	Programmable Interrupt 15 (MTMDMA Interrupt).
19	Reserved	
20	CB7I	DAG1 Circular Buffer 7 Overflow Interrupt. When the processor is servicing another interrupt, this bit indicates if the CB7I interrupt is unmasked (if set, = 1), or masked (if cleared, = 0). A circular buffer overflow occurs when the DAG circular buffering operation increments the I register past the end of the buffer.
21	CB15I	DAG2 Circular Buffer 15 Overflow Interrupt. When the processor is servicing another interrupt, this bit indicates if the CB15I interrupt is unmasked (if set, = 1), or masked (if cleared, = 0). A circular buffer overflow occurs when the DAG circular buffering operation increments the I register past the end of the buffer.
22	TMZLI	Timer Expired (Low Priority) Interrupt. When the processor is servicing another interrupt, this bit indicates if the TMZLI interrupt is unmasked (if set, = 1), or masked (if cleared, = 0). For more information, see “TMZHI” on page B-15.
23	FIXI	Fixed-Point Overflow Interrupt. When the processor is servicing another interrupt, this bit indicates if the FIXI interrupt is unmasked (if set, = 1), or the FIXI interrupt is masked (if cleared, = 0).
24	FLTOI	Floating-Point Overflow Interrupt. When the processor is servicing another interrupt, this bit indicates if the FLTOI interrupt is unmasked (if set, = 1), or masked (if cleared, = 0).

Interrupt Registers

Table B-7. IMASKP Register Bit Descriptions (Cont'd)

Bit	Name	Description
25	FLTUI	Floating-Point Underflow Interrupt. When the processor is servicing another interrupt, this bit indicates if the FLTUI interrupt is unmasked (if set, = 1), or masked (if cleared, = 0).
26	FLTII	Floating-Point Invalid Operation Interrupt. When the processor is servicing another interrupt, this bit indicates if the FLTII interrupt is unmasked (if set, = 1), or masked (if cleared, = 0).
27	EMULI	Emulator (Lower Priority) Interrupt. When the processor is servicing another interrupt, this bit indicates if the EMULI interrupt is unmasked (if set, = 1), or masked (if cleared, = 0). An EMULI occurs on reset and when an external device asserts the $\overline{\text{EMU}}$ pin. This interrupt has a lower priority than EMUI, but higher priority than software interrupts.
28	SFT0I	User Software Interrupt 0. When the processor is servicing another interrupt, this bit indicates if the SFT0I interrupt is unmasked (if set, = 1), or masked (if cleared, = 0). An SFT0I occurs when a program sets (= 1) this bit.
29	SFT1I	User Software Interrupt 1. When the processor is servicing another interrupt, this bit indicates if the SFT1I interrupt is unmasked (if set, = 1), or masked (if cleared, = 0).
30	SFT2I	User Software Interrupt 2. When the processor is servicing another interrupt, this bit indicates if the SFT2I interrupt is unmasked (if set, = 1), or masked (if cleared, = 0).
31	SFT3I	User Software Interrupt 3. When the processor is servicing another interrupt, this bit indicates if the SFT3I interrupt is unmasked (if set, = 1), or masked (if cleared, = 0).

I INDEX

Numerics

- 128-channel TDM, [5-4](#)
- 16-bit to 32-bit word packing enable (PACK), [5-62](#)
- 16-bit word lengths, [5-45](#), [6-32](#), [7-14](#)
- 32-bit word lengths, [5-45](#), [6-32](#), [7-14](#)
- 8-bit word lengths, [6-31](#)

A

- accessing IOP registers, latency in, [2-3](#)
- accuracy, PWM, [8-17](#)
- ACK (acknowledge) signal, [3-2](#), [3-19](#), [3-21](#)
- acknowledge (ACK) pin, [3-20](#)
- activate command, bank, [3-31](#)
- active low frame sync select for frame sync (INVSx) bit, [13-12](#)
- active low versus active high frame syncs, [5-39](#)
- active state multichannel receive frame sync select (LMFS) bit, [5-29](#)
- AD1855 stereo DAC
 - power down, [6-7](#)
- address
 - column, row and bank address mapping (32-bit), [3-53](#)
 - core to external memory, [3-52](#)
 - decoding address bank, [3-53](#)
 - row in SDRAM, [3-34](#)
 - SDRAM (external memory space), [3-52](#)
 - SPORT IOP (listing), [5-50](#)
- address bus (ADDR) pin, [3-20](#), [3-83](#)
- addressing, [14-53](#)
 - 7-bit in TWI, [12-1](#), [12-15](#)
 - general call in TWI, [12-14](#)
 - IOP, [2-29](#)
 - pre-modify, [2-39](#)
 - restrictions on external memory, [3-3](#)
 - transfer phase in TWI, [12-6](#)
- AMI
 - See also* external port, SDRAM controller, shared memory
 - ADDR23-0 bits, [3-28](#)
 - control (AMICTLx) register, [3-25](#), [A-17](#) to [A-19](#)
 - DMA, [3-20](#)
 - hold cycles, [3-23](#)
 - idle cycle, [3-22](#)
 - memory bank support, [3-28](#)
 - modes, setting, [3-24](#)
 - most significant word first (MSWF) bit, [3-25](#)
 - packing data (PKDIS) bit, [3-25](#)
 - read/write throughput, [3-28](#)
 - reading external memory, [3-25](#)
 - receive (AMIRX) register, [3-25](#)
 - signals, [3-20](#)
 - status (AMISTAT) register, [A-20](#)
 - unpacking data, [3-27](#)
 - wait states, [3-21](#)
 - writing external memory, [3-26](#)

Index

AMI bits

- ACK pin enable (ACKEN), [A-18](#)
- AMI enable (AMIEN), [A-18](#)
- buffer flush (FLSH), [A-19](#)
- bus hold cycle (HC), [A-18](#)
- bus idle cycle (HC), [A-19](#)
- disable packing (PKDIS), [A-18](#)
- external bus data width (BW), [A-18](#)
- most significant word first (MSWF), [A-18](#)
- predictive read disable (NO_OPT), [A-19](#)
- read hold cycle (RHC), [A-19](#)
- wait state enable (WS), [A-18](#)

AND breakpoints (ANDBKP) bit, [A-178](#)

AND, logical, [2-43](#), [14-3](#), [A-178](#)

architecture

- I/O processor, [2-25](#)
- PWM, [8-2](#)
- SPDIF transmitter, [9-8](#)
- SPORT, [5-8](#)
- TWI controller, [12-3](#)

asynchronous access mode (external memory), [3-81](#)

asynchronous memory interface. *See* AMI

asynchronous serial communications (UART), [11-2](#)

audience, intended, [xxxi](#)

autobaud detection, [11-1](#)

B

bandwidth in the I/O processor, [2-12](#)

bank

- internal memory, [3-33](#)
- SDRAM address mapping, [3-51](#) to [3-57](#)

bank activate command, [3-31](#)

bank column address width, setting, [3-42](#)

baud rate, [6-30](#), [14-49](#)

- setting, [2-43](#), [6-10](#)

baud rate *(continued)*

- SPIBAUD (serial peripheral interface baud rate) register, [6-5](#), [A-60](#)
- UART, [11-4](#), [11-5](#), [11-12](#)

beginning and ending an SPI transfer, [6-29](#)

BHD (buffer hang disable) bit, [5-64](#), [5-69](#)

BI (break interrupt) bit, [11-4](#)

bidirectional connections through the signal routing unit, [4-13](#)

bidirectional functions (transmit, receive), [5-1](#)

biphase

- encoded audio stream, [9-10](#)
- encoded data register (SPDIF_RX_I), [9-18](#)
- encoding, [9-11](#)

bits

- See also* peripheral specific bits
- circular buffer x overflow interrupt (CBxI), [B-17](#), [B-21](#), [B-25](#)
- emulator lower priority interrupt (EMUI), [B-14](#), [B-18](#), [B-19](#), [B-21](#), [B-23](#), [B-26](#)
- timer expired high priority (TMZHI), [B-15](#), [B-19](#), [B-23](#)
- timer expired low priority (TMZLI), [B-17](#), [B-21](#), [B-25](#)

block diagram

- I/O processor, [2-25](#)
- input data port, [7-2](#)
- PWM, [8-2](#)
- SPDIF transmitter, [9-8](#)
- SPORT, [5-8](#)
- SRC, [10-9](#)
- TWI controller, [12-3](#)

boolean operator

- AND, [2-43](#), [14-3](#), [A-178](#)
- OR, [7-30](#), [9-24](#), [A-175](#), [A-177](#), [A-178](#)

boot memory select ($\overline{\text{BMS}}$) pin, [3-30](#), [14-39](#)

- booting
 - boot kernel, [14-37](#)
 - BOOT_CFGx (boot source configuration) pins, [14-38](#)
 - bootstrap loading, [14-37](#)
 - from SPI flash, [14-42](#)
 - SPI port, [14-42](#)
 - broadcast mode, [6-3](#), [6-8](#)
 - buffer, [2-39](#), [5-7](#)
 - chain pointer load sequence, [2-38](#)
 - circular, [2-37](#)
 - DAI pin, [4-3](#)
 - data buffer registers, [2-25](#)
 - data, addressing, [2-29](#)
 - external, [3-75](#)
 - external register, [3-45](#)
 - flash (AMI boot mode), [14-39](#)
 - output example, [4-12](#)
 - pin, defined, example, [4-10](#)
 - SPORT data, [5-1](#)
 - SPORT DMA, [5-15](#)
 - SPORT transmit and receive, [5-61](#)
 - SPORTs, activating, [5-4](#)
 - system example (multiple SDRAM), [3-46](#), [3-47](#)
 - UART restriction, [2-47](#)
 - buffer hang disable (BHD) bit, [5-64](#), [5-69](#), [A-39](#)
 - burst length, in SDRAM, [3-32](#)
 - burst stop command (SDRAM), [3-32](#)
 - burst type definition, [3-32](#)
 - bus lock (BUSLK) bit, [3-88](#), [3-92](#)
 - bus lock and semaphores, [3-92](#)
 - bus master (Bm) condition, [3-82](#), [3-93](#)
 - bus master count (BCNT) register, [3-87](#)
 - bus master max time-out (BMAX) register, [3-87](#)
 - bus master, current (CRBMx) bit, [3-93](#), [A-9](#)
 - bus request $\overline{\text{BRx}}$ signal, [3-85](#)
 - bus request, shared memory ($\overline{\text{BRx}}$) pins, [3-81](#), [3-92](#)
 - bus synchronized (BSYN) bit, [3-89](#), [3-93](#), [A-9](#)
 - bus transition cycle (BTC), [3-82](#)
 - buses
 - arbitration, [2-20](#), [3-79](#), [3-82](#)
 - bus lock, [3-92](#)
 - bus lock (BUSLK) bit, [A-8](#)
 - bus master
 - timeout, [3-87](#)
 - conflict resolution ratio, [3-25](#)
 - contention, [2-20](#), [6-17](#), [A-3](#)
 - errors in, [3-71](#)
 - external bus data width (BW) bit, [A-18](#)
 - force sync of shared memory bus (FSYNC) bit, [A-8](#)
 - granting, [2-21](#), [6-20](#)
 - hold cycle bit, [A-18](#)
 - I/O address (IOA), [2-29](#)
 - I/O data (IOD), [2-19](#), [2-20](#), [6-17](#)
 - I/O processor (IOP), [2-25](#), [5-44](#)
 - I²S and, [5-20](#)
 - idle cycle bit, [A-19](#)
 - master, [3-84](#)
 - master timeout, [3-87](#)
 - serial, [5-27](#)
 - shared memory bus arbitration, [3-79](#)
 - slave, [3-84](#)
 - stalls on, [14-54](#)
 - synchronization, [3-88](#)
 - TDM method over serial, [5-27](#)
 - bypass as a one-shot (strobe pulse), [13-13](#)
- ## C
- capacitors
 - bypass, [14-35](#)
 - decoupling, [14-35](#)

Index

- CAS latency
 - bit (SDCL), [A-22](#)
 - definition, [3-32](#)
 - setting, [3-40](#)
- catastrophic interrupts, [4-65](#)
- CBR (CAS before RAS) definition, [3-33](#)
- CBxI (circular buffer x overflow interrupt)
 - bit, [B-17](#), [B-21](#), [B-25](#)
- center-aligned paired PWM
 - double-update mode, [8-11](#)
 - single-update mode, [8-9](#)
- chain loading sequence, [2-18](#)
- chain pointer registers, [2-15](#), [2-16](#)
 - defined, [2-27](#)
 - DMA buffer, [5-76](#)
 - SPI address in memory (CPSPI), [2-42](#), [6-16](#), [6-19](#)
 - SPORT address in memory (CPSPx), [A-51](#)
 - SPORTs, [5-77](#), [5-81](#), [A-51](#)
 - starting address, [2-27](#)
- chained DMA, [2-14](#)
- chaining, [2-13](#) to [2-42](#)
 - chain insertion mode, [2-41](#)
 - chained DMA enable (SCHEN_A and SCHEN_B) bit, [5-22](#), [5-63](#), [5-75](#), [5-81](#), [A-38](#)
 - chained DMA sequences, [2-14](#)
 - DMA, [2-13](#) to [2-42](#), [6-27](#)
 - in serial ports, [5-81](#), [A-38](#)
 - SPI chained DMA enable (SPICHEN) bit, [6-19](#)
- chaining requests, multiple, [2-18](#)
- changing SPI configuration, [6-21](#)
- channel
 - buffer, [2-7](#)
 - DMA, [2-9](#), [2-12](#), [2-13](#)
 - interrupt, [2-8](#)
 - priority scheme, [2-2](#)
 - status, [2-7](#), [2-13](#)
 - channel B transmit status register (SPDIF_TX_CHSTB), [A-90](#), [A-91](#)
 - channel double frequency mode, single, [9-8](#)
 - channel mode (SPDIF), two, [9-8](#)
 - channel number, encoded, [7-19](#)
 - channel selection registers, [5-31](#)
 - channels, input data port, [7-1](#)
 - circular, [2-39](#)
 - circular buffering, [2-37](#), [2-39](#)
 - CLKOUT (clock output) signal, [14-12](#)
 - clock A source (CLKASOURCE) bit, [A-157](#)
 - clock divisor (CLKDIV) bits, [A-44](#)
 - clock input (CLKIN) pin, [4-72](#), [13-2](#), [13-20](#), [14-20](#)
 - clock rising edge select (CKRE) bit, [5-62](#)
 - clocks and system clocking
 - CLKOUT and CCLK clock generation, [14-30](#)
 - clock and frame sync frequencies (DIVx) registers, [5-69](#), [A-44](#)
 - clock distribution, [14-34](#)
 - clock divisor (CLKDIV) bit, [A-44](#)
 - clock input (CLKIN) pin, [14-13](#), [14-20](#)
 - clock output enable, [13-3](#)
 - clock polarity (CLKPL) bit, [A-55](#)
 - clock ratio, [14-31](#)
 - clock relationships, [14-31](#)
 - clock rising edge select (CKRE) bit, [5-62](#), [A-38](#)
 - clock signal options, [5-71](#)
 - clocking edge selection, [7-12](#)
 - core clock ratio, [14-31](#)
 - definitions, [14-31](#)
 - determining switching frequencies, [14-29](#)
 - external master clock, [14-21](#)
 - frame sync bypass mode, [13-6](#)

clocks and system clocking *(continued)*
 frame sync bypass mode, direct bypass,
 13-6
 frame sync bypass mode, one shot, 13-6
 internal clock select (ICLK) bit, A-37
 jitter, 14-33
 master clock (MCLK), 10-8, 13-18
 output control (ENCLKA/B) bits, 13-3
 precision clock generator registers, 13-3,
 13-7
 SDRAM controller, 3-37
 serial port count (SPCNTx) registers,
 A-45
 source select (MSTR) bit, A-37
 SPI clock phase select (CPHASE) bit,
 A-55
 SPI clock rate, 6-5
 code select (CSEL) bit, 3-82
 column, row and bank address mapping
 (32-bit), 3-53
 commands
 auto-refresh, 3-70
 bank activate, 3-31, 3-65
 booting, 14-48, 14-51
 burst stop, 3-32
 load mode register, 3-64
 NOP, 3-72
 precharge, 3-34, 3-66
 read/write, 3-67, 14-50
 SDRAM read, A-22
 self-refresh, 3-70
 SPI master, 6-20
 SPI transfer, 6-10
 compand data in place, 5-48
 companding (compressing/expanding), 5-3
 conditioning input signals, 14-32
 configurable channels, digital audio
 interface interrupts, A-112
 configuring frame sync signals, 5-6
 connecting peripherals, 4-8

connections
 group A, DAI, clock signals, 4-19
 group A, DPI, input routing signals,
 4-52
 group B, DAI data signals, 4-25
 group B, DPI, pin assignment signals,
 4-56
 group C, DAI, frame sync signals, 4-31
 group C, DPI, pin enable signals, 4-64
 group D, DAI, pin signal assignments,
 4-36
 group E, DAI, miscellaneous signals,
 4-43
 group F, DAI, pin enable signals, 4-47
 controller, SDRAM, 3-30
 conventions, manual, xliii
 core access to IOP registers, 2-3
 core address mapping, 3-52
 core clock cycle, 6-30
 core PLL, 13-2
 core transmit/receive operations, 6-13
 count registers
 DMA sport (CSPx), A-51
 DMA, defined, 2-27
 DMA, restrictions, 2-31
 IDP DMA (IDP_DMA_Cx), 7-28
 CPSPI (SPI chain pointer) registers, A-65
 crossover mode, PWM, 8-16
 crosstalk, reducing, 14-34
 CSPI, CSPIB (SPI DMA word count)
 registers, A-64
 CSPx (peripheral DMA counter) registers,
 2-31, A-51
 customer support, xxxv

D

DAI
 buffers, pin, 4-3
 channel number, encoded, 7-19

Index

DAI *(continued)*
clock routing control registers (group A),
4-19
configurable interrupts, A-112
configuration macro, 4-77
connecting peripherals with, 4-8
default configuration, 4-18
general-purpose (GPIO) and flags, 4-64
interrupt controller, 4-65 to 4-71
interrupt controller registers, A-112
interrupts, 4-66, 7-22, A-112
latches, high and low priority, 4-69
miscellaneous signals, 4-65
pin buffer example, 4-3, 4-10, 4-12
pin buffer, bidirectional, 4-13
rising and falling edge masks, 4-70
selection group B (data), 4-25
selection group C (frame sync), 4-31
selection group D (pin assignments),
4-36
selection group E (miscellaneous signals),
4-43 to 4-46
selection group F, 4-47
SRU1 connections for SPORTx, 4-14,
4-16
system configuration, sample, 4-76
system design, 4-3
DAI registers
core interrupt priority assignment
(DAI_IRPTL_PRI), 4-69
core interrupt priority assignment
register (DAI_IRPTL_PRI), 7-19,
A-114
DAI_IRPTL_FE register
as replacement to IMASK, 4-69
DAI_IRPTL_H register as replacement
to IRPTL, 4-69
DAI_IRPTL_L register as replacement
to IRPTL, 4-69

DAI registers *(continued)*
DAI_IRPTL_RE register as replacement
to IMASK register, 4-69
falling edge interrupt mask register
(DAI_IRPTL_FE), A-115
high priority interrupt
(DAI_IRPTL_H), A-113
high priority interrupt latch register
(DAI_IRPTL_H), 7-17, 7-28, A-114
interrupt falling edge (DAI_IRPTL_FE),
7-19, A-113
interrupt high priority
(DAI_IRPTL_H), A-114
interrupt rising edge (DAI_IRPTL_RE),
7-19, A-113
low priority interrupt (DAI_IRPTL_L),
A-113
low priority interrupt latch register
(DAI_IRPTL_L), 7-17, A-114
pin status (DAI_PIN_STAT), A-112,
A-116
ping-pong DMA status
(SRU_PINGx_STAT), A-110
resistor pull up enable
(DAL_PIN_PULLUP), A-111, A-115
resistor pullup enable
(DAL_PIN_PULLUP), A-111, A-115
rising edge interrupt mask register
(DAI_IRPTL_RE), A-115
shadow high priority interrupt latch
register (DAI_IRPTL_HS), A-114
shadow low priority interrupt latch
register (DAI_IRPTL_LS), A-114
status (DAI_STAT), 7-19, 7-25, 7-26,
A-109
data
alignment, external port, 3-25
buffers in DMA registers, 2-32
direction control (SPTRAN) bit, 5-64,
A-39

- data *(continued)*
 - packing and unpacking, [5-45](#)
- data buffers, in serial ports, [5-4](#)
- data bus (DATA) pins, [3-20](#)
- data direction control (SPTRAN) bit, [5-64](#), [A-39](#)
- data fetch, external port, [3-25](#)
- data memory breakpoint hit (STATDx) bit, [A-181](#)
- data pin multiplexing, [A-8](#)
- data pins, function of, [14-7](#)
- data ready (DR) status flag (UART), [11-5](#)
- data status (DXS_A, DSX_B) bits, [5-66](#)
- data type, [5-46](#)
 - formatting (multichannel), [5-46](#)
 - formatting (non-multichannel), [5-46](#)
 - select (DTYPE) bit, [5-61](#), [A-37](#)
- data words
 - in FIFO, [7-15](#)
 - packing, [5-45](#), [14-42](#)
 - single word transfers, [5-81](#)
 - transferring, [2-49](#), [5-4](#), [5-17](#), [5-22](#), [5-31](#), [5-37](#), [6-33](#)
 - UART, [11-4](#)
- data-independent frame sync, [5-41](#)
 - (DIFS) mode, [5-41](#)
- dead time equation, [8-8](#)
- definition
 - burst type, SDRAM, [3-32](#)
 - CAS latency, SDRAM, [3-32](#)
 - CBR, SDRAM, [3-33](#)
- delay line DMA, [2-40](#)
- diagrams
 - uniprocessor system with multiple SDRAM devices diagram, [3-46](#), [3-47](#)
- DIFS (data independent frame sync select) bit, [A-38](#)
- digital audio interface. *See* DAI
- DIVEN (PLL divider enable) bit, [A-170](#), [A-172](#)
- division multiplexed (TDM) mode, time, [9-19](#)
- divisor
 - clock output, [13-3](#)
 - reset, UART, [11-12](#)
 - SPORT (DIVx) registers, [5-6](#), [A-44](#)
- divisor, UART, [11-11](#), [A-125](#)
- DIVx (divisor) registers, [5-6](#), [5-69](#), [A-44](#)
- DLAB (divisor latch access) bit, [11-6](#), [11-7](#), [A-119](#), [A-122](#), [A-123](#)
- DMA
 - See also* I/O processor, external port, SPI, SPORTs
 - active chain TCB, [2-41](#)
 - and UART, [2-44](#)
 - chain insertion mode, [2-41](#)
 - chain insertion, active chain, [2-41](#)
 - chaining in SPI, [2-42](#)
 - channel buffer registers, listed, [2-32](#)
 - channel parameter registers, [2-32](#)
 - channel priority, [2-19](#)
 - complete interrupt, [2-8](#)
 - configuring in the I/O processor, [2-2](#)
 - control registers, [2-32](#)
 - controller enhancements, [1-11](#)
 - count registers, [A-73](#)
 - data buffer registers, [2-25](#)
 - delay line, [2-40](#)
 - enable bits, [2-48](#)
 - enabling, [2-48](#)
 - error interrupts, [6-25](#)
 - external port throughput, [3-18](#)
 - IDP index registers, [A-70](#)
 - IDP modify registers, [A-71](#)
 - index registers, [2-29](#), [A-72](#)
 - input data port enable (IDP_DMA_EN) bit, [7-20](#)
 - interrupt regeneration, [2-11](#)
 - interrupt-driven, [2-8](#)
 - latency, [2-13](#)

Index

DMA *(continued)*

- master mode operation, [6-15](#)
- memory-to-memory, [2-48](#)
- multiple chain requests, [2-18](#)
- non-chained, [2-13](#)
- parameter registers, [2-17](#)
- ping-pong, [7-22](#) to [7-24](#)
- ping-pong enable (IDP_PING) bits, [A-68](#)
- sequence start, end, [2-14](#)
- sequences, TCB loading, [2-16](#)
- slave mode operation, [6-19](#)
- SPI chain pointer registers, [A-65](#)
- SPI count registers, [A-64](#)
- SPI modifier registers, [A-64](#)
- SPI slave mode, [6-11](#), [6-12](#)
- SPORT chain status bits (DMACHSxy), [A-43](#)
- SPORT index registers, [A-50](#)
- SPORT modify registers, [A-50](#)
- SPORT status bit (DMASxy), [A-43](#)
- switching from receive to transmit mode, [6-24](#)
- switching from transmit to receive mode, [6-23](#)
- transfers in IDP, [7-20](#)
- transmit or receive operations (SPI), [6-16](#)
- UART, [11-7](#)

DMA enable bits (all peripherals), [2-48](#)

DMA registers

- channel listed, [2-32](#)

DMACx (external port DMA registers), [A-14](#)

Dolby, DTS audio standards, [9-16](#)

double frequency mode, single channel, [9-8](#)

DPI

- connections, group A, [4-52](#) to [4-56](#)
- connections, group B, [4-56](#) to [4-60](#)
- connections, group C, [4-60](#) to [4-64](#)

DPI *(continued)*

- default configuration, [4-51](#)
- input routing (group A) signals, [4-52](#)
- interrupts, [4-67](#)
- pin assignment (group B) signals, [4-56](#)
- pin enable (group C) signals, [4-60](#)
- registers, [A-109](#)

DSP serial mode, [5-74](#)

DSxEN (SPI device select) bits, [6-15](#), [A-58](#)

DTYPE (data type) bits, [5-61](#), [A-37](#)

- DSP serial mode data formatting, [5-46](#)
- multichannel data formatting, [5-46](#)

duty cycles and dead time in PWM, [8-8](#)

DXS_A (data buffer channel B status) bit, [A-40](#)

DXS_B, DSX_A (data buffer channel A/B status) bit, [5-66](#), [A-39](#), [A-40](#)

E

early vs. late frame syncs, [5-40](#)

edge-related interrupts

- four conditions, [4-70](#)

EEMUINENS bit, [A-182](#)

EEMUINFULLS bit, [A-182](#)

EEMUOUIRQENS bit, [A-181](#)

EEMUOUTRDY bit, [A-181](#)

EEMUSTAT register, [A-179](#)

EIPPx (DMA external index) registers, [2-27](#)

ELSI (enable RX status interrupt) bit, [11-9](#), [A-123](#)

EMPPx (DMA external modify) registers, [2-28](#)

EMUI (emulator lower priority interrupt) bit, [B-14](#), [B-18](#), [B-19](#), [B-21](#), [B-23](#), [B-26](#)

emulator

- interrupt (EMUI) bit, [B-14](#), [B-18](#), [B-19](#), [B-21](#), [B-23](#), [B-26](#)

- enable
 - breakpoint (ENBx) bit, [A-178](#)
 - clock outputs, [13-3](#)
 - DMA, [7-20](#)
 - DMA interrupt (INTEN) bit, [6-34](#)
 - external port (asynchronous memory interface), [A-18](#)
 - input data port, [7-15](#)
 - multichannel mode in SPORTs, [A-42](#)
 - PCGs, [13-7](#)
 - PDAP, [7-12](#)
 - pulse width modulation groups, [8-4](#)
 - sample rate converters, [10-22](#)
 - SPDIF transmit buffer, [9-25](#)
 - SPI DMA, [6-16](#)
 - SPI slave, [6-20](#)
 - SPORT DMA (SDEN bit), [5-24](#)
 - SPORT master mode (MSTR), [5-18](#)
- enable receive buffer full interrupt (ERBFI) bit, [11-7](#), [A-123](#)
- enable transmit buffer empty interrupt (ETBEI) bit, [11-7](#), [A-123](#)
- endian format, [5-15](#), [5-45](#), [5-62](#), [6-1](#), [A-37](#)
- enhanced emulation
 - feature enable (EEMUENS) bit, [A-182](#)
 - FIFO status (EEMUOUTFULLS) bit, [A-182](#)
 - INDATA FIFO status (EEMUINFULLS) bit, [A-182](#)
 - OUTDATA FIFO status (EEMUOUTFULLS) bit, [A-182](#)
 - OUTDATA interrupt enable (EEMUOUIRQENS) bit, [A-181](#)
 - OUTDATA ready (EEMUOUTRDY) bit, [A-181](#)
- equation
 - clock, [14-30](#)
 - dead time, [8-8](#)
 - duty cycles in PWM, [8-10](#)
 - frame sync frequency, [5-71](#)
- equation *(continued)*
 - PWM dead time, [8-8](#)
 - PWM switching frequency, [8-5](#)
 - SDRAM refresh rate, [3-50](#)
 - serial clock frequency, [5-70](#)
 - serial port clock divisor, [5-70](#)
 - SPI clock baud rate, [A-60](#)
 - TWI clock divider, [12-5](#)
- error signals and flags, [6-35](#)
- errors
 - clearing in SPI, [6-23](#), [6-25](#)
 - conditions in DAI/DPI, [4-71](#)
 - data buffer status (SPORT), [5-65](#)
 - data in SPORTs, [5-42](#)
 - data truncation, [1-4](#)
 - frame sync (SPORT), [5-26](#), [5-42](#)
 - IDP FIFO, [7-15](#)
 - internal bus (SDRAM), [3-71](#)
 - PCG quantization, [13-2](#)
 - preventing in DMA chaining, [2-16](#)
 - reception (SPI), [6-37](#)
 - S/PDIF error handling, [9-1](#), [9-22](#) to [9-23](#)
 - S/PDIF receiver, [9-17](#)
 - SDRAM, [3-77](#)
 - SPI DMA, [6-14](#), [6-19](#)
 - SPI master, [6-6](#)
 - SPI mode fault, [6-36](#)
 - SPI transmission, [6-21](#), [6-37](#)
 - SPORT, [5-15](#)
 - SRC, [10-3](#)
 - SRC phase difference, [10-12](#)
 - SRC resampling, [10-4](#)
 - TWI master mode, [12-17](#), [12-18](#)
 - TWI repeat start, [12-19](#)
 - TWI slave transfer, [12-14](#), [12-16](#)
 - UART, [11-4](#), [11-9](#)
 - UART baud rate, [11-12](#)
 - UART DMA, [2-46](#), [11-7](#)
 - UART line, [11-7](#)
 - UART sampling, [11-6](#)

Index

errors/flags, DMA, external port, host port,
serial port, SPI port, UART port, [6-35](#)

examples

bidirectional DAI pin buffer, [4-13](#)

DAI pin buffer, [4-10](#)

DAI pin buffer output, [4-12](#)

hold time in AMI, [3-24](#)

idle cycle in AMI, [3-23](#)

interrupt latency, regeneration, [2-11](#)

PCG setup for I2S or left-justified DAI,
[13-15](#)

pin buffer, [4-10](#)

PLL, [14-16](#)

programming SPORTs (DMA
chaining), [5-83](#)

programming SPORTs direct core
access, [5-86](#)

programming SPORTs transmit
(DMA), [5-89](#)

programming the IDP, [7-31](#) to [7-33](#)

programming the PCG channels, [13-23](#)
to [13-25](#)

rotating priority arbitration, [3-86](#)

SPI DMA chaining, [2-43](#)

SRU1 connections for SPORTx, [4-14](#),
[4-16](#)

examples, timing

IDP hold timing mode 00, [7-13](#)

IDP hold timing mode 01, [7-14](#)

IDP I²S, [7-7](#)

IDP left-justified sample pair, [7-7](#)

PDAP, [7-14](#)

SPI clock, [6-5](#)

SPI transfer protocol, [6-28](#), [6-29](#)

SPORT framed vs. unframed data, [5-40](#)
SPORT left-justified sample pair mode,
[5-19](#)

SPORT normal vs. alternate framing,
[5-40](#)

SPORT word select, [5-25](#)

execution stalls, bus transition, [3-84](#)

external data path width, setting, [3-43](#)

external master clock, [14-21](#)

external memory

access timing, [3-36](#)

address bank decoding, [3-53](#)

banked, [3-30](#)

banks, [3-30](#)

external physical address, [3-29](#)

interface, [3-20](#)

most significant word first (MSWF) bit,
[A-18](#)

packing and unpacking (PKDIS) bits,
[A-18](#)

pin descriptions, [3-19](#)

signals, [3-16](#), [3-20](#)

external memory DMA

chained, setting up, [2-36](#)

DMA count (ECEPx) registers, [2-28](#)

DMA index (EIPPx) registers, [2-27](#)

DMA modifier (EMEPx) registers, [2-28](#)

external memory restrictions, [3-52](#)

external port

buffer register pipeline option, [3-45](#)

bus hold cycle bit, [A-18](#)

bus idle cycle bit, [A-19](#)

chain pointer loading sequence, [2-38](#),
[2-40](#)

chain pointer register (CPEP), [2-36](#)

channel freezing, [3-18](#)

conflict resolution, [3-25](#)

core address mapping, [3-52](#)

data pin mode select (EPDATA) bits,
[A-8](#)

delay line DMA, [2-40](#)

DMA, [2-35](#) to [2-40](#)

DMA registers, [A-14](#) to [A-16](#)

DMA throughput, [3-18](#)

hold cycles, [3-23](#)

modes, [3-24](#)

external port *(continued)*

 modes, setting, [3-24](#)
 read hold cycle (RHC) bits, [A-19](#)

external port bits

 bank select (BxSD), [A-11](#)
 bus priority (EPBR), [A-12](#)
 data enable (DATA), [A-13](#)
 delay line write pointer write back status (WBS), [A-16](#)
 DMA chain status (CHS), [A-16](#)
 DMA chaining enable (CHEN), [A-15](#)
 DMA channel priority, [A-12](#)
 DMA circular buffer enable (CBEN), [A-15](#)
 DMA delay-line enable (DLEN), [A-15](#)
 DMA direction (DMADR), [A-15](#)
 DMA enable (DMAEN), [A-15](#)
 DMA external interface status (EXTS), [A-16](#)
 DMA FIFO status (DFS), [A-15](#)
 DMA flush FIFO (DFLSH), [A-15](#)
 DMA transfer direction status (DIRS), [A-16](#)
 DMA transfer status (DMAS), [A-15](#)
 flush tap list FIFO (TFLSH), [A-15](#)
 freeze length (FRZDMA), [A-12](#)
 freeze length core (FRZCR), [A-12](#)
 tap list FIFO status (TFS), [A-15](#)
 tap list loading status (TLS), [A-16](#)
external port control bus, [3-25](#), [3-26](#)
external port DMA
 chain pointer loading sequence, [2-37](#)
 chained, [2-36](#)
 channel priority, [2-35](#)
 delay line, [2-39](#)
 FIFO, [2-35](#)
 set up, [2-36](#)
external port registers, [A-10](#) to [A-26](#)
 AMI control (AMICTLx), [A-17](#)
 DMA control (DMACx), [2-26](#)

external port registers *(continued)*

 length and base (ELEP, EBEP), [2-37](#)

F

FE (framing error) bit, [11-4](#)

FIFO

 control and status in input data port, [7-15](#)
 overflow clear bit, [7-15](#)
 to memory data transfer, [7-16](#)

FIR filter in SRC, [10-2](#), [10-7](#)

fixed-point overflow interrupt (FIXI) bit, [B-17](#), [B-21](#), [B-25](#)

flag select bits (SPI), [6-10](#)

flags

 digital audio interface, [4-64](#)
 errors DMA, external port, host port, serial port, SPI port, and UART port, [6-35](#)
 flag interrupt mode (IRQxEN) bits, [A-7](#)
 input/output (FLAGx) pins, [6-4](#), [14-8](#)

FLAGx pins, [6-4](#), [14-8](#)

floating-point

 compatibility, [1-3](#)
 FLTII (invalid operation interrupt) bit, [B-17](#), [B-21](#), [B-26](#)
 FLTOI (overflow interrupt) bit, [B-17](#), [B-21](#), [B-25](#)
 FLTUI (underflow interrupt) bit, [B-17](#), [B-21](#), [B-26](#)
 overflow interrupt (FLTOI) bit, [B-17](#), [B-21](#), [B-25](#)
 underflow interrupt (FLTUI) bit, [B-17](#), [B-21](#), [B-26](#)

FLTII (floating-point invalid operation interrupt) bit, [B-21](#), [B-26](#)

formula, for frame sync pulse, [5-70](#)

frame sync

 A source (FSASOURCE) bit, [A-157](#)
 active low vs. active high, [5-39](#)

Index

frame sync *(continued)*

- both enable (FS_BOTH) bit, [5-64](#)
 - early vs. late, [5-40](#)
 - equations, [13-11](#)
 - frequencies, [5-69](#)
 - in multichannel mode, [5-28](#)
 - internal vs. external, [5-38](#)
 - options (FS_BOTH and DIFS), [5-18](#), [5-24](#)
 - options (FS_BOTH), [5-18](#)
 - output, synchronizing, [13-7](#)
 - PCG B source (FSBSOURCE) bit, [13-13](#)
 - signals, configuring, [5-6](#)
 - SPORT frame on rising frame sync (FRFS) bit, [5-17](#)
 - SPORT frame sync required (FSR) bit, [5-63](#)
- frame sync rates
- setting in SPORTs, [5-17](#), [5-21](#)
 - setting the internal serial clock in SPORTs, [5-21](#)
- framed versus unframed data in SPORTs, [5-37](#)
- framing bits, SPORT, [5-17](#)
- freezing, channel (in external port DMA), [3-18](#)
- frequency mode, single-channel, double-frequency, [9-8](#)
- frequency of the frame sync output, [13-9](#)
- full-duplex operation, specifications, [5-6](#)

G

- general-purpose (GPIO) and flags for digital audio interface, [4-64](#)
- general-purpose IOP timer 2 interrupt mask (GPTMR2IMSK) bit, [B-10](#)
- general-purpose IOP timer interrupt mask (GPTMRxIMSK) bits, [B-10](#)
- generators, optional reset, [14-27](#)

- glitch vulnerability (SPORTs), [5-10](#)
- GM (get more data) bit, [6-11](#), [6-20](#), [6-37](#), [A-54](#)
- ground plane, in PCB design, [14-34](#)
- groupings of signals in DAI/DPI, [4-8](#)

H

- handshaking, external port, [3-20](#), [3-79](#)
- hardware interrupt
 - bits, [B-15](#), [B-16](#), [B-19](#), [B-20](#), [B-24](#)
 - signals $\overline{\text{IRQ2-0}}$, [14-4](#), [14-6](#), [14-8](#), [B-16](#), [B-24](#)
- high and low priority latches, [2-7](#), [4-69](#)
- hold cycle (external bus) bit, [A-18](#)
- hold cycles, external port, [3-23](#)
- hold off, processor bus transition, [3-84](#)
- hold time
 - inputs, [14-32](#)
 - recognition of asynchronous input, [14-32](#)
- hold time cycles, setting, [3-23](#)
- hysteresis on $\overline{\text{RESET}}$ pin, [14-33](#)

I

- I/O address breakpoint hit (STATI0) bit, [A-181](#)
- I/O interface to peripheral devices, [5-1](#)
- I/O processor
 - See also* DMA; specific peripherals
 - address bus (IOA), [2-29](#)
 - addressing in, [2-29](#)
 - bandwidth, [2-12](#)
 - baud rate, [2-43](#)
 - bus arbitration and contention, [2-20](#)
 - bus diagram, [2-25](#)
 - bus grant, [2-21](#)
 - chain insertion mode (DMA), [2-41](#)
 - chain pointer (CPSPI) register, [2-16](#)
 - chain pointer registers, [2-27](#), [2-43](#)

I/O processor *(continued)*

- chained DMA, [2-14](#)
- configuring DMA, [2-2](#)
- count registers, [2-27](#), [2-31](#)
- DAI interrupt registers (DAI_IRPTL_H, DAI_IRPTL_L), [2-7](#)
- data (IOD) bus, [2-20](#)
- data buffers in DMA, [2-32](#)
- DMA channel priority, [2-19](#)
- DMA channel registers, [2-32](#)
- DMA enable (DEN) bit, [2-48](#)
- DMA interrupt registers, [2-7](#)
- DMA interrupt vector locations, [2-8](#)
- DMA parameter registers, [2-17](#)
- DMA sequence complete interrupt, [2-8](#)
- external count (ECEPx) registers, [2-28](#)
- external index (EIPPx) registers, [2-27](#)
- external modify (EMEPx) registers, [2-28](#)
- interrupt driven I/O, [2-6](#)
- interrupt service routine restriction, [2-5](#)
- latency, [2-11](#) to [2-13](#)
- memory access, DMA, [2-8](#)
- polling driven I/O, [2-12](#)
- program control interrupt (PCI) bit, [2-8](#)
- regenerated interrupts, avoiding, [2-11](#)
- registers, listed, [A-2](#)
- restrictions, [2-2](#), [2-31](#)
- restrictions (ISR), [2-5](#)
- stall conditions, [2-3](#)
- stall cycles in, [2-5](#)
- status driven I/O, [2-12](#)
- status polling, [2-12](#)
- transfer types, [2-1](#)
- type 1 or LW instructions, [2-2](#)

I²C port. *See* TWI controller

I²S

- (Tx/Rx on left channel first), [5-11](#), [5-12](#), [A-30](#)
- (Tx/Rx on right channel first), [5-11](#), [5-12](#), [A-30](#)

I²S *(continued)*

- control bits, [5-21](#)
- example for DAI, [13-15](#)
- mode, [5-74](#)
- mode (IDP), [7-3](#), [7-5](#), [7-21](#)
- SPCTLx control bits, [5-22](#)
- timing (IDP), [7-7](#)
- transmit and receive channel order (FRFS), [5-18](#), [5-23](#)

ICLK (internal clock select) bit, [A-37](#)

identification (ID2-0) pin, [3-81](#)

identification code (IDC) bit, [3-93](#), [A-10](#)

idle cycle (external bus) bit, [A-19](#)

idle cycle in external port, [3-22](#)

IDP

- (DAI) interrupt service routine
 - steps, [7-28](#)
- clocking select, [7-12](#)
- FIFO control, [7-15](#)
- FIFO memory data transfer, [7-16](#)
- FIFO status, [7-15](#)
- hold input, [7-12](#)
- illustrated, [7-1](#)
- interrupt-driven transfers, [7-17](#), [7-18](#)
- interrupts, [7-17](#), [7-19](#), [7-22](#), [7-24](#)
- masking, [7-9](#), [A-74](#)
- memory data transfer, [7-16](#)
- packing modes, [7-9](#), [7-11](#)
- packing unit, [7-9](#)
- parallel input mode, [7-8](#)
- PDAP control (IDP_PP_CTL) register, [7-9](#)
- ping-pong DMA, [7-22](#) to [7-24](#)
- programming examples, [7-31](#)
- serial inputs, [7-3](#)
- serial modes, setting, [7-5](#)

IDP bits

- bus hang disable (IDP_BHD), [7-15](#), [7-19](#), [A-67](#)

Index

IDP bits *(continued)*

- clear buffer overflow (IDP_CLROVR),
[7-15](#), [7-16](#), [7-25](#), [A-67](#)
- DMA enable (IDP_DMA_EN), [7-20](#),
[7-22](#), [7-25](#), [A-67](#)
- DMA status (IDP_DMAx_STAT),
[7-26](#), [A-111](#)
- enable (IDP_ENABLE), [7-15](#), [7-18](#),
[7-20](#), [7-22](#), [7-23](#), [A-67](#)
- FIFO number of samples
(IDP_FIFOSZ), [7-15](#), [7-16](#), [7-19](#),
[A-111](#)
- FIFO overflow (IDP_FIFO_OVER),
[7-15](#), [7-16](#)
- FIFO overflow (SRU_OVF), [7-25](#)
- FIFO samples exceed interrupt
(IDP_FIFO_GTN_INT), [7-17](#),
[7-19](#), [A-113](#)
- frame sync format (IDP_SMODEx),
[7-4](#), [7-18](#), [7-21](#), [A-67](#), [A-70](#)
- IDP_DMA_EN (DMA enable
restriction), [7-19](#)
- IDP_DMA_EN (input data port DMA
enable), [7-20](#)
- monitor number of samples
(IDP_NSET), [7-17](#), [7-18](#), [7-19](#), [A-67](#)
- PDAP clock edge
(IDP_PDAP_CLKEDGE), [7-12](#),
[7-18](#), [7-21](#), [A-77](#)
- PDAP enable (IDP_PDAP_EN), [7-12](#),
[7-22](#), [A-78](#)
- PDAP input mask bits, [7-18](#)
- PDAP mask (IDP_Pxx_PPMASK), [7-9](#),
[7-21](#), [A-74](#)
- PDAP packing mode
(IDP_PDAP_PACKING), [7-9](#), [A-77](#)
- PDAP reset (IDP_PDAP_RESET), [7-8](#),
[A-78](#)
- ping-pong DMA enable (IDP_PING)
bits, [A-68](#)

IDP bits *(continued)*

- port select (IDP_PORT_SELECT), [7-8](#),
[7-18](#), [7-21](#), [A-77](#)
- reset (IDP_PDAP_RESET) bit, [A-78](#)
- IDP registers
 - control (IDP_CTL0), [7-18](#), [7-19](#), [7-20](#),
[A-66](#)
 - control (IDP_CTL1), [7-19](#), [A-68](#)
 - DMA control, [2-26](#), [A-70](#)
 - DMA count (IDP_DMA_Cx), [7-21](#),
[7-28](#), [A-71](#)
 - DMA index (IDP_DMA_Ix), [7-21](#),
[7-28](#), [A-70](#)
 - DMA modify (IDP_DMA_Mx), [7-21](#),
[7-28](#), [A-71](#)
 - FIFO (IDP_FIFO), [7-15](#), [7-16](#), [7-17](#),
[A-69](#)
 - IDP_CTLx (input data port control),
[7-18](#), [7-19](#), [7-20](#), [A-66](#), [A-68](#)
 - IDP_DMA_AIx (ping-pong DMA
index), [7-23](#)
 - IDP_DMA_PCx (ping-pong DMA
count), [7-23](#)
 - PDAP control (IDP_PP_CTL), [7-8](#), [7-9](#),
[7-12](#), [A-74](#)
 - ping-pong DMA count
(IDP_DMA_PCx) registers, [7-23](#)
 - ping-pong DMA index
(IDP_DMA_AIx), [7-23](#)
- IFS (SPORT internal frame sync select) bit,
[5-63](#), [A-38](#)
- IICD (illegal input condition interrupt) bit,
[B-15](#), [B-19](#), [B-23](#)
- IISPI (SPI DMA start address) register,
[A-64](#)
- IISPx (serial port DMA internal index)
registers, [2-27](#), [2-29](#), [A-50](#)
- illegal input condition detected (IICD) bit,
[B-15](#), [B-19](#), [B-23](#)
- IMASK (interrupt mask) register, [B-18](#)

- IMASKP (interrupt mask pointer) register, [B-22](#)
- IMSPI (serial peripheral interface address modify) register, [6-16](#), [6-19](#), [A-64](#)
- IMSPx (SPORT DMA address modifier) registers, [2-27](#), [2-29](#), [A-50](#)
- INCLUDE directory, [5-49](#)
- INDATA interrupt enable (EEMUINENS) bit, [A-182](#)
- INDIV (input divisor) bit, [A-172](#)
- input data port. *See* IDP
- input setup and hold time, [14-32](#)
- input signal conditioning, [14-32](#)
- input slave select enable (ISSEN) bit, [6-36](#), [A-54](#)
- input synchronization delay, [14-8](#)
- instruction address breakpoint hit (STATIx) bit, [A-181](#)
- instruction rate, [14-14](#)
- instructions
 - atomic, using for clock and frame sync, [13-10](#)
 - conditional in SDRAM, [3-82](#), [3-93](#)
 - dual data move restriction, [3-78](#)
 - external memory fetch, [3-25](#)
 - packed, [3-20](#)
 - return from interrupt, [4-67](#)
- INTEN (DMA interrupt enable) bit, [6-34](#)
- interconnections, master-slave, [6-4](#)
- internal clock select (ICLK) bit, [5-62](#)
- internal frame sync (SPORT IFS) bit, [5-63](#)
- internal I/O bus, [2-19](#)
- internal I/O bus arbitration (request and grant), [2-19](#)
- internal interrupt vector table (IIVT) bit, [A-6](#)
- internal memory
 - DMA count (CSPx) registers, [A-51](#)
 - DMA index (IDP_DMA_Ix) registers, [7-28](#)
- internal memory *(continued)*
 - DMA index (IISPx) registers, [2-27](#), [2-29](#), [A-50](#)
 - DMA modifier (IDP_DMA_Mx) registers, [7-28](#)
 - DMA modifier (IMSPx) registers, [2-27](#), [2-29](#)
 - memory-to-memory data transfers, [2-48](#)
 - transfers, [2-48](#)
- internal memory banks, [3-33](#)
- internal serial clock (ICLK) bit, [5-62](#)
- setting, [5-17](#)
- internal transmit frame sync (IFS) bit, [5-63](#)
- internal vs. external frame syncs, [5-38](#)
- INTERR (enable interrupt on error) bit, [6-34](#)
- interrupt
 - hardware, [B-15](#), [B-20](#)
 - input x interrupt (IRQxI) bit, [B-15](#), [B-16](#), [B-20](#), [B-24](#)
 - latch (IRPTL) register, [B-13](#)
 - latch/mask (LIRPTL) register, [B-6](#)
 - mask (IMASK) register, [B-18](#)
 - mask pointer (IMASKP) register, [B-22](#)
 - peripheral interrupt priority registers (PICR), [A-164](#)
 - PWM interrupt (P13I) bit, [B-9](#)
 - transfers, starting, [7-18](#), [7-20](#), [7-23](#)
- interrupt and timer pins, [14-8](#)
- interrupt controller, digital audio interface, [4-65](#), [A-112](#)
- interrupt driven DMA, I/O processor, [2-8](#)
- interrupt input ($\overline{\text{IRQ2-0}}$) pins, [14-8](#)
- interrupt input x interrupt (IRQxI) bit, [B-19](#), [B-24](#)
- interrupt latch (IRPTL) register, [6-34](#), [B-13](#)
- interrupt latch/mask (LIRPTL) registers, [6-34](#), [B-6](#)

Index

interrupt mask (IMASK) control register, [B-18](#)
interrupt vector, sharing, [5-72](#)
interrupts, [B-15](#), [B-19](#), [B-24](#)
 (enable RX status interrupt) bit, [A-123](#)
 assigning priority for UART, [11-11](#)
 catastrophic, [4-65](#)
 conditions for generating interrupts in
 SPORTs, [5-75](#)
 conditions, UART, [11-9](#)
 digital audio interface, [4-66](#), [7-22](#)
 digital peripheral interface, [4-67](#)
 I/O processor, using in the, [2-6](#)
 latch status for, [B-13](#)
 listed in registers, [B-1](#)
 non-maskable software (RSTI), [A-6](#)
 normal, [4-65](#)
 UARTLSIE (enable UART RX status
 interrupt) bit, [11-9](#)
INVSFSx (active low frame sync select for
 frame sync) bits, [13-12](#)
IOP register set, [5-49](#)
IRPTL (interrupt latch) register, [B-13](#)
IRQ2-0 (hardware interrupt) pins, [14-8](#)
IRQxI (hardware interrupt) bit, [B-15](#),
 [B-16](#), [B-19](#), [B-20](#), [B-24](#)
ISSS (input service select) bit, [A-58](#)

J

jitter, clock, [14-33](#)
JTAG interface pins, [14-12](#)

L

LAFS (late transmit frame sync select) bit,
 [A-38](#)
LAFS (SPORT late transmit frame sync
 select) bit, [5-11](#), [5-14](#), [5-16](#), [5-20](#),
 [5-21](#), [5-40](#), [5-63](#), [A-30](#), [A-38](#)

latching
 high and low priority (DAI/DPI), [4-69](#)
 interrupt latch (IRPTL) register, [B-13](#)
 status for interrupts, [B-13](#)
latchup, [14-32](#)
latency
 CAS, setting, [3-40](#)
 definition, CAS, [3-32](#)
 I/O processor registers, [A-2](#)
 in SPORT registers, [5-58](#)
 input synchronization, [14-8](#)
 instruction fetch, external memory, [3-25](#)
 setting CAS, [3-40](#)
left-justified data (S/PDIF), [9-7](#), [9-10](#)
left-justified data (SRC), [10-13](#)
 framing, [10-17](#)
 matched-phase mode, [10-17](#)
 setting, [10-14](#)
left-justified sample pair mode (IDP), [7-21](#)
 data transfer, core, [7-18](#)
 data transfer, DMA, [7-21](#)
 DMA, [7-26](#)
 FIFO data packing, [7-6](#)
 setting, [7-5](#)
 timing, [7-7](#)
left-justified sample pair mode (SPORTs),
 [5-10](#), [5-16](#), [5-17](#)
 control bits, [5-17](#)
 Tx/Rx on FS falling edge, [5-12](#), [A-30](#)
 Tx/Rx on FS rising edge, [5-12](#), [A-30](#)
left-justified waveform (PWM), [8-1](#)
LIRPTL (interrupt) registers, [6-34](#), [B-6](#)
little endian (TWI controller), [12-9](#)
loader kernel, [14-37](#)
low active transmit frame sync (LFS, LTFs
 and LTDV) bits, [5-63](#), [A-159](#)
LRFS (SPORT logic level) bit, [5-29](#)
LSBF (least significant bit first) bit, [5-62](#),
 [A-37](#)

M

making connections via the signal routing unit, [4-15](#)

manual

contents, [xxxii](#)

conventions, [xliii](#)

new in this edition, [xxxiv](#)

related documents, [xxxviii](#)

revisions, [xxxiv](#)

maskable interrupts, [A-6](#)

masking data (SDRAM), [3-33](#)

master clock, external, [14-21](#)

master input slave output (MISOx) pins, [6-2](#), [6-7](#), [6-8](#), [6-27](#)

master mode enable (SPORT), [5-13](#), [5-29](#)

master mode operation, SPI, [6-10](#)

master out slave in (MOSIx) pin, [6-2](#), [6-7](#), [6-27](#)

master-slave interconnections, [6-4](#)

memory

boot memory, [3-30](#), [14-39](#)

data transfer, FIFO, [7-16](#)

internal banks, [3-33](#)

memory banks, internal, [3-33](#)

memory read \overline{RD} pin, [3-21](#), [3-83](#)

memory select (flags) programming (MSEN) bit, [A-8](#)

memory select (\overline{MSx}) pins, [3-21](#), [3-53](#), [3-83](#), [3-84](#), [3-89](#)

memory transfer types, [2-1](#)

memory-mapped emulation, breakpoint registers, [2-4](#)

memory-mapped IOP RXSPI buffer registers, [A-59](#)

memory-to-memory DMA, [2-48](#)

memory-to-memory DMA register, [A-28](#)

MISCAx_I (signal routing unit external miscellaneous) register, [13-13](#)

miscellaneous signals, [4-65](#)

MISOx pins, [6-27](#)

mode

broadcast (SPI), [6-8](#)

chain insertion, [2-14](#), [2-41](#)

chained DMA, [2-41](#)

left-justified (SPORT), [5-16](#)

left-justified sample pair (IDP), [7-3](#)

loopback (SPORT), [5-6](#), [A-42](#)

master (SPI), [6-38](#)

multichannel (SPORT), [5-3](#)

open drain (SPI), [6-9](#)

packing (IDP), [7-9](#), [7-10](#)

right-justified (IDP), [7-3](#)

self-refresh, [3-34](#)

serial mode settings (IDP), [7-4](#)

single channel double frequency (S/PDIF), [9-8](#)

standard serial (SPORT), [5-12](#)

standard serial, signals (SPORT), [5-5](#)

TDM (SPORT), [5-26](#)

time division multiplexed (TDM), [9-19](#)

two channel (SPDIF), [9-8](#)

UART DMA, [2-44](#)

UART non-DMA, [11-13](#)

mode (SPDIF), two channel, [9-8](#)

mode fault (multimaster error) SPI DMA status (MME) bit, [6-35](#), [6-36](#)

mode fault error (MME) bit, [6-9](#), [6-35](#), [6-36](#)

mode register (SDRAM controller), [3-33](#)

MOSIx pins, [6-27](#)

MSBF (most significant byte first) bit, [A-54](#)

\overline{MSx} pins, memory select, [3-52](#), [3-53](#)

MTM_FLUSH (memory-to-memory FIFO flush) bit, [2-48](#)

MTMDMACTL (memory-to-memory DMA control) register, [2-48](#)

MTxCCSx (serial port transmit compand) registers, [A-47](#)

MTxCCSy and MRxCCSy (multichannel compand select) registers, [5-47](#)

Index

MTxCSx (serial port transmit select)
 registers, [A-46](#)
multichannel A and B channels, [A-30](#)
multichannel compand select (MTxCCSy
 and MRxCCSy) registers, [5-47](#)
multichannel operation (SPORT), [5-25](#)
multichannel selection registers (SPORT),
 [5-31](#)
multichannel, A and B channels, [5-12](#)
multi-device SPI configuration, [6-12](#)
multimaster conditions, [6-12](#)
multimaster environment, [6-8](#)
multiplexed (TDM) mode, time division,
 [9-19](#)
multiplexing
 in external port data pins, [A-8](#)
 pins, [14-3](#) to [14-12](#)
 PWM pins, [A-8](#)
multiprocessing. *See* shared memory

N

negate breakpoint (NEGx) bit, [A-175](#),
 [A-177](#)
NINT (pending interrupt) bit, [11-10](#)
non-chained DMA, [2-13](#)
normal frame sync (SPORT), [5-40](#)
normal interrupts, [4-65](#)

O

OE (overflow error) bit, [11-4](#)
one shot frame sync A or B (STROBE_x)
 bits, [13-12](#)
one shot option (STROBE_B) bit, [13-13](#)
one shot, defined, [13-13](#)
OPD (open drain output) pin, [6-9](#)
OPMODE (SPORT serial port operation
 mode) bit, [5-13](#), [5-17](#)
OR, logical, [7-30](#), [9-24](#), [A-175](#), [A-177](#),
 [A-178](#)

OSPID (operating system process ID), [2-4](#),
 [A-182](#)
OSPIDENS (operating system process ID)
 register enable bit, [A-182](#)
output control unit, PWM, [8-17](#)
output pulse width, defined, [13-13](#)
output strobe, PDAP, [7-14](#)
over-modulation, PWM, [8-12](#)

P

PACK (SPORT packing enable) bit, [A-37](#)
packing modes in PDAP, illustrated, [7-10](#)
packing, data, [6-32](#)
page size (SDRAM), [A-25](#)
page sizes in SDRAM, [3-34](#)
parallel data acquisition port. *See* PDAP,
 IDP
parallel input mode, [7-8](#)
parameter registers, I/O processor, [2-25](#)
PCG
 active low frame sync select for frame
 sync (INVFS_x) bits, [13-12](#)
 bypass mode, [13-12](#)
 clock A source (CLKASOURCE) bit,
 [A-157](#)
 clock input (CLKIN) pin, [4-72](#), [13-2](#),
 [13-20](#)
 clock input source enable
 (CLK_x_SOURCE_IOP) bit, [A-161](#)
 clock with external frame sync enable
 (FS_x_SYNC) bit, [A-161](#)
 control (PCG_CTL_A_x) registers,
 [13-13](#), [A-156](#)
 division ratios, [13-16](#)
 enable clock (ENCLK_x) bit, [A-156](#)
 enable frame sync (ENFS_x) bit, [A-156](#)
 frame sync A source (FSASOURCE) bit,
 [13-13](#), [A-157](#)
 frame sync B source (FSBSOURCE) bit,
 [13-13](#), [A-157](#)

- PCG *(continued)*
- frame sync input source enable
(CLKx_SOURCE_IOP) bit, [A-161](#)
 - frame sync with external frame sync
enable (FSx_SYNC) bit, [A-161](#),
[A-162](#)
 - frame syncs, [13-11](#)
 - frequency of the frame sync output, [13-9](#)
 - one shot frame sync A or B (STROBEx)
bits, [13-12](#)
 - one shot option, [13-13](#)
 - PCG_CTLA0 (control) register, [A-156](#)
 - phase shift of frame sync, [13-9](#)
 - pulse width (PCG_PW) register, [13-11](#),
[13-13](#)
 - pulse width for frame sync (PWFSx) bit,
[A-158](#)
 - setup for I²S or left-justified DAI
example, [13-15](#)
 - synchronization with the external clock,
[13-7](#)
- PCI (program control interrupt) bit, [2-8](#),
[2-16](#)
- PDAP, [7-8](#) to [7-14](#)
- (rising or falling) clock edge
(IDP_PDAP_CLKEDGE) bit, [A-77](#)
 - control (IDP_PP_CTL) register, [7-8](#),
[A-74](#)
 - data signal, [7-14](#)
 - enable (IDP_PDAP_EN) bit, [A-78](#)
 - hold signal, [7-14](#)
 - port mask bits (IDP_Pxx_PDAPMASK),
[A-75](#)
 - strobe, output, [7-14](#)
 - timing, [7-14](#)
- PE (parity error) bit, [11-4](#)
- peripheral devices, I/O interface to, [5-1](#)
- peripheral DMA counter registers, [2-27](#)
- peripheral interrupt priority control
(PICR) registers, [A-164](#)
- peripherals
- memory mapped, [3-20](#)
 - overview, [1-7](#)
- peripherals, processor specific, [1-5](#)
- phase shift of frame sync, [13-9](#)
- PICR (peripheral interrupt priority)
registers, [A-164](#)
- ping-pong DMA, [7-22](#) to [7-24](#)
- pins
- See also* signals
 - ACK, enabling, [A-18](#)
 - data, function of, [14-7](#)
 - descriptions, [14-2](#)
 - external memory, [3-19](#)
 - memory select (\overline{MSx}), [3-53](#)
 - multiplexing, [14-2](#) to [14-12](#)
 - open drain output, [6-9](#)
 - pin states during SDRAM commands,
[3-72](#)
 - \overline{RESET} , [14-33](#)
 - test clock (TCK), [14-12](#)
 - test data input (TDI), [14-12](#)
 - test data output (TDO), [14-12](#)
 - test mode select (TMS), [14-12](#)
 - test reset (\overline{TRST}), [14-12](#)
- plane, ground, [14-34](#)
- PLL programming restrictions, [14-16](#)
- PLL startup, [14-19](#), [14-21](#)
- PLL-based clocking, [14-13](#) to [14-19](#)
- PLLDx (PLL divider) bits, [A-172](#)
- PLLM (PLL multiplier) bit, [A-172](#)
- PMCTL (power management control)
register, [A-170](#), [A-172](#)
- polarity
- clock polarity (CLKPL) bit, [A-55](#)
 - IDP encoding, [7-6](#)
 - PWM configuration, [8-15](#)
 - PWM double-update mode, [8-11](#)
 - PWM polarity select registers
(PWMPOLx), [A-83](#)

Index

polarity *(continued)*

- PWM single update mode, [8-9](#)
- SPDIF connections, [9-5](#)
- SPI clock, [6-21](#), [6-27](#)
- polling the I/O processor, [2-12](#)
- porting from previous SHARCs
 - paged DRAM boundary, [3-22](#)
 - symbol changes, [1-12](#)
- post-modify, [14-53](#)
- power management control register (PMCTL), [A-170](#), [A-172](#)
- power supply, monitor and reset generator, [14-28](#)
- power-up reset circuit, [14-27](#), [14-28](#)
- power-up sequence start, SDRAM controller, [3-43](#)
- power-up, SDRAM (SDPM) bit, [A-23](#)
- preambles, S/PDIF, [9-7](#)
- precharge command, [3-34](#)
- precision clock generators. *See* PCG
- predictive reads, disable bit (NO_OPT), [A-19](#)
- printed circuit board design, [14-34](#)
- priority of the SPORT interrupts, [5-72](#)
- priority, rotating priority arbitration
 - example, [3-86](#)
- processor
 - architectural overview, [1-6](#)
 - clock frequency, [5-1](#)
 - core overview, [1-7](#)
 - peripheral set by model, [1-5](#)
 - product information, [xxxvii](#)
 - stalls, [14-52](#)
- product-related documents, [xxxvii](#)
- program control interrupt (PCI) bit, [2-8](#), [2-16](#)
- program memory breakpoint hit (STATPA) bit, [A-181](#)
- programmable interrupt bits, [B-6](#) to [B-9](#)

programmable interrupt registers (PICRx), [A-164](#) to [A-170](#)

programming examples

- input data port, [7-31](#) to [7-33](#)
- power management, [14-14](#) to [14-16](#)
- precision clock generators, [13-23](#) to [13-25](#)
- SPORTs, [5-83](#) to [5-86](#)
- TWI controller, [12-15](#)

programming guidelines, S/PDIF transmitter, [9-24](#)

pulse code modulation (PCM), [5-20](#)

PWM

- 16-bit read/write duty cycle registers, [8-7](#)
- accuracy, [8-17](#)
- center-aligned modes, [8-3](#)
- center-aligned paired PWM
 - double-update mode, [8-11](#)
- channel duty control (PWMA, PWMB) registers, [A-84](#)
- channel low duty control (PWMAL, PWMBL) registers, [A-84](#)
- control (PWMCTL) register, [A-80](#)
- control register use, [8-4](#)
- crossover mode, [8-16](#)
- dead time equation, [8-8](#)
- double update mode, [8-15](#)
- duty cycles, [8-8](#)
- edge-aligned mode, [8-3](#)
- emergency dead time, [8-13](#)
- equations, [8-5](#) to [8-12](#)
- full on to full off transition, [8-14](#)
- full on, off conditions, [8-12](#)
- global control (PWMGCTL) register, [A-78](#)
- global status (PWMGSTAT) register, [A-79](#)
- interrupts, [8-5](#)
- output control unit, [8-17](#)
- over-modulation, [8-12](#)

PWM *(continued)*

- period (PWMPERIOD) registers, [8-5](#), [A-81](#)
- period completion status bits, [8-5](#)
- polarity select (PWMPOL) registers, [A-83](#)
- short-circuit condition, [8-6](#)
- single update mode, [8-15](#)
- status (PWMSTAT) register, [A-81](#)
- switching dead time, [8-6](#)
- switching frequencies, [8-5](#)
- switching frequency equation, [8-5](#)
- three-phase timing unit, [8-7](#), [8-13](#)
- PWMAL, PWMBL (pulse width modulation channel low duty control) registers, [A-84](#)
- PWMCTL (pulse width modulation control) register, [A-80](#)
- PWMGCTL (pulse width modulation global control) register, [A-78](#)
- PWMGSTAT (pulse width modulation global status) register, [A-79](#)
- PWMPERIOD (pulse width modulation period) registers, [A-81](#)
- PWMPOL (pulse width modulation polarity select) registers, [A-83](#)
- PWMSTAT (pulse width modulation status) register, [A-81](#)

R

- RAS definition, CBR (CAS before RAS), [3-33](#)
- read (\overline{RD}) pin, [3-21](#), [3-83](#)
- receive busy (overflow error) SPI DMA status (SPIOVF) bit, [6-35](#), [A-63](#)
- receive busy (overflow error) SPI status (ROVF) bit, [6-37](#)
- receive data (RXSPI) buffer, [6-2](#)
- receive data buffer shadow (RXSPI_SHADOW) register, [A-59](#)

- receive data buffer status (RXS) bit, [6-30](#)
- receive data, serial port (RXSPx) registers, [2-25](#)
- receive data, SPI (RXSPI) register, [6-37](#)
- receive overflow error (SPIOVF) bit, [6-25](#), [6-26](#), [6-34](#)
- receive shift (RXSR) register, [6-2](#)
- reception error bit (ROVF, in SPI), [6-37](#)
- refresh rate in SDRAM, [3-50](#)
- register writes and effect latency (SPORT), [5-67](#)
- registers. *See* peripheral specific registers related documents, [xxxviii](#)
- reset
 - default settings, [14-3](#), [14-9](#)
 - generators, [14-27](#)
 - input hysteresis, [14-33](#)
 - interrupt (RSTI) bit, [B-14](#), [B-19](#), [B-23](#)
 - partial, [14-22](#)
 - pin, [14-20](#), [14-33](#)
 - PLL startup, [14-19](#), [14-21](#)
 - running, [14-22](#) to [14-26](#)
 - running (RUNRST), [14-22](#)
 - running reset control register ($\overline{RUNRSTCTL}$), [14-25](#)
- \overline{RESET} pin, [14-33](#)
- RESETOUT (reset output) signal, [14-12](#)
- restrictions
 - core hang in SPORTS, [5-7](#), [5-9](#)
 - count (DMA) registers, [2-31](#)
 - DMA, [2-47](#)
 - external port, [3-94](#)
 - I/O processor, [2-2](#), [2-31](#)
 - idle cycle in page boundary, [3-22](#)
 - PLL clock, [14-16](#)
 - \overline{RESET} input, [14-33](#)
 - RS-232, [5-9](#)
 - SDRAM, [3-52](#)
 - SPORTS read/write inactive buffer, [5-69](#)
 - SPORTs SLEN value, [5-44](#)

Index

restrictions *(continued)*
 UART port controller, [11-11](#)
right channel status for subframe B
 (DIRCHANR) register, [A-96](#)
right-justify format (SPORTs)
 companding, [5-61](#)
 setting, [5-46](#), [A-37](#)
rising and falling edge masks
 digital audio interface, [4-70](#)
rotating priority arbitration example, [3-86](#)
rotating priority bus arbitration (RPBA)
 pin, [3-81](#), [3-86](#)
ROVF bit, [6-37](#)
ROVF_A or TUVF_A (channel A error
 status) bit, [A-40](#)
ROVF_A or TUVF_A (serial port error
 status) bits, [A-40](#)
ROVF_B or TUVF_B (channel B error
 status) bit, [A-39](#)
row addresses, [3-34](#)
RS-232 device restrictions, [5-9](#)
RSTI (reset interrupt) bit, [B-14](#), [B-19](#),
 [B-23](#)
running reset, [14-22](#)
running reset control register
 (RUNRSTCTL), [14-25](#)
RUNRSTIN (running reset input) signal,
 [14-12](#)
RX_UACEN (DMA receive buffer enable)
 bit, [11-12](#)
RXFLSH (flush receive buffer) bit, [6-23](#),
 [6-25](#), [6-26](#)
RXS (SPI data buffer status) bit, [6-30](#), [A-57](#)
RXSPI, RXSPIB (SPI receive buffer)
 registers, [6-13](#), [6-37](#), [A-59](#)
RXSPI_SHADOW, RXSPIB_SHADOW
 (SPI receive buffer shadow) registers,
 [A-59](#)
RXSPx (serial port receive buffer) registers,
 [2-25](#), [A-44](#)

RXSR (SPI receive shift) register, [6-2](#)

S

S/PDIF
 audio standards, [9-16](#)
 biphase encoded data input, [9-18](#)
 biphase encoding, [9-11](#)
 BLK_START signal, [9-11](#)
 block structure, [9-2](#)
 channel status bit, [9-9](#)
 clock, [9-10](#)
 clock (SCLK) input, [9-10](#)
 DIR_I (receiver input) signal, [9-18](#)
 DIR_LRCLK_O (receiver frame sync
 feed back out) signal, [9-19](#)
 DIR_LRCLK_REF_O (receiver frame
 sync reference out) signal, [9-19](#)
 DIR_PLLCLK_I (External 512 x FS
 (frame sync) PLL clock input) signal,
 [9-18](#)
 DIT_CLK_I (transmitter serial clock)
 signal, [9-10](#)
 DIT_DAT_I (transmitter serial data)
 signal, [9-10](#)
 DIT_EXT_SYNCEN signal, [9-11](#)
 DIT_FS_I (frame sync input to the
 S/PDIF transmitter) signal, [9-10](#)
 DIT_HFCLK_I (transmitter over
 sampling clock) signal, [9-11](#)
 DIT_O (transmitter biphase encoded
 data stream) signal, [9-11](#)
 feed back out, [9-19](#)
 frame sync, [9-10](#)
 frame sync (LRCLK) input, [9-10](#)
 output routing, [9-8](#)
 oversampling clock, [9-11](#)
 PLL clock input, [9-18](#)
 preambles, [9-7](#)
 programming guidelines, [9-24](#)
 reference clock out, [9-19](#)

S/PDIF *(continued)*

- serial clock input, [9-24](#)
- serial data, [9-10](#)
- single-channel, double-frequency
 - format, [9-8](#)
- SRU control registers, [9-8](#), [9-18](#)
- SRU routing, [9-11](#)
- stream disconnected
 - (DIR_NOSTREAM) bit, [A-95](#)
- subframe format, [9-4](#)
- two channel mode, [9-8](#)

S/PDIF bits

- biphase error (DIR_BIPHASEERROR), [A-95](#)
- buffer enable (DIT_CHANBUF), [9-25](#)
- channel status buffer enable
 - (DIT_CHANBUF), [A-88](#)
- channel status byte 0 A
 - (DIT_BOCHANL), [A-89](#)
- channel status byte 0 B
 - (DIT_BOCHANR), [A-89](#)
- channel status byte 0 for subframe A
 - (DIR_BOCHANL), [A-95](#)
- channel status byte 0 for subframe B
 - (DIR_BOCHANR), [A-95](#)
- disable PLL (DIR_PLLDIS), [A-93](#)
- frequency multiplier (DIT_FREQ), [A-88](#)
- lock error (DIR_LOCK), [A-93](#)
- lock receiver (DIR_LOCK), [A-95](#)
- non-audio frame mode channel 1 and 2
 - (DIR_NOAUDIOLR), [A-95](#)
- non-audio subframe mode channel 1
 - (DIR_NOAUDIOL), [A-95](#)
- parity (DIR_PARITYERROR), [A-95](#)
- parity biphase error (DIR_BIPHASE), [A-93](#)
- receive mute (DIR_MUTE), [A-93](#)

S/PDIF bits *(continued)*

- select single-channel, double-frequency
 - mode channel (DIT_SCDF_LR), [A-88](#)
- serial data input format
 - (DIT_SMODEIN), [A-88](#)
- single channel enable (TX_SCDF_EN), [9-9](#)
- single channel enable left right
 - (TX_SCDF_EN), [9-9](#)
- single channel left right
 - (TX_SCDF_LR), [9-9](#)
- single-channel, double-frequency
 - channel select (DIR_SCDF_LR), [A-93](#)
- single-channel, double-frequency mode
 - enable (DIR_SCDF), [A-93](#)
- transmit mute (DIT_MUTE), [A-87](#)
- transmit single-channel,
 - double-frequency enable
 - (DIT_SCDF), [A-88](#)
- transmitter enable (DIT_EN), [A-87](#)
- user, [9-9](#)
- validity (DIR_VALID), [A-95](#)
- validity bit A (DIT_VALIDL), [A-88](#)
- validity bit B (DIT_VALIDR), [A-88](#), [A-89](#)

S/PDIF registers

- audio data output
 - (SPDIF_RX_DAT_O), [9-18](#)
- bi-phase encoded data (SPDIF_RX_I), [9-18](#)
- channel A transmit status
 - (SPDIF_TX_CHSTA), [A-89](#), [A-90](#)
- channel B transmit status
 - (SPDIF_TX_CHSTB), [A-90](#)
- channel status, [9-12](#)
- control (DITCTL), [9-12](#)
- external frame sync
 - (SPDIF_EXTPCLK_I), [9-18](#)

Index

S/SPDIF registers *(continued)*

- extracted receiver frame sync output (SPDIF_RX_FS_O), [9-18](#)
- extracted receiver sample clock output (SPDIF_RX_CLK_O), [9-18](#)
- receive control (DIRCTL), [9-12](#)
- receiver status (DIRSTAT), [A-94](#)
- receiver TDM output (SPDIF_RX_TDMCLK_O), [9-19](#)
- right channel status for subframe A (DIRCHANL), [A-96](#)
- right channel status for subframe B (DIRCHANR), [A-96](#)
- right channel transmit status (DITCHANR), [9-12](#)
- transmit control (DITCTL), [9-12](#), [9-17](#), [A-86](#)
- transmit status (DITCHANL) left channel, [9-12](#)
- user bit buffer (DITUSRBITAx), [9-12](#)
- sampling clock period, UART, [11-6](#)
- sampling point, UART, [11-6](#)
- SB (UART set break) bit, [11-3](#), [A-118](#)
- SCHEN_A and SCHEN_B (serial port chaining enable) bit, [5-63](#), [A-38](#)
- SDEN (serial port DMA enable) bit, [2-48](#), [5-63](#), [A-38](#)
- SDRAM
 - buffered system, [3-46](#), [3-47](#)
 - bus errors, [3-71](#)
 - core address mapping, [3-52](#)
 - errors, [3-77](#)
 - page size, [3-34](#)
 - refresh rate, [3-50](#)
 - restrictions, [3-52](#)
- SDRAM bits
 - burst stop (NOBSTOP), [A-25](#)
 - CAS latency (SDCL), [A-22](#)
 - column address width (SDCAW), [A-23](#)

SDRAM bits *(continued)*

- disable clock and control (DSDCTL), [A-22](#)
- external data path width (X16DE), [A-24](#)
- force auto refresh (Force AR), [A-24](#)
- force load mode register write (Force LMR), [A-24](#)
- force precharge (Force PC), [A-24](#)
- optimization (SDROPT), [A-27](#)
- optional refresh (SDORF), [A-24](#)
- page size is 128 words (PGSZ 128), [A-25](#)
- pipeline option with external register buffer (SDBUF), [A-24](#)
- power-up mode (SDPM), [A-23](#)
- power-up sequence start (SDPSS), [A-23](#)
- predictive addressing (SDMODIFY), [A-27](#)
- RAS setting (SDTRAS), [A-23](#)
- RDC setting (SDTRCD), [A-24](#)
- refresh delay (RDIV), [A-27](#)
- row address width (SDRAW), [A-25](#)
- RP setting (SDTRP), [A-23](#)
- self-refresh enable (SDSRF), [A-23](#)
- WR setting (SDTWR), [A-24](#)
- SDRAM controller, [3-30](#)
 - address space, external memory, [3-52](#)
 - addressing (16-bit), [3-55](#) to [3-57](#)
 - addressing (32-bit), [3-53](#) to [3-54](#)
 - bank activate, [A-25](#)
 - burst disable, [A-25](#)
 - burst length definition, [3-32](#)
 - burst type, [3-32](#)
 - calculating refresh rate, [3-50](#)
 - CAS latency, [3-32](#)
 - CAS latency (SDCL) bit, [3-37](#)
 - CBR (CAS before RAS), [3-33](#)
 - clock frequencies, [3-37](#)
 - configuring, [3-62](#)
 - control (SDCTL) register, [3-39](#)
 - data mask, [3-33](#)

SDRAM controller *(continued)*

- definitions, [3-31](#) to [3-36](#)
- disable, [3-40](#)
- external data path width, setting, [3-43](#)
- external memory access timing, [3-36](#)
- forcing auto refresh, [3-45](#)
- forcing precharge, [3-45](#)
- internal bank, [3-33](#)
- mode register, [3-33](#)
- optimal data throughput, [3-74](#) to [3-77](#)
- page size, [3-34](#)
- power-up sequence, [A-23](#), [A-24](#)
- power-up sequence start, [3-43](#)
- precharge (SDTRP) bit, [3-37](#)
- RAS to CAS delay (SDTRCD) bit, [3-37](#)
- read/write command, [3-67](#)
- refresh rate (SDRRC) register, [3-49](#)
- row address and page size, [3-34](#)
- row address width, setting, [3-48](#)
- self-refresh, [3-34](#)
- setting bank column address width, [3-42](#), [A-25](#)
- setting CAS latency, [3-40](#)
- status (SDSTAT) register, [3-49](#)
- stop burst, [3-32](#)
- timing, [3-74](#)
- tMRD definition, [3-35](#)
- tRAS definition, [3-35](#)
- tRC definition, [3-36](#)
- tRCD definition, [3-35](#)
- tRFC definition, [3-36](#)
- tRP definition, [3-35](#)
- tRRD definition, [3-36](#)
- tWR definition, [3-35](#)
- tXSR definition, [3-36](#)
- uniprocessor system with multiple
 - SDRAM devices diagram, [3-46](#), [3-47](#)
- write before precharge (SDTWR) bit, [3-37](#)

SDRAM controller commands

- auto-refresh, [3-70](#)
- bank activate, [3-31](#), [3-65](#)
- burst stop, [3-32](#), [3-69](#)
- command pin states, [3-72](#)
- load mode register, [3-64](#)
- NOP/command inhibit, [3-72](#)
- precharge, [3-34](#)
- precharge all, [3-66](#)
- self-refresh, [3-70](#)
- single precharge, [3-66](#)
- stop command, burst, [3-32](#)

SDRAM controller registers, [A-21](#) to [A-26](#)

- control (SDCTL), [A-21](#) to [A-25](#)
- control status (SDSTAT), [A-26](#)
- refresh rate control (SDRRC), [A-26](#)

select ($\overline{\text{MSx}}$) pins, memory, [3-53](#)

self-refresh mode, [3-34](#)

semaphores, [3-92](#)

SENDZ (send zeros) bit, [6-11](#), [6-37](#)

serial clock (SPORTx_CLK) pins, [5-6](#)

serial communications, [11-2](#)

serial inputs, [7-3](#)

serial modes, specifying, [7-5](#)

serial port transmit 4 (SP4I) bit, [B-9](#)

serial word

- endian select (LSBF) bit, [5-62](#)
- length select bits (SLENx) bits, [5-17](#), [5-62](#)

setting CAS latency, [3-40](#)

setting hold time cycles, [3-23](#)

setting the internal serial clock and frame sync rates, [5-21](#)

setting up DMA on SPORT channels, [5-75](#)

setting word length (SLEN) bits, [5-17](#), [5-22](#)

setup time, inputs, [14-32](#)

SFTx (user software interrupt) bits, [B-18](#), [B-21](#), [B-26](#)

Index

shared memory

- See also* external port
- asynchronous access mode, [3-81](#)
- bus arbitration, [3-79](#)
- code select (CSEL) bit, [3-82](#)
- force sync of shared memory bus (FSYNC) bit, [A-8](#)
- interface status, [3-93](#)
- memory select ($\overline{\text{MSx}}$) pins, [3-83](#)
- pins, [3-81](#)
- rotating priority bus arbitration select (RPBA) pin, [3-81](#)
- system design diagram, [3-79](#)
- write ($\overline{\text{WR}}$) pin, [3-83](#)

short-circuit condition, PWM, [8-6](#)

signal naming convention, [4-9](#)

signal routing unit external miscellaneous (MISCAX) registers, [13-13](#)

signal routing unit. *See* SRU, DAI, DPI

signals

- acknowledge (ACK), [3-2](#), [3-19](#), [3-21](#)
- AMI, listed, [3-20](#)
- bidirectional, [4-13](#)
- BLK_START, [9-11](#)
- bus grant $\overline{\text{HBG}}$, [3-89](#)
- bus request $\overline{\text{BRx}}$, [3-79](#), [3-85](#), [3-89](#), [3-92](#)
- CLKIN, [14-20](#)
- clock (serial port), [5-13](#)
- external memory, [3-16](#)
- falling edge, [4-70](#)
- frame sync (serial port), [5-13](#)
- groups in signal routing units, [4-8](#)
- input pins in DAI/DPI, [4-12](#)
- interrupt generation from SRU, [4-69](#)
- memory read $\overline{\text{RD}}$, [3-21](#)
- memory select $\overline{\text{MSx}}$, [3-89](#)
- miscellaneous for general-purpose I/O SRU, [4-65](#)
- mnemonics in signal routing units, [4-9](#)
- output pins in DAI/DPI, [4-11](#)

signals

(continued)

- pin buffers, [4-10](#)
- pin output routing, [4-12](#)
- pin routing, [4-10](#)
- read, [3-25](#)
- responding to, [4-71](#)
- rising edge, [4-70](#)
- rotating priority bus select (RPBA), [3-81](#)
- routing in SRU, [4-18](#)
- sensitivity in serial ports, [5-9](#)
- serial port, [5-5](#) to [5-10](#)
- signal routing group assignments, [4-8](#)
- slave select (SPI), [2-43](#)
- SPDIF bi-phase encoded data input (DIR_I), [9-18](#)
- SPDIF bi-phase output (DIT_O), [9-11](#)
- SPDIF clock (DIT_CLK_I), [9-10](#)
- SPDIF external sync (DIT_EXT_SYNCEN), [9-11](#)
- SPDIF frame sync (DIT_FS_I), [9-10](#)
- SPDIF frame sync feed back out (DIR_LRCLK_O), [9-19](#)
- SPDIF frame sync reference out (DIR_LRCLK_REF_O), [9-19](#)
- SPDIF oversampling clock (DIT_HFCLK_I), [9-11](#)
- SPDIF PLL clock (DIR_PLLCLK_I), [9-18](#)
- SPDIF serial data (DIT_DAT_I), [9-10](#)
- SPI clock (SPICLK), [6-2](#)
- SPI device select ($\overline{\text{SPIDS}}$), [6-2](#)
- structure in signal routing units, [4-8](#)
- write $\overline{\text{WR}}$, [3-21](#)
- single-channel, double-frequency mode (S/PDIF), [9-8](#)
- sizes in SDRAM, page, [3-34](#)
- slave mode DMA operations (SPI), [6-19](#)
- slave mode operation, configure for, [6-11](#)
- SLEN (select word length) bits, [5-17](#), [5-22](#), [5-29](#), [5-62](#)

software interrupt (SFT0x) bit, [B-18](#), [B-21](#), [B-26](#)
 software interrupt x, user (SFTxI) bit, [B-18](#), [B-21](#), [B-26](#)
 software reset (SRST) bit, [A-6](#)
 SOVFI (stack overflow/full) bit, [B-15](#), [B-19](#), [B-23](#)
 SPOI (serial port interrupt) bit, [B-8](#)
 SP2I (serial port interrupt) bit, [B-8](#)
 SP4I (serial port interrupt) bit, [B-9](#)
 SPCNTx (serial port count) registers, [A-45](#)
 SPCTLx (serial port control) registers, [2-26](#), [5-4](#), [5-6](#), [5-7](#), [5-59](#), [5-72](#)
 SPCTLx control bits for left-justify sample pair mode, [5-13](#)
 SPDIF_TX_CHSTA (Sony/Philips digital interface channel status) register, [A-89](#), [A-90](#), [A-91](#)
 SPDIF_TX_CTL (Sony/Philips digital interface transmit control) register, [A-86](#)
 special IDP registers, [A-109](#)
 specifications, timing, [14-28](#)
 SPEN_A (serial port channel A enable) bit, [5-17](#), [5-61](#), [A-37](#)
 SPEN_B (serial port channel B enable) bit, [5-17](#), [5-61](#)
 SPI
 See also SPI bits, registers
 block diagram, [6-3](#)
 broadcast mode, [6-3](#), [6-8](#)
 chaining, DMA, [6-16](#)
 change clock polarity, [6-21](#)
 changing configuration, [6-21](#)
 clock (SPICLK) pin, [6-4](#), [6-5](#), [6-8](#), [6-27](#)
 clock (SPICLK) signal, [6-2](#)
 clock phase, [6-28](#)
 clock rate, [6-5](#)
 clock signal (SPICLK), [6-4](#)
 clock, active edge, defined, [6-5](#)

SPI *(continued)*
 clock, sampling edge, defined, [6-5](#)
 configuring and enabling, [6-15](#)
 data transfer operations, [6-13](#)
 device select signal, [6-6](#)
 DMA, [6-14](#) to [6-27](#), [A-61](#) to [A-65](#)
 DMA, switching from transmit to receive mode, [6-23](#)
 error signals and flags, [6-35](#)
 features, [6-1](#)
 finished (SPIF) bit, [6-30](#)
 FLAGx pins, [6-4](#)
 formats, [6-35](#)
 functional description, [6-2](#)
 general operations, [6-8](#)
 interface signals, [6-4](#)
 interrupts, [6-14](#), [6-17](#), [6-33](#)
 master input slave output (MISOx) pins, [6-2](#), [6-7](#)
 master mode operation, configuring for, [6-10](#)
 master out slave in (MOSIx) pins, [6-2](#), [6-7](#), [6-8](#)
 master-slave interconnections, [6-4](#)
 MISOx (master in, slave out) pins, [6-7](#)
 multidevice configuration, [6-12](#)
 multimaster environment, [6-8](#)
 multimaster error or mode-fault error (MME) bit, [6-9](#)
 open drain output enable (OPD) pin, [6-9](#)
 operation, master mode, [6-15](#)
 operation, slave mode, [6-19](#)
 operations, [6-8](#), [6-10](#)
 packed data transfers, [6-32](#)
 programming examples, [6-38](#)
 receive buffer register (RXSPI, RXSPIB), [2-26](#)
 receive data (RXSPI) buffer, [6-2](#), [6-10](#)
 registers, [A-52](#) to [A-65](#)

Index

- send zero (SENDZ) bit, [6-11](#), [6-37](#)
- serial peripheral interface clock (SPICLK) signal, [6-4](#)
- slave mode, [6-11](#), [6-20](#)
- slave mode operation, [6-11](#)
- slave select outputs (SPIDS0-3), [6-6](#)
- SPIDS pin, [6-6](#), [6-9](#), [6-11](#), [6-19](#), [A-54](#)
- switching from receive to transmit mode, [6-23](#), [6-24](#)
- system, configuring and enabling bits, [6-16](#), [A-52](#)
- transfer formats, [6-27](#)
- transfer, beginning and ending, [6-29](#)
- transfers, data, [6-30](#)
- transmit data (TXSPI) buffer, [6-2](#)
- transmit underrun error (SPIUNF) bit, [6-25](#), [6-26](#), [6-34](#)
- TXFLSH (flush transmit buffer) bit, [6-23](#), [A-55](#)
- unpacking data, [6-33](#)
- word lengths, [6-31](#)
- SPI bits
 - baud rate enable (BAUDR), [A-60](#)
 - chain loading status (SPICHS), [A-64](#)
 - chained DMA enable (SPICHEN_A and SPICHEN_B), [6-16](#), [A-63](#)
 - clock phase (CPHASE), [A-55](#)
 - clock polarity (CLKPL), [6-5](#), [6-27](#), [A-55](#)
 - device select control (SPIFLGx3-0), [A-58](#)
 - device select enable (DSxEN), [6-15](#)
 - DMA control, [A-63](#)
 - DMA interrupt enable (INTEN), [6-34](#)
 - enable (SPIEN), [A-55](#)
 - enable interrupt on error (INTERR), [6-34](#)
 - external transaction complete (SPIFE), [A-57](#)
 - flag select (FLGx), [6-10](#)
 - flush receive buffer (RXFLSH), [6-23](#), [6-25](#), [A-55](#)
 - flush transmit buffer (TXFLSH), [A-55](#)
 - get more data (GM), [6-11](#), [6-37](#)
 - input service select (ISSS), [A-58](#)
 - input slave select (ISSEN), [A-54](#)
 - input slave select enable (ISSEN), [6-36](#)
 - internal loop back (ILPBK), [A-56](#)
 - low priority interrupt (SPILI), [6-34](#)
 - master select (SPIMS), [A-55](#)
 - MISO disable (DMISO), [A-54](#)
 - mode-fault error (MME), [6-35](#), [A-57](#)
 - most significant byte first (MSBF), [A-54](#)
 - multimaster error (SPIMME), [6-35](#)
 - open drain output select (OPD), [A-55](#)
 - packing enable (PACKEN), [A-55](#)
 - reception error (ROVF), [6-34](#), [6-35](#), [6-37](#), [A-57](#)
 - seamless transfer (SMLS), [A-55](#)
 - send zero (SENDZ), [A-54](#)
 - sign-extend (SGN), [A-55](#)
 - transfer finished (SPIF), [6-30](#)
 - transfer initiation mode (TIMOD), [A-54](#)
 - transmission error (TUNF), [6-35](#), [6-37](#), [A-57](#)
 - transmit collision error (TXCOL), [6-37](#), [A-57](#)
 - transmit data buffer status (TXS), [6-15](#), [6-21](#), [6-30](#), [A-57](#)
 - transmit underrun error (SPIUNF), [A-63](#)
 - transmit underrun error (SPIUNFE), [6-35](#)
 - word length (WL), [A-54](#)
- SPI master booting, [14-37](#)
- SPI receive DMA interrupt mask (SPILIMSK) bit, [B-11](#)
- SPI receive DMA interrupt mask pointer (SPILIMSKP) bit, [B-12](#)

Index

SPI registers

- baud rate (SPIBAUDx), [A-60](#)
- DMA address modify (IMSPI), [A-64](#)
- DMA chain pointer (CPSPI), [A-65](#)
- DMA configuration (SPIDMAC), [6-14](#), [6-16](#), [6-19](#), [6-22](#), [6-23](#), [6-35](#), [A-62](#)
- DMA start address (IISPI), [A-64](#)
- DMA word count (CSPI), [A-64](#)
- flag (SPIFLGx), [A-58](#)
- interrupt (LIRPTL), [6-34](#)
- interrupt latch/mask (LIRPTL), [6-34](#)
- receive buffer (RXSPI), [2-26](#)
- receive control (SPICCTL, SPICTLB), [2-26](#), [A-52](#)
- RXSR (SPI receive shift), [6-2](#)
- SPIBAUD (baud rate) register, [6-5](#), [A-60](#)
- status (SPISTAT), [6-9](#), [6-17](#), [6-22](#), [6-24](#), [6-25](#), [6-35](#), [6-36](#), [A-56](#), [A-59](#)
- status (SPISTAT, SPISTATB), [A-56](#)
- transmit buffer (TXSPI), [6-10](#), [6-37](#), [A-59](#), [A-60](#)
- TXSR (SPI transmit shift), [6-2](#)

SPI slave booting, [14-37](#)

SPIBAUDx (SPI baud rate) registers, [A-60](#)

SPICHEN_A and SPICHEN_B (SPI DMA chaining enable) bits, [6-16](#), [6-19](#), [A-38](#)

SPICLK (serial peripheral interface) timing, [6-5](#)

SPICLK (SPI clock) pins, signals, [6-2](#), [6-4](#), [6-27](#)

SPICCTL (SPI port control) registers, [2-26](#), [A-52](#)

SPIDMAC (SPI DMA control) register, [6-14](#), [6-35](#), [A-62](#)

SPIDS (SPI device select) pin, [6-27](#)

SPIEN (SPI enable) bit, [2-48](#)

SPIF (SPI transfer finished) bit, [6-30](#)

SPIFLGx (SPI device select control) bits, [A-58](#)

SPII (SPI low priority interrupt) bit, [6-34](#)

SPIMME bit, [6-35](#)

SPIOVF (SPI receive overflow error) bit, [6-25](#), [6-26](#), [6-34](#), [6-35](#)

SPISTAT, SPISTATB (SPI status) registers, [6-35](#), [A-56](#), [A-59](#)

SPIUNF (SPI transmit underrun error) bit, [6-25](#), [6-26](#), [6-34](#), [6-35](#)

SPORT bits

- B channels enable (MCEB), [A-43](#)
- chained DMA enable (SCHEN), [5-63](#), [A-38](#)
- channel A enable (SPEN_A), [5-17](#), [5-61](#), [A-37](#)
- channel B enable (SPEN_B), [5-17](#), [5-61](#)
- channel error status (ROVF_A or TUVF_A), [A-40](#)
- clock rising edge select (CKRE), [5-62](#)
- clock, internal (ICLK), MSTR (I²S mode only), [A-37](#)
- control (SPCTLx), [5-22](#)
- current channel selected (CHNL), [A-43](#)
- data buffer channel A/B status (DXS_A), [A-40](#)
- data direction control (SPTRAN), [5-4](#), [5-64](#)
- data independent transmit/receive frame sync (DIFS), [A-38](#)
- DERR_B error status (ROVF_B or TUVF_B), [A-39](#)
- DMA chaining enable (SCHEN_x), [5-63](#), [A-38](#), [A-39](#)
- DMA chaining status (DMACHSxy), [A-43](#)
- DMA enable (SDEN), [2-48](#), [5-63](#), [A-38](#)
- DMA status (DMASxy), [A-43](#)
- DXS_B (data buffer status), [A-39](#)
- enable (SPEN_x), [5-61](#)
- frame on rising frame sync (FRFS), [5-17](#), [A-38](#)

Index

SPORT bits *(continued)*

- frame sync both (FS_BOTH), [5-64](#), [A-39](#)
- frame sync required (FSR), [A-38](#)
- FS both enable (FS_BOTH), [A-39](#)
- internal clock select (ICLK), [5-62](#), [A-37](#)
- internal frame sync select (IFS), [5-63](#), [A-38](#)
- late frame sync (LAFS), [A-38](#)
- left-justified sample pair mode control, [5-17](#)
- loopback mode (SPL), [A-42](#)
- multichannel frame delay (MFD), [A-42](#)
- multichannel mode enable (MCEA), [A-42](#)
- number of multichannel slots (NCH), [A-42](#)
- operation mode (OPMODE), [5-17](#)
- packing enable (PACK), [A-37](#)
- receive underflow status (DERR_A, ROVF_A or TUVF_A), [A-40](#)
- serial word endian select (LSBF), [5-62](#), [A-37](#)
- serial word length (SLEN), [5-17](#), [5-62](#)
- serial word length select (SLEN), [A-37](#)
- transmit underflow status (TUVF_A), [5-65](#), [A-40](#)
- word packing enable (packing 16-bit to 32-bit words) PACK, [5-62](#)

SPORT modes

- I²S, [5-10](#), [5-20](#)
- I²S (Tx/Rx on left channel first), [5-11](#), [5-12](#), [A-30](#)
- I²S (Tx/Rx on right channel first), [5-11](#), [5-12](#), [A-30](#)
- left-justified sample pair, [5-12](#), [5-16](#), [5-18](#), [5-19](#), [A-30](#)
- multichannel, [5-3](#), [5-25](#)
- multichannel, A and B channels, [5-12](#), [A-30](#)

SPORT modes *(continued)*

- operation mode (OPMODE) bit, [5-62](#)
- standard DSP, [5-11](#), [A-30](#)

SPORTs

- See also* SPORT bits, modes, registers
- 128-channel TDM, [5-4](#)
- 16-bit to 32-bit word packing enable (PACK), [5-62](#)
- bidirectional functions, [5-1](#)
- buffer, DMA, [5-15](#)
- buffers, using, [5-7](#)
- clock (SCLKx) pins, [5-6](#)
- companding (compressing/expanding), [5-2](#)
- configuring frame sync signals, [5-6](#)
- configuring standard DSP serial mode, [5-13](#)
- connections, [5-6](#)
- control (SPCTLx) registers, [5-59](#)
- DAI pin routing, [5-5](#)
- data buffers, [5-4](#)
- data types, [5-46](#)
- debugging, [A-42](#)
- disabling the serial port(s), [5-72](#)
- DMA chaining, [5-81](#)
- DMA channels, [5-73](#), [5-74](#)
- enabling B channels, [A-43](#)
- enabling I²S mode (OPMODE), [5-17](#), [5-22](#)
- enabling master mode (MSTR), [5-18](#)
- enabling SPORT DMA (SDEN), [5-19](#)
- features, [5-2](#)
- finding currently selected channel, [A-43](#)
- frame sync rates, setting the internal serial clock, [5-21](#)
- full-duplex operation, [5-6](#)
- I/O processor bus, [5-44](#)
- I²S control bits, [5-21](#)
- internal serial clock setting, [5-17](#)
- interrupts, [5-72](#), [5-75](#)

SPORTs *(continued)*

- interrupts, priority of, [5-72](#)
- latency in writes, [5-58](#)
- operation modes, [5-10](#), [5-11](#), [5-59](#)
- pairing, [5-27](#)
- primary and secondary data buffers, [5-4](#)
- pulse code modulation (PCM), [5-20](#)
- registers, listed, [5-50](#) to [5-58](#)
- reset, [5-71](#)
- serial clock pins, [5-6](#)
- setting frame sync rates, [5-17](#)
- setting word length, [5-17](#), [5-22](#)
- signal sensitivity, [5-9](#)
- signals, [5-6](#)
- SPORTx_DA and SPORTx_DB
 - channel data signal, [5-6](#)
- SPORTx_FS (serial port frame sync)
 - pins, [5-6](#)
- SPxI (serial port interrupt priority) bit, [5-72](#)
- standard DSP serial operation mode, [5-12](#)
- timing, left-justified sample pair mode, [5-19](#)
- timing, word select timing in I²S mode, [5-25](#)
- transferring data words, [5-4](#)
- transmit and receive data buffers, [5-4](#)
- Tx/Rx on FS falling, rising edge, [5-12](#), [A-30](#)
- Tx/Rx on FS rising edge, [5-12](#)
- using, [5-7](#)
- using with SRU, [5-5](#)
- word length, [5-43](#)
- word select timing in left-justified sample pair mode, [5-19](#)

SPORT's registers

- control (SPCTLx), [2-26](#), [5-4](#), [5-6](#), [5-7](#), [5-59](#), [5-72](#)
- count (SPCNTx), [A-45](#)

SPORT's registers *(continued)*

- divisor (DIVx), [5-6](#)
- DMA parameter, [5-76](#)
- modify (IMSPx), [A-50](#)
- receive buffer (RXSPx), [A-44](#)
- SPCTLx (serial port control), [A-29](#)
- transmit buffer (TXSPx), [A-43](#)
- transmit compand (MTxCSx, MTxCCSx), [A-47](#)

SPTRAN (serial port data direction control) bit, [A-39](#)

sample rate converter. *See* SRC

SRC

- AD1896 core use with, [10-1](#)
- block diagram, [10-9](#)
- clocking, [10-15](#)
- configuring modes, [10-13](#), [10-21](#)
- control (SRCCTLx) register, [10-13](#), [10-21](#)
- data paths, [10-19](#)
- data ports and, [10-13](#), [10-21](#)
- de-emphasis (DEEMPHASIS) bits, [10-18](#)
- FIR filter, [10-2](#), [10-7](#), [10-10](#)
- frame sync signal, [10-18](#), [10-22](#)
- I²S, [10-14](#)
- input formats, [A-99](#), [A-104](#)
- MCLK (master clock), [10-2](#)
- mute (MUTE_OUT/MUTE_IN)
 - signals, [10-19](#)
- mute (SRCMUTE) register, [10-19](#), [10-21](#)
- muting, [10-19](#)
- normal, slow, fast modes, [10-11](#)
- parallel load shift register, [10-15](#)
- programming, [10-22](#)
- ratio (SRCRAT) register, [10-19](#), [10-21](#)
- registers, described, [10-21](#)
- right justified mode, [10-14](#)
- right-justified mode, [10-14](#), [10-18](#)

Index

SRC *(continued)*

- sample rate ratio, [10-2](#), [10-19](#)
- sample rates, input, [10-18](#)
- servo loop, [10-2](#)
- time division multiplexing mode, [10-16](#), [10-18](#)
- tracking input and output rates, [10-2](#)
- SRC bits
 - auto hard mute
 - (SRCx_AUTO_MUTE), [A-98](#), [A-103](#)
 - bypass SRC (SRCx_BYPASS), [A-99](#), [A-104](#)
 - de-emphasis filter select
 - (SRCx_DEEMPHASIS), [A-99](#), [A-104](#)
 - dither select (SRCx_DITHER), [A-99](#), [A-104](#)
 - hard mute (SRCx_HARD_MUTE), [A-98](#), [A-103](#)
 - match phase mode select
 - (SRCx_MPHASE), [A-100](#), [A-105](#)
 - output word length (SRCx_LENOUT), [A-100](#), [A-105](#)
 - serial input format (SRCx_SMODEIN), [A-99](#), [A-104](#)
 - serial output format
 - (SRCx_SMODEOUT), [A-99](#), [A-104](#)
 - soft mute (SRCx_SOFTMUTE), [A-99](#), [A-104](#)
 - SRC enable (SRCx_ENABLE), [A-100](#)

SRU

- bidirectional pin buffer, [4-13](#)
- buffers, [4-13](#)
- connecting peripherals with, [4-8](#)
- connecting through, [4-15](#)
- connection to precision clock generator (PCG), [13-1](#)
- group A (clock) signals, [4-19](#), [4-52](#)
- group B (data) signals, [4-25](#)

SRU *(continued)*

- group C (frame sync) signals, [4-31](#)
- group D (pin assignments) signals, [4-36](#)
- group E (miscellaneous) signals, [4-43](#) to [4-46](#)
- group F signals, [4-47](#)
- inputs, [4-8](#)
- mnemonics, [4-9](#)
- naming conventions, [4-9](#)
- outputs, [4-8](#)
- register groups, [4-17](#)
- register use of, [4-15](#)
- serial ports and, [5-5](#)
- signal groups, [4-8](#), [4-18](#)
- SPORT connection example, [4-14](#), [4-16](#)
- SRU_CLKx (SRU clock) registers, [7-18](#), [7-21](#)
- SRU_DATx (SRU data) registers, [7-18](#), [7-21](#)
- SRU_PINGx_STAT (ping-pong DMA status) register, [A-110](#)
- SRU2
 - default configuration, [4-51](#)
 - group A (input routing) signals, [4-52](#)
 - group B (pin assignment) signals, [4-56](#)
 - group C (pin enable) signals, [4-60](#)
- stack overflow/full interrupt (SOVFI) bit, [B-15](#), [B-19](#), [B-23](#)
- stall cycles, in IOP register access, [2-5](#)
- stalls, core, [14-54](#)
- stalls, execution, [14-53](#) to [14-56](#)
- standard DSP serial mode, [5-12](#)
- starting an interrupt driven transfer, [7-18](#), [7-20](#), [7-23](#)
- STATDAx (data memory breakpoint hit) bit, [A-181](#)
- STATIO (I/O address breakpoint hit) bit, [A-181](#)
- STATIx (instruction address breakpoint hit) bit, [A-181](#)

- STATPA (program memory data breakpoint hit) bit, [A-181](#)
- STATUS field, [11-10](#)
- strobe period, [13-13](#)
- strobe pulse, [13-13](#)
- strobe, PDAP output, [7-14](#)
- STROBEA (one shot frame sync A) bit, [13-13](#), [A-159](#)
- STROBEB (one shot frame sync B) bit, [13-13](#), [A-159](#)
- supervisory circuits, [14-27](#)
- support, technical or customer, [xxxv](#)
- switching frequencies
 - determining, [14-29](#)
- switching from receive to transmit DMA (SPI), [6-24](#)
- switching from transmit to receive DMA (SPI), [6-23](#)
- synchronization with the external clock, [13-7](#)
- synchronizing frame sync output, [13-7](#)
- synchronous access mode, [3-81](#)
- synchronous access mode (external port), [3-22](#)
- SYSCTL (system control) register, [A-5](#)
- SYSCTL register
 - bus lock request (BUSLK) bit, [A-8](#)
 - external port data pin mode select (EPDATA) bits, [A-8](#)
 - force sync of shared memory bus (FSYNC) bit, [A-8](#)
 - internal interrupt vector table (IIVT) bit, [A-6](#)
 - internal memory data width (IMDWx) bits, [A-7](#)
 - interrupt request enable (IRQxEN) bits, [A-7](#)
 - memory select (MSEN) bit, [A-8](#)
 - pulse width modulation select (PWMx) bits, [A-8](#)
- SYSCTL register *(continued)*
 - rotating priority bus arbitration (RBPR) bit, [A-7](#)
 - SRST (software reset) bit, [A-6](#)
 - timer (flag) expired mode (TMREXPEN) bit, [A-8](#)
- system control register. *See* SYSCTL register
- system design
 - baud rate, init value, [14-49](#)
 - boot configuration (BOOT_CFGx) pins, [14-38](#)
 - bypass capacitors, [14-35](#)
 - CLKIN pin, [14-13](#), [14-20](#)
 - CLKOUT and CCLK clock generation, [14-30](#)
 - clock distribution, [14-34](#)
 - clock input, [14-20](#)
 - conditioning input signals, [14-32](#)
 - crosstalk, reducing, [14-34](#)
 - decoupling capacitors, [14-35](#)
 - designing for high frequency operation, [14-33](#)
 - determining switching frequencies, [14-29](#)
 - flags (FLAGx) pins, [14-8](#)
 - FLAGx pins, [14-8](#)
 - generators, reset, [14-28](#)
 - ground plane, [14-34](#)
 - hold time, inputs, [14-32](#)
 - input setup and hold time, [14-32](#)
 - input signal conditioning, [14-32](#)
 - JTAG interface pins, [14-12](#)
 - latchup, [14-32](#)
 - latency, input synchronization, [14-8](#)
 - pins, descriptions, [14-2](#)
 - plane, ground, [14-34](#)
 - PLL-based clocking, [14-13](#)
 - power supply, monitor and reset generator, [14-28](#)

Index

system design *(continued)*
 recommendations and suggestions,
 [14-34](#)
 RESET pin, [14-33](#)
 shared system diagram, [3-79](#)
 stalls, [14-54](#)
 switching frequencies, [14-29](#)
 timing specifications, [14-28](#)
system status (SYSTAT) register, [A-9](#)

T

TCB chain loading, [2-15](#), [2-16](#)
TCK (test clock) pin, [14-12](#)
TDI (test data input) pin, [14-12](#)
TDM mode, time division multiplexed,
 [9-19](#)
TDO (test data output) pin, [14-12](#)
technical or customer support, [xxxv](#)
technical publications online or on the web,
 [-xxxix](#)
TFSDIV (frame sync divisor) bit, [A-45](#)
THR register empty (THRE) flag, [11-4](#),
 [11-13](#)
time division multiplexed (TDM) mode,
 [5-25](#), [5-27](#), [9-19](#), [10-15](#), [10-19](#)
timeout, bus mastership, [3-87](#)
timer expired (TIMEXP) pin, [14-8](#)
timers, UART, [11-1](#)
timing
 definitions, [14-29](#)
 external memory accesses, [3-36](#)
 IDP hold timing mode 00, [7-13](#)
 IDP hold timing mode 01, [7-14](#)
 IDP I²S, [7-7](#)
 IDP left-justified sample pair, [7-7](#)
 PDAP, [7-14](#)
 SDRAM, [3-74](#)
 specifications, system design, [14-28](#)
 SPI clock, [6-5](#)
 SPI transfer protocol, [6-28](#), [6-29](#)

timing *(continued)*
 SPORT framed vs. unframed data, [5-40](#)
 SPORT left-justified sample pair mode,
 [5-19](#)
 SPORT normal vs. alternate framing,
 [5-40](#)
 SPORT word select, [5-25](#)
TIMOD (SPI transfer initiation mode) bit,
 [6-10](#), [6-34](#)
TMS (test mode select) pin, [14-12](#)
TMZHI (timer expired high priority) bit,
 [B-15](#), [B-19](#), [B-23](#)
TMZLI (timer expired low priority) bit,
 [B-17](#), [B-21](#), [B-25](#)
transfer control block (TCB), [2-16](#)
transfer initiation and interrupt (SPI
 TIMOD) mode, [6-34](#)
transferring data words, [5-4](#)
transmission error (SPI TUNF) bit, [6-37](#)
transmit and receive channel order (FRFS),
 [5-18](#), [5-23](#)
transmit and receive data buffers
 (TXSPxA/B, RXSPxA/B), [5-67](#)
transmit collision error (SPI TXCOL) bit,
 [6-37](#)
transmit data (SPI TXSPI) buffer, [6-2](#)
transmit frame sync divisor. *See* TFSDIV
 bit
transmit shift register (SPI TXSR), [6-2](#)
 $\overline{\text{TRST}}$ (test reset) pin, [14-12](#)
TUNF (SPI transmission error) bit, [6-37](#)
TUVF_A (channel error status) bit, [5-65](#),
 [A-40](#)
TWI controller
 architecture, [12-2](#)
 block diagram, [12-3](#)
 bus arbitration, [12-12](#)
 call address, [12-14](#)
 clocking, [12-11](#)
 fast mode, setting, [12-14](#)

TWI controller *(continued)*

- little endian word order, [12-9](#)
- prescale register, [12-4](#)
- programming examples, [12-15](#)
- start and stop conditions, [12-13](#)
- transferring data, [12-10](#)

TWI controller bits

- address not acknowledged (TWIANAK), [A-141](#)
- buffer write error (TWIWERR), [A-142](#)
- clock high (TWICKLHI), [A-133](#)
- clock low (TWICKLOW), [A-133](#)
- data not acknowledged (TWIDNAK), [A-141](#)
- data transfer count (TWIDCNT), [A-138](#)
- enable (TWIEN), [12-4](#), [A-132](#)
- fast mode (TWIFAST), [A-137](#)
- general call enable (TWIGCE), [A-134](#)
- issue stop condition (TWISTOP), [A-137](#)
- lost arbitration (TWILOSE), [A-141](#)
- master address length (TWIMLEN), [A-137](#)
- master mode enable (TWIMEN), [A-137](#)
- master transfer direction (TWIMDIR), [A-137](#)
- master transfer in progress (TWIMPROG), [A-141](#)
- not acknowledged (TWINAK), [A-134](#)
- repeat START (TWIRSTART), [A-137](#)
- serial clock override (TWISCLOVR), [A-138](#)
- serial clock sense (TWISCLSEN), [A-142](#)
- serial data override (TWISDAOVR), [A-138](#)
- serial data sense (TWISDASEN), [A-142](#)
- slave address length (TWISLEN), [A-134](#)
- slave enable (TWISEN), [A-134](#)

- slave transmit data valid (TWIDVAL), [A-134](#)

TWI controller registers

- clock divider (TWIDIV), [12-5](#), [A-132](#)
- RXTWI16 (16-bit receive FIFO)
 - register, [12-10](#)
- RXTWI8 (8-bit receive FIFO), [12-9](#)
- TWIFIFOCTL (FIFO control), [12-7](#)
- TWIFIFOSTAT (FIFO status), [12-7](#)
- TWIIMASK (interrupt mask), [12-8](#)
- TWIIRPTL (interrupt), [12-7](#)
- TWIMADDR (master mode address), [12-6](#), [A-139](#)
- TWIMCTL (master mode control), [12-6](#), [A-136](#)
- TWIMSTAT (master mode status), [12-7](#), [A-140](#)
- TWISADDR (slave mode address), [12-6](#), [A-135](#)
- TWISCTL (slave mode control), [A-133](#)
- TWISSTAT (slave mode status), [12-6](#), [A-135](#)
- TXTWI16 (16-bit transmit FIFO), [12-8](#)
- TXTWI8 (8-bit transmit FIFO), [12-8](#)
- two channel mode (SPDIF), [9-8](#)
- TX_UACEN (DMA transmit buffer enable) bit, [11-12](#)
- TXCOL (SPI transmit collision error) bit, [6-37](#)
- TXFLSH (flush SPI transmit buffer) bit, [6-23](#), [A-55](#)
- TXS_A (data buffer channel B status) bit, [A-40](#)
- TXSPI, TXSPIB (SPI transmit buffer)
 - registers, [2-26](#), [6-10](#), [6-13](#), [6-37](#), [A-59](#), [A-60](#)
- TXSPx (serial port transmit buffer)
 - registers, [2-26](#), [A-43](#)

U

UART, [11-1](#)

- assigning interrupt priority, [11-11](#)
- baud rate, [11-4](#), [11-5](#)
- baud rate examples, [11-12](#)
- data ready flag, [11-13](#)
- data word, [11-4](#)
- divisor, [11-11](#), [A-125](#)
- divisor reset, [11-12](#)
- DMA channel latency requirement, [2-44](#)
- DMA channels, [2-44](#)
- DMA mode, [2-44](#)
- interrupt conditions, [11-9](#)
- non-DMA mode, [11-13](#)
- restrictions, [11-11](#)
- sampling clock period, [11-6](#)
- sampling point, [11-6](#)
- standard, [1-10](#), [11-1](#)
- system DMA and, [11-7](#)
- timers, [11-1](#)

UART bits

- 9-bit RX enable (RX9), [A-127](#)
- 9-bit TX enable (TX9), [A-127](#)
- address detect enable (UARTAEN), [A-127](#)
- data ready (DR), [11-4](#)
- DMA TX/RX control, [A-128](#)
- DMA TX/RX status, [A-129](#)
- enable receive buffer full interrupt (UARTBFIE), [11-7](#), [A-123](#)
- enable transmit buffer empty interrupt (UARTTBEIE), [11-7](#), [A-123](#)
- interrupt enable, [A-123](#)
- packing enable (PACK), [A-127](#)
- pin status (UARTPSTx), [A-127](#)
- synch data packing in RX (UARTPKSYN), [A-127](#)
- THR register empty (UARTTHRE), [11-4](#), [A-120](#)

UART bits *(continued)*

- TSR and UARTxTHR empty (UARTTEMT), [11-4](#)
 - UARTNOINT (pending interrupt), [A-124](#)
 - UARTSTAT (interrupt), [A-124](#)
- ### UART registers
- divisor latch (UARTxDLH), [11-11](#), [A-125](#)
 - divisor latch register (UARTxDLL), [11-11](#), [A-125](#)
 - DMA control, [2-27](#)
 - interrupt enable register (UARTxIER), [11-7](#), [A-123](#)
 - interrupt identification register (UARTxIIR), [11-9](#), [A-124](#)
 - line control register (UARTxLCR), [11-3](#), [A-118](#)
 - line status register (UARTxLSR), [11-4](#), [A-120](#)
 - receive buffer register (UARTxRBR), [11-5](#), [A-122](#)
 - scratch register (UARTxSCR), [11-12](#), [A-126](#)
 - shadow, [A-122](#)
 - transmit holding (UARTxTHR), [11-4](#), [11-5](#), [A-121](#)
 - transmit shift register (UART_TSR), [11-4](#)
 - UARTxDLH (divisor latch register), [11-11](#), [A-125](#)
 - UARTxDLL (divisor latch register), [11-11](#), [A-125](#)
 - UARTxIER (interrupt enable register), [11-7](#), [A-123](#)
 - UARTxIIR (interrupt identification register), [11-9](#), [A-124](#)
 - UARTxLCR (line control register), [11-3](#), [A-118](#)

UART registers *(continued)*

UARTxLSR (line status register), [11-4](#),
[A-120](#)

UARTxRBR (receive buffer register),
[11-5](#)

UARTxSCR (scratch register), [11-12](#),
[A-126](#)

UARTxTHR (transmit holding register),
[11-4](#), [11-5](#), [A-121](#)

UARTxTSR (transmit shift register),
[11-4](#)

UARTBI (UART break interrupt), [11-9](#)

UARTFE (UART framing error), [11-9](#)

UARTOE (UART overrun error), [11-9](#)

UARTPE (UART parity error), [11-9](#)

UMODE (user mode breakpoint) bit,
[A-175](#)

user mode breakpoint (UMODE), [A-175](#)

W

wait states, enabling (WS bit), [A-18](#)

word length, [5-43](#)

word length (SLEN) bits, [5-17](#), [5-29](#)

word packing enable (packing 16-bit to
32-bit words) PACK bit, [5-62](#), [A-37](#)

word select timing in I²S mode, [5-25](#)

write (\overline{WR}) pin, [3-21](#), [3-83](#)

